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 is necessary 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 [11, 14, 17]. 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 [15] and Chord [18], 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 [2]. 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 [3, 9, 11], 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 [8] 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 [6] 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 PlanetLab. In Section 4, we describe the design of our resilient transport layer and argue for its correctness. Section 5 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 [5] 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 [12] 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. [2] 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 [11], 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 [9]. 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 [3].

3 Routing Anomalies

In order to demonstrate the extent to which routing anomalies have become a problem, we examined data collected from the PlanetLab Internet testbed collected over 10 days in September of 2004. PlanetLab consists 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 of the course of the run, generously made available by Jeremy Stribling [19].

In order to investigate the impact of routing anomalies, we limited our evaluation to nodes which were online and were 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 increasing frequency of ping failures. The right graph just shows pairs who were never successful in pinging. The data we used for both graphs was collected between September 1 and September 10, 2004.

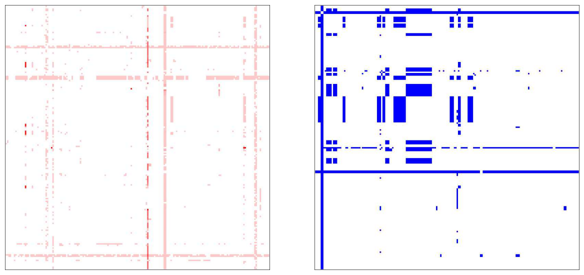
Figure 1. Transient (left) and permanent (right) path failures in PlanetLab during 10 days in September 2004

These results clearly show that routing anomalies are a persistent problem in the general Internet. 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, but the distribution is heavy-tailed; a full 35% of the outages were present for less than one hour, while 9% persisted for more than a day. Figure 2 shows a cumulative distribution.

A good number of the anomalies occurred between seemingly random nodes. However, there are some nodes with a high number of permanent failures. In our experiment, this is explained by the presence of some Internet-2 nodes, which do not have a direct IP-to-IP connection to the classical Internet; nodes behind NATs and firewalls would have similar characteristics. While our analysis mainly focuses on transient path failures, our solution also works for networks with a moderate number of permanent failures.

4 A Resilient Transport Layer

In this section, we present the design of a maintenance protocol that is resilient against path failures. For concreteness, we describe our protocol in the context of FreePastry [6], although we are confident that it can be applied to other overlays as well. We first describe the distributed algorithm that is used in Pastry to implement consistent hashing, and we define requirements for making this algorithm resilient against path failures. Then we

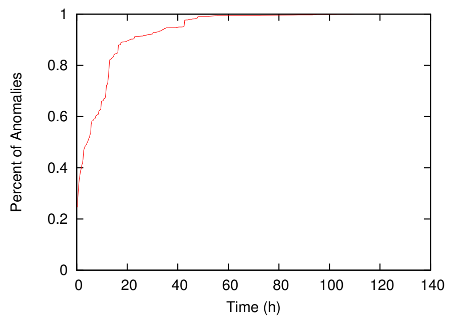

Figure 2. CDF of routing anomaly durations for 192 PlanetLab nodes for September 1 to September 10, 2004.

present a number of techniques used in our new maintenance protocol and argue that in combination, these techniques make the occurrence of inconsistencies extremely unlikely.

4.1 Consistent Hashing

Like many other overlays [18], Pastry uses *consistent hashing* to map keys from a large key space of range $0..2^k$ to a set of nodes. Each node N_i is assigned a unique nodeId k_i from the key space and is responsible for all keys to which k_i is the numerically closest nodeId.

To determine the range R_i of keys for which it is responsible, it is sufficient for a node N_i to know its direct neighbors N_{i-1} and N_{i+1} in the key space. The range R_i can then be computed as

$$R_i = \left[\frac{k_{i-1} + k_i}{2} \bmod 2^k, \frac{k_i + k_{i+1}}{2} \bmod 2^k \right)$$

This range is dynamically updated as nodes join or leave the overlay.

4.2 Routing in Pastry

Pastry provides a routing primitive which delivers messages to the node that is currently responsible for a given key. For this purpose, each Pastry node N_i maintains two data structures: A *leaf set* of up to l neighbors on both sides, and a *routing table* with links to more distant nodes. The former is used to determine the range R_i and must therefore be kept consistent, while the latter is used as an optimization. Since inconsistencies in the routing table merely increase the routing overhead, the following discussion focuses on the leaf sets only.

When a new node joins a Pastry overlay, it discovers its leaf set by routing a message to its own nodeId and then announces its presence to its neighbors. From that point on, it periodically checks the leaf set for faulty nodes and replaces them with other neighbors. In the unlikely event that all nodes on one side of the leaf set fail simultaneously, the node can recover by executing a more expensive repair operation [10].

4.3 Requirements

As long as Pastry's leaf sets accurately reflect the current overlay membership, messages are always accepted by the correct node. However, node failures and path failures may cause the leaf sets to become inconsistent. In this case, our goal is to maintain the following invariant, which we call *routing consistency*, with high probability[1]:

For each time t and key k, there is at most one node which will accept messages for k.

This invariant is guaranteed by the overlay in the case of no path failure, and numerous applications rely on it in order to perform serialization of operations, mutual exclusion, and so forth. Routing consistency is enforced by the *leaf set stabilization protocol*, which has the following functions:

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, it must ensure that the leaf sets eventually reflect a consistent view of overlay membership again.

The current protocol used in FreePastry performs these functions reliably only if the underlying network is fully connected; path failures can cause it to oscillate or deliver messages incorrectly. In the following, we describe a set of techniques that relax this constraint. Our new protocol assumes only that leaf sets are always *connected*, i.e. that for each pair of leaf set members A and B, there exists a path of leaf set nodes A, N_1, ... N_k, B such that each is directly connected to the next, and A is therefore able to send messages to B along this path. Since the size of the leaf set is usually chosen between 8 and 24, we believe this is a reasonable assumption.

[1] It is easy to show, using an adversarial argument, that *no* protocol can maintain the invariant in *all* failure scenarios.

4.4 Virtual links

If a path $A \to B$ fails, messages sent from A to B may silently disappear in the network. This not only increases message loss in the overlay, but may lead B to believe that A has failed. If $A \to B$ is being used as an ordinary overlay link, this is not critical; B will simply replace A in its routing table by another node. However, if A is in B's leaf set, B may decide to take over responsibility for A's region of the key space, thus creating an inconsistency.

To prevent this, we allow leaf set members to maintain connectivity by using other nodes as intermediaries. We view all leaf set connections as *virtual links*, rather than physical links. For example, if there is another node C in A's leaf set who can still reach B, A can replace its direct link $A \to B$ with a virtual link $A \to C \to B$. This is a well-known technique that is widely used in mobile ad-hoc networks, where path failures are common. 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.

We use *source routing* to forward packets over virtual links. Messages sent via source routes are not subject to normal 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 follow invariants on overlay routing (e.g. always routing to a node closer to the message destination). Since virtual links act exactly as physical links, these invariants are maintained. Also, source routes are used only within the same leaf set; hence their maximum length is bounded by the leaf set size.

4.5 Route discovery

Every node periodically advertises its best virtual links to all other leaf set members, who use them to derive virtual links for themselves. For example, if A advertises a link $A \to B$ to C, and C's best link to A is currently $C \to D \to A$, then C concludes that B may be reached via $C \to D \to A \to B$. 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 physical 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 fresh link, if one is available. If not, the sender can broadcast a *route request* to all of its leaf set members, who attempt to forward it to the destination. The destination responds with a *route reply*. This mechanism is inspired by the DSR routing protocol [8].

Unlike many other ad-hoc routing protocols, DSR does not require connectivity to be symmetric. This is not only necessary for handling asymmetric path failures, it also allows us to incorporate nodes behind NATs and firewalls into the overlay, whose connectivity is asymmetric as well.

4.6 Liveness checks

Direct neighbors in the key space – and optionally all nodes within the same leaf set – constantly monitor each other's liveness. For this purpose, they make sure that they receive at least one message from each other within a time period T_P. 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.

Nodes characterize each other's perceived liveness in four *stages* $S_1 \dots S_4$. When a node A receives a message from another node B, it changes B's perceived state to S_1. However, if no further messages from B arrive for a time period $T_1 < T_P$, A changes B's perceived state to S_2 and sends a `Ping` message to B over the best known virtual link.

Should B not respond after T_2, A starts sending additional `Ping` messages over all known virtual links to B. If a `Pong` arrives now, A changes B's virtual link to the source route listed in the `Pong` and resets B's state to S_1. If, however, B still does not respond after T_3, A changes its state to S_3 and broadcasts a route request for B to discover a new virtual link.

However, if T_P expires before any route is found, A has established that *none* of the other leaf set members can reach B any more, so under our assumption that leaf sets are always connected, B cannot be alive. Hence, A declares B dead by setting its state to S_4. In this state, A neither advertises the route to B nor accepts any advertisements regarding B from other nodes except B itself. Once T_P expires a second time, A knows that all other leaf set members must have declared B dead as well, and it is free to delete B from its leaf set.

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.

4.7 Gossiping

As described earlier, nodes periodically advertise source routes for all of their leaf set members to their entire leaf set. This gossiping process ensures that leaf sets converge; even if an inconsistency between leaf sets arises, every member eventually learns about all the other members. Since failed nodes have to pass through state S_4, oscillations cannot occur.

Since the liveness of a node is determined individually by each of its leaf set members, there is no need to propagate information about node failures. This not only prevents 'disagreement' between nodes, which is difficult to resolve; it also makes the protocol much easier to secure in an environment with malicious nodes.

4.8 Key Ownership

To prevent overlaps between the responsible region of a newly joined node and those 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 message for keys outside of this range. When the first node N_1 in the network starts up, it automatically has ownership over the range $[k_1, k_1)$. As nodes join, they request range transfers from existing nodes. For example, when the next node N_2 boots up, it will ask N_1 to transfer ownership of the range

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

Note that once a node transfers a 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.

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 4.6). If a neighbor is declared dead, then, by the assumption that leaf sets are connected, we know that no leaf set 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.

4.9 Justification

Earlier, we made the assumption that leaf sets were always connected, i.e. that in a given node N's leaf set $[L_{-l}, L_{-l+1}, \ldots, N, \ldots, L_{l-1}, L_l]$ there always exists a path from N to each L_i, and vice versa. In this section, we provide a quick justification of why this is a reasonable assumption.

As mentioned earlier, we assume that each path $L_i \rightarrow L_j$ in a given leaf set fails independently with probability p. For simplicity, we consider virtual links with at most two hops. Then A cannot reach B if the direct path $A \rightarrow B$ fails *and* for every leaf set member L_i, either the path $A \rightarrow L_i$ or the path $L_i \rightarrow B$ fails. If the leaf set 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 leaf set of A and B, which ranges from $l-1$, when A and B are far apart, to $2(l-1)$, when they are adjacent. We consider A and B 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 A 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 leaf sets ($l = 8$) 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 allow virtual links with more than two hops by increasing the hop limit in route requests, 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.

4.10 Experience

We have implemented the above techniques into FreePastry [6], 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 is available as part of the FreePastry 1.4.1 release [6].

4.11 Partitions

So far, we have shown that routing consistency is maintained by our techniques given the assumption that leaf sets are always connected. In this section, we discuss failure scenarios that break this assumption.

As shown in Section 4.9, it is highly unlikely that a leaf set becomes partitioned from churn or node failures alone. However, a partition in the underlying network can cause a large group of nodes to become unreachable, with the likely result that each group of nodes forms an independent ring. In this case, there is a tradeoff between availability and consistency [7]. At one end of this tradeoff is the primary-partition model [13], in which consistency is maintained by discontinuing services in all partitions except one designated partition. At the other end are the optimistic consistency models used in distributed file systems such as Coda [16], which optimize for availability by allowing updates in all partitions and later resolve any conflicts that might have occurred.

Optimistic consistency is difficult to implement at the overlay level because conflict resolution techniques are highly application-specific. However, there are several ways to provide support for the primary-partition model. For example, nodes can monitor the node density in their local leaf sets. If a node observes a sharp drop (e.g. more than 50% of the leaves disappearing during a single timeout period), it is likely that the node is in a minority partition, and it can respond by resigning from the overlay. After that, the node can periodically attempt to rejoin, using exponential backoff on the retry intervals.

5 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] 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.

[3] 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.

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

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

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

[7] R. Friedman and K. Birman. Trading consistency for availability in distributed systems. Technical Report TR95-1579, Cornell University, Apr 1996.

[8] 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.

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

[10] R. Mahajan, M. Castro, and A. Rowstron. Controlling the cost of reliability in peer-to-peer overlays. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*, Berkeley, CA, Feb 2003.

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

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

[13] A. M. Ricciardi and K. P. Birman. Using process groups to implement failure detection in asynchronous environments. In *Proceedings of the 10th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 341–353, Montreal, Canada, 1991.

[14] 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.

[15] 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.

[16] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere. Coda: A highly available file system for a distributed workstation environment. *IEEE Transactions on Computers*, 4(39):447–459, Apr 1990.

[17] Z. Shu and Y. Kadobayashi. Troubleshooting on intra-domain routing instability. In *Proceedings of the ACM SIGCOMM workshop on network troubleshooting*, pages 289–294, 2004.

[18] 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.

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