Scaling All-Pairs Overlay Routing

David Sontag*, Yang Zhang*, Amar Phanishayee†, David G. Andersen†, David Karger*

*Massachusetts Institute of Technology, †Carnegie Mellon University

ABSTRACT

This paper presents and experimentally evaluates a new algorithm for efficient one-hop link-state routing in full-mesh networks. Prior techniques for this setting scale poorly, as each node incurs quadratic (n^2) communication overhead to broadcast its link state to all other nodes. In contrast, in our algorithm each node exchanges routing state with only a small subset of overlay nodes determined by using a quorum system. Using a two round protocol, each node can find an optimal one-hop path to any other node using only $n^{1.5}$ per-node communication. Our algorithm can also be used to find the optimal shortest path of arbitrary length using only $n^{1.5}\log n$ per-node communication. The algorithm is designed to be resilient to both node and link failures.

We apply this algorithm to a Resilient Overlay Network (RON) system, and evaluate the results using a large-scale, globally distributed set of Internet hosts. The reduced communication overhead from using our improved full-mesh algorithm allows the creation of all-pairs routing overlays that scale to hundreds of nodes, without reducing the system's ability to rapidly find optimal routes.

Categories and Subject Descriptors

C.2.2 [**Communication/Networking and Information Technology**]: Network Protocols—*Routing protocols*; D.4.7 [**Operating Systems**]: Organization and Design—*Distributed Systems*; D.4.5 [**Operating Systems**]: Reliability—*Fault-tolerance*; D.4.8 [**Operating Systems**]: Performance—*Measurements*

General Terms

Algorithms, Performance, Experimentation, Measurement, Theory

Keywords

Networks, Availability, Overlay Networks, RON, Routing, Scalability, Distributed Shortest Path

1. INTRODUCTION

A number of modern systems rely on the ability to find one-hop paths in full mesh networks, ranging from overlay networks (e.g., RON [3]) to peer-to-peer systems that find one-hop "reputation paths" between two nodes to create incentives to share data [17]. Full-mesh link-state routing is challenging at scale, however, because conventional approaches send a full copy of their link-state table to every other node in the system. At the same time, the size of the link-state table grows linearly with the number of nodes. As a result, route computation requires $O(n^2)$ per-node communication, where n is the number of nodes in the overlay. Because of this difficulty, such systems either scale poorly (e.g., Resilient Overlay Networks are commonly perceived to scale only to 50 or so nodes [3]), relax their requirements for optimal paths [7, 9, 11, 16], or centralize communication and route computation at a handful of highly-provisioned nodes [17].

In a typical full-mesh routing system, each node both probes every other node (*measurement*) and sends its link-state routing table to every other node (*route computation*). While there has been significant progress in recent years towards adaptive measurement schemes, distributed routing algorithms remain a significant obstacle to scaling.

In this paper, we seek to lower this barrier by presenting a new algorithm for one-hop link-state routing in full-mesh networks. While prior attempts at scaling overlay routing either reduced the fidelity of routing or removed from consideration particular inter-node links, the algorithm we present here provides the provably optimal one-hop path on the full mesh using only $4\sqrt{n}$ messages of size $O(n)$, for a total of $O(n^{1.5})$ per-node communication. Our algorithm can also be used to find the optimal shortest path of arbitrary length using only $O(n^{1.5}\log n)$ per-node communication. Through this increased scalability, we seek to further the goal of efficient full-mesh routing on hundreds of nodes. Among other applications, such routing could make overlay networks amenable to use in modern peer-to-peer networks, Voice-over-IP platforms such as Skype that route in a peer-to-peer manner, or as control planes for distributed platforms such as PlanetLab.

Our system operates by a simple mechanism: each node measures its links to all other nodes. Next, each node transmits its full link state to a *subset* of other nodes. The routing mechanism ensures that for every pair of nodes in the network, there exists at least one node that has received a full copy of the link state table for both members of the pair. As a result, this "rendezvous" node can tell both nodes what their best path is to the other. The key, of course, is constructing a routing graph that provides this property; we do so using a novel algorithm inspired by quorum systems [4, 14], which we present in more detail in Section 3.

We evaluate the effectiveness of the algorithm by applying it to a Resilient Overlay Network [3], where we substitute the new routing algorithm for the original routing algorithm, which had each node transmit its full link state to all other nodes.[1] We use a simplified

CoNEXT, December 1–4, 2009, Rome, Italy.
Copyright 2009 ACM 978-1-60558-636-6/09/12

[1] Code for our algorithm and its evaluation may be downloaded from: http://projects.csail.mit.edu/overlayrouting/

version of RON where we have eliminated several redundant features in the routing announcements to further increase scalability.

In practical terms, our improved algorithm can increase the capacity of a RON several-fold. For example, a RON with 56Kbps of probing and routing traffic with 30-second failover would be able to support nearly twice as many nodes (from 165 to 300); viewed another way, using our techniques, an overlay running at each of the 416 PlanetLab sites would consume 86Kbps for both incoming and outgoing probing and routing traffic; using prior systems, such a network would exceed the capacity of many residential links, consuming 307Kbps. Importantly, our algorithm allows these overlays to scale with no reduction in fidelity.

The rest of the paper is organized as follows: Section 3 describes the routing algorithm, and Section 4 extends the basic routing algorithm to deal with link and node failures. In Section 5, we describe the design and implementation details of our system. In Section 6 we evaluate our system for scalability, resilience to failures, and routing effectiveness. Finally, in Appendix A we show that under some reasonable assumptions our algorithm is *optimal*, i.e. that no other algorithm can find optimal one-hop routes with less per-node communication.

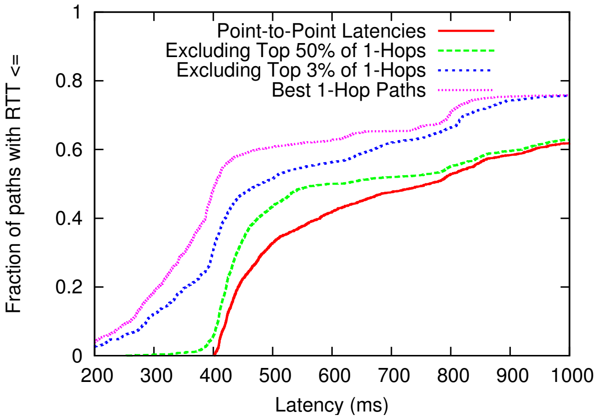


Figure 1: Comparison of RTT for pairs of PlanetLab hosts whose point-to-point latencies were larger than 400 ms (high-latency paths). For the "excluding top n%" graphs, we removed the top n% of one-hop alternatives for each high-latency path from consideration, then used the best remaining one-hop.

2. RELATED WORK

Overlay networks and one-hop routing. The major target of our work is increased scalability of routing overlay networks. Overlays are attractive because deploying them requires only the cooperation of the end hosts involved. Existing systems have shown that such overlays can improve availability by between two and ten times [3, 11, 18], and on paths with unusually high latency can improve delay by hundreds of milliseconds.

Despite these potential benefits, routing overlays are as yet only used in limited contexts, such as inside some distributed storage products and in route optimization products such as Akamai's SureRoute [2]. We hypothesize that one barrier to the wider use of routing overlays are their perceived scalability limits. One of the main goals of our work is to make possible the creation of routing overlays with thousands of nodes.

For example, a Voice-over-IP (VoIP) company like Skype could provision thousands of computers near the edges of the Internet that frequently probe each other and execute our routing algorithm, maintaining a list of optimal one-hop routes between any two locations. If the direct Internet route between two Skype users has unacceptable latency, the users could ask their nearest overlay nodes for the best one-hop route from themselves to their destination. Previous work has shown that point-to-point latencies remain roughly constant over short intervals of time [20]. Thus, we can imagine that measurement and route computation in this overlay could be done every 5 minutes, with more frequent updates done when changes are discovered through passive measurement.

Scaling by path pruning. An alternate approach to reducing the cost of routing and probing in routing overlays is to reduce the number of paths that must be considered. While this could require more hops and may miss some useful paths, in practice it seems to be effective. This approach is enticingly complementary to the one we present in this paper; the two might be combined to create overlays that scale to even larger numbers of nodes.

Nakao et al.'s routing "Underlay" provides overlay programs with topological information from sources such as BGP [15]. Using information derived from the physical topology can assist in pruning the paths that the overlay considers. The authors found that this reduces routing overhead by a factor of two [16]. A similar approach is taken by Fei et al., who use BGP routing information to efficiently select AS-disjoint paths between nodes [9].

Cui, Stoica, and Katz propose using a model of correlated link failures to select failure-disjoint backup paths in overlay networks [7]. If these correlations are known a priori, this technique can efficiently select a diverse set of paths. Applying it in practice requires mechanisms for estimating joint failure probabilities using either topological knowledge or periodic active measurements.

Random intermediate selection is used in SOSR [11] to find a working path to a destination server. This work shows that picking from four randomly chosen intermediaries is enough to find paths that optimize one specific metric: availability. Our goal in this work is to scalably find optimal one-hop (or multi-hop) routes for arbitrary metrics in a full mesh network. A natural question to ask, therefore, is: Can random intermediate selection (perhaps with many intermediates) discover these routes?

To explore this question, we perform a measurement study on PlanetLab of the total path latency for direct and single-hop indirect paths. Figure 1 shows the average latency (over 10 pings in a 15 minute interval) between node pairs on PlanetLab on November 23, 2005 (data from [19]). This graph excludes paths for which all pings were lost. The figure shows the improvement in latency given by the best one-hop paths for the 2656 direct Internet paths whose point-to-point latencies were larger than 400 ms, between 359 PlanetLab hosts. In some cases, the improvement is substantial. For this overlay, longer routes (e.g. two- or three-hops) would not result in better overall latencies.

How many random nodes would a scalable overlay need to contact to find a low latency route? Figure 1 shows that for most host pairs, the bottom 50% of potential 1-hops (sorted by total latency) does not contain *a single* low latency route—and even considering the bottom 97% of one-hop intermediates still miss many of the latency improvements (only 30% of the paths would have lower than a 400 ms latency, compared to at least 45% of the paths when using the best 1-hop). In other words, 97% of the time, a randomly chosen intermediary will not significantly improve latency. From this data, we conclude that, while random intermediary selection is effective for *availability*, it works poorly for other metrics such as latency where the best path must be selected more carefully.

Similar observations have also been made by Lumezanu et al. [13], who suggest using the errors made by network coordinates to locate potential low-latency detours in overlay networks.

Quorum systems are used in distributed systems to allow data replication while ensuring consistency among nodes. The grid quorum [4] provides a deterministic construction in which every node communicates with the same number of replicas. Quorum systems have been recently used for search and routing in peer-to-peer networks [1, 10]. These works are concerned with object look-up and routing in sparse overlays, and measure latency with respect to the latency of the direct *Internet* route between two nodes. Another similar case is the use of quorum systems to track node location in wireless networks [8, 12]. Our work differs from these in that, for us, the quorum is used to compute the actual route to the destination. To our knowledge, our work is the first to apply quorum systems to the problem of route computation. Also, we show in Appendix A that the construction provided by the quorum system is optimal for this problem.

3. ROUTING ALGORITHM

The goal of our routing algorithm is to find the optimal one-hop route for all pairs of nodes in the network with as little per-node communication as possible. The standard technique of broadcasting each nodes' link state to all other nodes, which consumes n^2 communication bandwidth *per node*, actually provides more information than necessary: Not only can a node calculate the best one-hop path from itself to each destination, it can also calculate the optimal path from any other node to all nodes.

One possible low-communication approach would be for each node to transmit its link state to one central *rendezvous* node, which then computes all of the optimal one-hop paths and returns routing tables to each node. This would reduce the *total* communication to n^2. However, because all communication must go through a single rendezvous, it does not reduce the communication work for this one node. Such a centralized solution is a bottleneck that impairs scalability and makes the overlay vulnerable to an unlucky network failure, or a malicious attack, breaking all routing.

Our strategy is to *distribute* this rendezvous work, so that every node acts as a rendezvous for a *small* number of other nodes. After receiving the link state of some of the nodes, each node has a partial view of the network. It can then use this partial view to compute best one-hop routes using the links it received, returning to the senders potentially good one-hop routes to other nodes. Key to this algorithm is showing that this strategy, when applied carefully, is guaranteed to find the optimal one-hop routes for all nodes.

We assume for this discussion that all links are bidirectional with identical cost.[2] Each node i is assigned a set of *rendezvous servers* R_i. These sets must be constructed such that every pair of nodes shares at least one rendezvous server; that is, for every pair of nodes i and j, $R_i \cap R_j$ is non-empty. One way to construct such sets is to use a *grid quorum*, shown in Figure 2 for $n = 9$ nodes. A grid of size $\sqrt{n} \times \sqrt{n}$ is filled in with the numbers $1, \ldots, n$ in any order. If node i is placed in position (x_i, y_i), R_i consists of all the nodes in row x_i and column y_i. If node j is at (x_j, y_j), then $R_i \cap R_j$ contains the nodes in positions (x_i, y_j) and (x_j, y_i). This construction provides two important properties: first, every node pair shares a rendezvous server, because every column and row intersect.[3] Second, the rendezvous load is evenly distributed among the nodes in the network. Nodes for which node i acts as a rendezvous server are called its *rendezvous clients*, and are denoted by C_i. Note that in the grid quorum construction, R_i is the same as C_i. This symmetry is unnecessary, however, and the routing algorithm could be applied with other quorum constructions that do not have it.

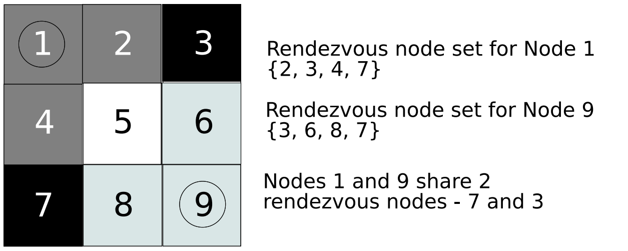
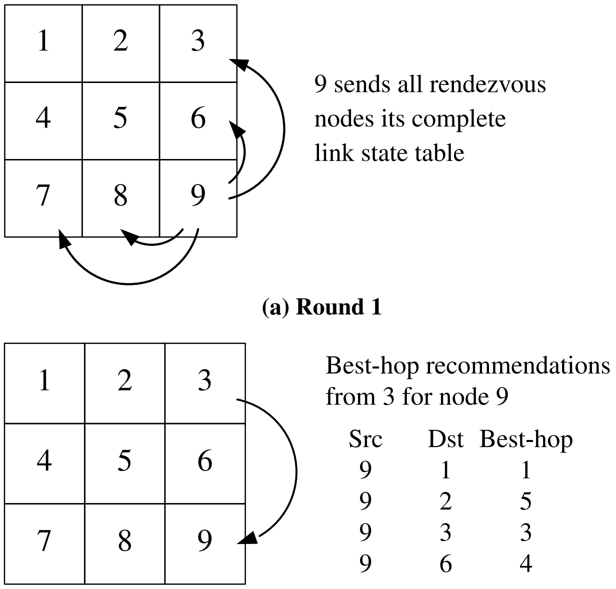

1
2
3
4
5
6
7
8
9
Rendezvous node set for Node 1
{2, 3, 4, 7}
Rendezvous node set for Node 9
{3, 6, 8, 7}
Nodes 1 and 9 share 2
rendezvous nodes - 7 and 3

Figure 2: Grid quorum for $n = 9$ nodes. Every pair of nodes in the grid has at least two rendezvous servers in common.



(a) Round 1

Src	Dst	Best-hop
9	1	1
9	2	5
9	3	3
9	6	4

(b) Round 2

Figure 3: The two rounds of the routing algorithm as observed by node 9. Figure (a) illustrates round 1, where node 9 sends its rendezvous servers its complete link state table. In round 2, node 9 receives routing recommendations from each of its rendezvous servers. Figure (b) shows the one-hop recommendations received from node 3.

Our algorithm operates in two rounds. In the first round, node n sends its link state table to all of its rendezvous servers R_n. In the second round, node n (now a rendezvous server) computes, for every pair of its rendezvous clients $i, j \in C_n$, the best one-hop route $\langle i, h, j \rangle$, where h ranges over all nodes in the overlay. This computation can be performed by n because it knows the full link state table for both i and j, so it can compute their best intersection.

[2] In the case of asymmetric link costs, the link state transmitted in round one would include both costs.

[3] Every node pair actually intersects *twice*. We will later describe how this provides additional redundancy

Finally, node n sends one-hop recommendations to its rendezvous clients C_n, simultaneously receiving one-hop recommendations from its rendezvous servers R_n.

Figure 3 shows an example of this two round table exchange. In the first round, node 9 sends its link state to its rendezvous servers (3, 6, 8 and 7). In the second round, node 9 receives from each of its rendezvous servers, r, the best path to r's rendezvous clients. Figure 3(b) shows one such exchange in round two, from node 3 to node 9. Note that node 3 would simultaneously send recommendations to its other rendezvous clients.

We now show that our construction of the rendezvous server sets using the grid quorum results in every node knowing the best one-hop path to every other node in the overlay:

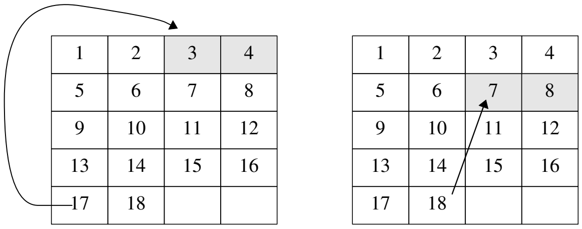
Theorem 1. *This algorithm finds all optimal one-hop routes, with each node sending and receiving at most* $4\sqrt{n}$ *total messages and* $\theta(n\sqrt{n})$ *bits.*

Proof. The communication follows from the two-round protocol and our construction of R_i. In the first round, a node sends its link-state to its $2(\sqrt{n}-1)$ rendezvous servers. In the second round, a node sends routing recommendations to its $2(\sqrt{n}-1)$ rendezvous clients. Let k be some node in $R_i \cap R_j$, which by construction is non-empty. Since k receives both i and j's link state, it can compute the optimal one-hop path from i to j. Node k will send this one-hop recommendation to i and j at the end of the second round. □

Furthermore, this scheme performs as well as *any* algorithm that operates by comparing all of the possible one-hop paths.[4] The proof in Appendix A shows that the minimum per-node communication required to solve the optimal one-hop problem is equal to the per-node communication required by our scheme.

Non perfect-square grids. The number of nodes in the overlay may not be a perfect square, resulting in empty spaces in the last two rows. If the entire last row would be empty, we instead form a grid of size $\sqrt{n} \times (\sqrt{n}-1)$.[5] However, the last row may still be incomplete, containing only k nodes. If this is the case, some node pairs may not have two rendezvous servers in common, because there will be a "blank space" in the grid. Note that the naive solution to this problem, duplicating nodes 1 and 2 into the blank spaces, would double the amount of routing traffic for those few nodes.

To cope with this case, we instead build on the observation that all of the nodes in columns $k+1$ and beyond have one fewer rendezvous client than the nodes in the first k columns, because the last entries in their column are blank. As a result, we can assign these nodes as additional rendezvous servers for other nodes. If the last column of the grid is column c (where $c = \sqrt{n}$ or $\sqrt{n}-1$), we give the node at $(\sqrt{n},1)$ the nodes at $(1,k+1)$ through $(1,c)$ as additional rendezvous servers, the node $(\sqrt{n},2)$ the remaining nodes in row 2, and so on:



More precisely, we assign a node at position $(\sqrt{n},i)$ all nodes (i,j) for $k+1 \leq j \leq c$. These rendezvous assignments are symmetric: we also assign the upper right nodes to one node from the bottom row.

With this construction, every node once again has a rendezvous server in every row and in every column. Clearly, every pair of nodes in the first $\sqrt{n}-1$ rows and every pair of nodes in the first k columns have at least two rendezvous servers in common. A node in the last row, grid position $(\sqrt{n},i)$ and any other node in grid position (a,b) for $b > k$ will have the rendezvous servers in positions (i,b) and (a,i) in common. It is also easily seen that all nodes have at most $2\sqrt{n}$ rendezvous clients (and servers).

Multi-hop routes. Our algorithm may also be applied to find optimal multiple-hop routes of length l. As applied to overlays, while research has suggested that one-hop paths are sufficient for latency reduction and reliability, certain ISP policy constraints may force nodes to take two-hop paths in order to route around failures. For example, an overlay route from a commercial node might first hop to an Internet2-connected node, traverse the Internet2, then emerge to go to a second commercial destination. Such a route could enable these commercial destinations to circumvent a full Internet partition, though its use may contravene network policy.

We find the optimal routes of length $\leq l$ by repeating the algorithm $\log l$ times. At iteration t, in round one, each node announces a modified link state where, for each destination, it gives the cost of the best path of length $\leq 2^{t-1}$ to that destination (found in the previous iteration). For $t = 1$ this corresponds to the usual link state table. The best one hop from i to j chooses among all paths $i \sim k \sim j$. However, since the modified cost for $i \sim k$ is that of the shortest path of length $\leq 2^{t-1}$ from i to k (and analogously for $k \sim j$), this now gives us the cost of the shortest path of length $\leq 2^t$.

Although this will succeed in finding the *cost* of the optimal route, discovering the actual paths would require additional communication. Luckily, for routing purposes, all we need to know is what node to forward a packet to. To recover this, we modify the link states sent in the first round to also include the identity of the second node along the best path found so far to each destination.

Let Sec^t_{ij} denote the identity of the second node along the best path of length $\leq 2^{t-1}$ from i to j. For example, for $t = 1$ we have that $Sec^1_{ij} = j$. For $t = 2$, Sec^2_{ij} is either j (if the direct route is best) or the optimal one-hop from i to j. Suppose in iteration t we find that node k is the best one-hop for the path from i to j. Then, in the second round, the routing recommendation returned to i for j will be the node Sec^t_{ik} and the total cost of the path $i \sim k \sim j$. In the first round of the next iteration we will have that $Sec^{t+1}_{ij} = Sec^t_{ik}$.

Notably, this technique can provide all-pairs shortest paths using only $\theta(n\sqrt{n}\log n)$ per-node communication, which is an asymptotic improvement on the earlier best known algorithm, $\theta(n^2)$. As one consequence, with just twice the communication this algorithm can find optimal 3-hop routes, which we suspect is long enough for most applications of overlay routing.

4. LINK AND NODE FAILURES

Our routing algorithm is vulnerable to additional failure modes because it relies on rendezvous nodes that are almost always not on the optimal path in order to find the optimal path. In this section, we describe mechanisms that ensure that our system continues to perform well in the face of these failures.

The system provides resilience to failures via three mechanisms. First, it uses redundant rendezvous nodes, so that no single failure

[4] A topology-aware technique, such as those we discuss in Section 2, might perform better because it need not examine all paths.

[5] More formally, let $a = \sqrt{n} - \lfloor\sqrt{n}\rfloor$. If $a < .5$, we form a grid of size $\lceil\sqrt{n}\rceil \times \lfloor\sqrt{n}\rfloor$, otherwise, we form a grid of size $\lceil\sqrt{n}\rceil \times \lceil\sqrt{n}\rceil$.

will impair route computation at all. Second, it can quickly recruit additional rendezvous nodes via a failover mechanism. Third, it can take advantage of redundant reachability information that it receives from the nodes in its rendezvous set.

Failure modes: To precisely explain the failover mechanisms, we first describe the failure modes that we consider. Two nodes i and j experience a *rendezvous failure* if either node cannot communicate with a rendezvous server k that should connect them. Node i observes a *proximal rendezvous failure* if it cannot reach node k, and a *remote rendezvous failure* if node k and node j are unable to communicate.

Recall that rendezvous node sets consist of all nodes in node i's row and column.[6] The rendezvous node sets of any two nodes intersect in at least two locations (i's row and j's column, and j's row and i's column). As a result, two nodes are only unable to find the optimal route to each other if they experience a *double rendezvous failure*.

4.1 Rapid rendezvous failover

In the event that all rendezvous nodes used for a destination node fail, the node observing the failure will quickly select new "failover" rendezvous servers. These failures could have been either proximal or remote failures. When node i detects a rendezvous failure to node j, it selects a node from among the (reachable) nodes in j's row and column to serve as a failover rendezvous f.

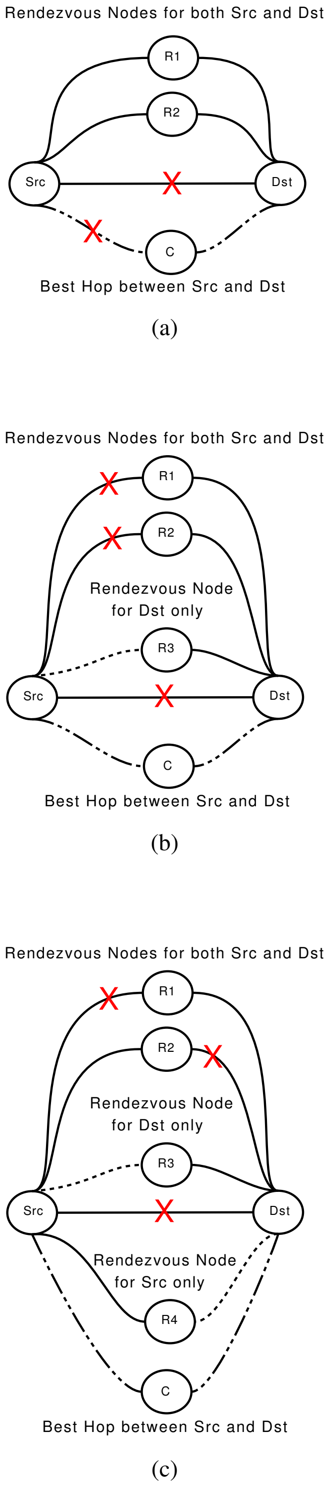
The failover rendezvous f is selected uniformly at random from this set of $2\sqrt{n}$ candidates so that, in the event of concurrent failovers, the failover load is evenly distributed. Once f is chosen, i includes f in its rendezvous server set and subsequently sends its link state to f, who will then consider i to be a new rendezvous client. f responds with the best one-hop routing recommendations between i and all other nodes in f's row and column. i determines whether f can reach j by looking at the one-hop routing recommendations from f. If j cannot be reached via f, i retries with another candidate. The probability that all $2\sqrt{n}$ candidates are failed with respect to j is low under any constant probability of failure, so i can always expect to find a failover rendezvous from this row-column set.

If a failover rendezvous fails, then the failover process restarts. i continually monitors its link state and reverts to its original rendezvous nodes when they become available again. j employs the same failover mechanism for its rendezvous nodes to i.

Nodes can detect proximal failures directly since they are already monitoring their links to every other node. They detect remote failures between a rendezvous k and a remote node j by observing that k stopped recommending any route to node j, which it will only do if k stops receiving link-state updates from j. In the worst case, remote failure detection could take one routing interval to detect and initiate recovery from. To understand better the failover times using our algorithm, we explore three failure scenarios in which node *Src* is seeking the best hop node C to a destination *Dst*, using rendezvous servers *Ri*:

Scenario 1: Direct and best hop failure: $\leq 2r$ seconds. In Figure 4(a), the links $Src - Dst$ and $Src - C$ fail. As shown in Figure 5, *Src* notifies its rendezvous servers $R1$ and $R2$ of the failures after one routing period r, so at most $2r$ seconds after detecting the failures, it will receive the new best hop recommendation for *Src* to *Dst*.

Scenario 2: Proximal rendezvous + direct failures: $\leq 2r$ seconds. In Figure 4(b), *Src* has proximal rendezvous failures to *both*


Rendezvous Nodes for both Src and Dst
R1
R2
Src
Dst
C
Best Hop between Src and Dst

(a)

Rendezvous Nodes for both Src and Dst
R1
R2
Rendezvous Node
for Dst only
R3
Src
Dst
C
Best Hop between Src and Dst

(b)

Rendezvous Nodes for both Src and Dst
R1
R2
Rendezvous Node
for Dst only
R3
Src
Dst
Rendezvous Node
for Src only
R4
C
Best Hop between Src and Dst

(c)

Figure 4: Three possible failure scenarios (described in Section 4.1). R1 and R2 are rendezvous servers for both Src and Dst. R3 is initially a rendezvous server only for Dst, but is a failover option for Src. Likewise, R4 is initially a rendezvous server only for Src, but is a failover option for Dst. C denotes the best one-hop route between Src and Dst.

[6] For clarity, we will assume for the rest of the paper that the number of nodes is a perfect square.

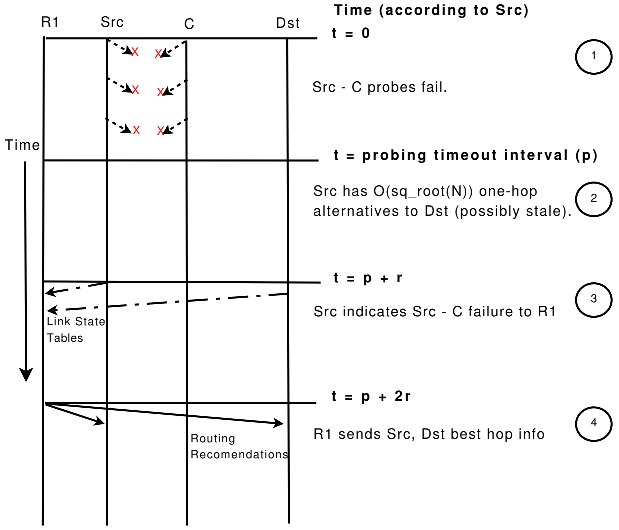

R1
Src
C
Dst
Time (according to Src)
t = 0
1
Src - C probes fail.
Time
t = probing timeout interval (p)
Src has O(sq_root(N)) one-hop
alternatives to Dst (possibly stale).
2
t = p + r
Src indicates Src - C failure to R1
3
Link State
Tables
t = p + 2r
R1 sends Src, Dst best hop info
4
Routing
Recomendations

Figure 5: Failure recovery for scenario shown in Figure 4(a).

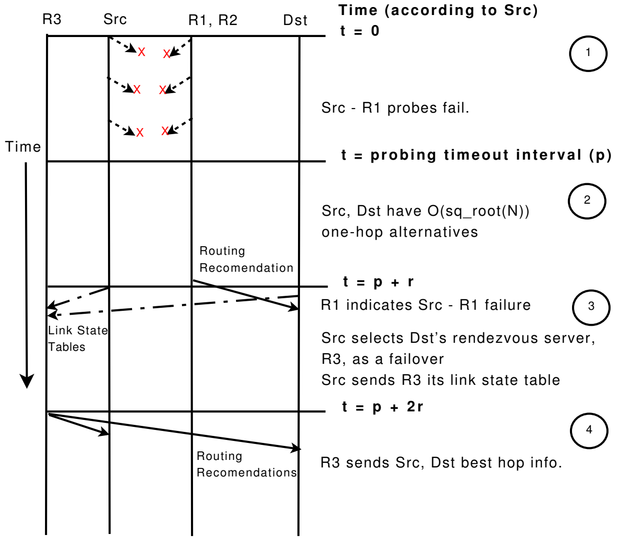

R3
Src
R1, R2
Dst
Time (according to Src)
t = 0
1
Src - R1 probes fail.
Time
t = probing timeout interval (p)
Src, Dst have O(sq_root(N))
one-hop alternatives
2
Routing
Recomendation
t = p + r
R1 indicates Src - R1 failure
3
Link State
Tables
Src selects Dst's rendezvous server,
R3, as a failover
Src sends R3 its link state table
t = p + 2r
4
Routing
Recomendations
R3 sends Src, Dst best hop info.

Figure 6: Failure recovery for scenario shown in Figure 4(b).

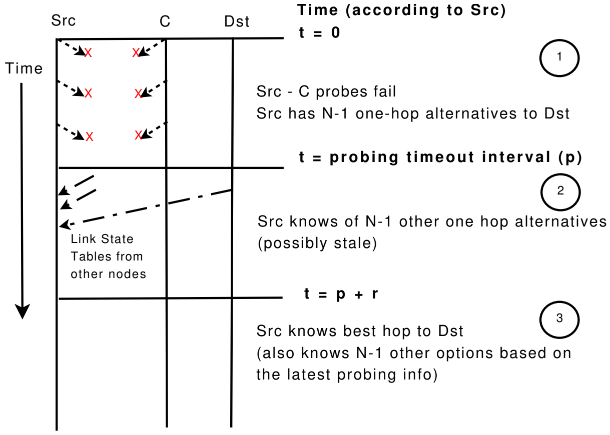

Src
C
Dst
Time (according to Src)
t = 0
Time
1
Src - C probes fail
Src has N-1 one-hop alternatives to Dst
t = probing timeout interval (p)
2
Src knows of N-1 other one hop alternatives
(possibly stale)
Link State
Tables from
other nodes
t = p + r
3
Src knows best hop to Dst
(also knows N-1 other options based on
the latest probing info)

Figure 7: Shown here, for comparison, is the typical failure recovery used in full-mesh link state routing (e.g., RON).

of its rendezvous nodes $R1$ and $R2$ *and* experiences a link failure to *Dst*. Figure 6 shows the timing diagram explaining the phases of the routing protocol as *Src* performs a rendezvous failover, and recovers from the failures to regain the best hop information to *Dst*. Once the probes between *Src* and $R1$ (also $R2$) fail and a link failure is detected in time interval 1, *Src* immediately selects another of *Dst*'s existing rendezvous nodes, $R3$. *Src* sends $R3$ its link state information in or before time interval 3, and receives the best hop to *Dst* from $R3$ in time interval 4. $R3$ can make this decision for *Src* because it is already receiving the link state tables from *Dst*. Thus it takes *Src* at most $2r$ seconds to find the best hop after detecting the failures. This timing diagram also illustrates, by swapping *Src* with *Dst*, the setting of two remote (rather than proximal) rendezvous failures. Since $R3$ sends both *Src* and *Dst* the best hop info, both nodes discover the new path simultaneously.

Scenario 3: Proximal and remote rendezvous + direct failures: $\leq 3r$ seconds. In Figure 4(c), *Src* experiences a proximal rendezvous failure to $R1$, a remote rendezvous failure to $R2$, and a direct link failure to *Dst*. In this case *Src* must wait up to r additional seconds to detect the remote rendezvous failure and select a failover rendezvous server $R3$. In total, *Src* will find the best hop to *Dst* at most $3r$ seconds after detecting the initial failure.

Comparison to n^2 link-state failover. Finally, Figure 7 shows the recovery timing for ordinary full-mesh link-state routing. Such a system (e.g., a RON overlay) recovers within one probing + one routing interval. In most failures cases (the major exception being scenario 3), quorum routing recovers within one probing + two routing intervals. To compensate for this difference, in our evaluation we set the routing interval for the quorum system to half that of the normal link-state algorithm.[7] As we show in the next two sections, for the majority of failures, an overlay using our algorithm recovers as quickly or faster than the original overlay would, while still consuming far less bandwidth.

The total recovery time could be much larger if there are additional link failures (e.g., if *Dst* cannot communicate with the node in its row/column that *Src* chooses as a failover rendezvous node, and vice-versa). It is possible to give a slightly different failover algorithm that can correct this problem at the expense of an additional routing interval.[8] However, in practice, double (and worse) proximal failures are rare enough that this additional time does not appear to increase the system's time to find a working path (Section 6).

If node *Dst* has failed, all nodes in the overlay will start failing over to nodes in *Dst*'s row or column, until they each have exhausted the possible rendezvous failover nodes for *Dst*. To avoid this, after the initial failover, each node ensures that node *Dst* is alive before failing over to a new rendezvous server. For example, *Src* will check if any of its rendezvous clients' link-state tables show that *Dst* is reachable. If no client can reach *Dst*, then *Src* assumes that *Dst* has failed and does not attempt further rendezvous node failover for *Dst*.

4.2 Redundant link-state information

The third source of robustness is that each node knows the full link-state tables of its $2\sqrt{n}$ neighbors. As a result, a node whose

[7] Since the second round messages are roughly $\sqrt{n}$ times smaller than the first round messages, we could reduce the recovery time further by sending more frequent recommendation messages.

[8] After the first round, *Src* knows the link state of rendezvous clients for which it did not have a link failure. It would then use these as temporary 1-hops to send link state to and receive recommendations from rendezvous servers for which it had a failed link.

rendezvous servers have both failed can still *directly* evaluate the costs of one-hop routes from itself through any of the other $2\sqrt{n}-2$ neighbors to this destination. While there are fewer indirect paths in this set than in the $n-1$ potential indirect paths examined by the rendezvous nodes, prior work on Internet overlays [11] suggests that being able to pick from as few as four intermediaries can significantly improve availability.

5. DESIGN: IMPROVED RON ROUTING

To evaluate the effectiveness of our new approach for full-mesh routing, we implemented a stand-alone version of the probing and routing mechanisms from RON [3], using both the original link-state algorithm and our quorum routing algorithm. Our goal with this implementation is to understand whether the algorithm is practically implementable in a real overlay system, what implementation details it necessitates, and to understand whether its use reduces the effectiveness of the overlay in the face of real Internet communication failures, compared to conventional link-state routing.

Our simplified implementation allows nodes to join and leave the overlay, provides link-state monitoring of latency and loss rates, and measures the quality of paths between the nodes as in the original RON system For simplicity, it omits the application interface, raw packet capture, and policy mechanisms present in the original RON system.

We first describe the three major components in our design:

- **The membership service** ensures that nodes in the overlay know the other participating nodes and have a consistent membership view.
- **Link monitoring** monitors the latency, loss, and availability of the virtual links between nodes, allowing the overlay to react to failures and find a new path.
- **The router** implements the two-round quorum routing algorithm that allows nodes to find optimal one-hop paths.

We conclude this section by examining the pertinent features of our Java-based implementation, which we subsequently evaluate both in emulation and in a PlanetLab-based deployment.

Membership Service: The membership service (MS) maintains a record of the participating nodes in the overlay. The correctness and efficiency of our routing computations depends on the quorum computation: if each node has a consistent view of the membership state, then it can independently construct the grid for quorum computation. We follow the original RON system design of having a long lifetime for membership timeouts (30 minutes), with transient failures handled instead by the overlay failover mechanisms.

Because the focus of this paper is to evaluate the effectiveness of the overlay routing, we use a simple centralized membership service, running on a coordinator node, instead of implementing a more robust distributed membership protocol or consensus protocol (e.g., [6]). This simplification requires that all nodes communicate with the MS node when a new node joins or an existing node leaves the system, but imposes no such constraints during the steady-state operation of the overlay.

On receiving a membership update, nodes restructure their internal grid representation, traversing the grid in row-major order and populating the grid from a sorted list of member IDs. Thus all nodes with the same membership view have consistent grid structures.

Link Monitoring: Link monitoring occurs as in RON. Nodes ping each other to monitor latency and liveness and mark nodes as failed after 5 consecutive failed probes. We implement RON's rapid failure detection technique of temporarily increasing the probing rate after a first probe loss. As a result, our implementation detects failures within 1 probing period. Each node records in its link-state table L an exponentially weighted moving average of the latency to every other node.

Table Exchange: Nodes exchange link-state routing tables much as they do in RON, but using a considerably more compact representation.[9]

Each node maintains a partial $n \times n$ link state table of the estimated latency and liveness $L_{i,j}$ between nodes i and j. It measures its own neighbors as described above, and the other rows in the matrix are updated based on the link states received during table exchanges.

Nodes perform a table exchange with their neighbors. When there are no failures, this corresponds to all nodes in the current node's row and column. (In the naive routing scheme, *all* nodes are neighbors.) The link-state tables are compactly exchanged using two bytes for latency (in milliseconds) and one byte for liveness and loss. With n nodes in the overlay, this array requires at most $3 \cdot n$ bytes (for the outgoing message).

After receiving the link states of its rendezvous clients, node i replies to each of its clients j with best hop recommendations. This message contains recommendations for the best one-hop paths from j to each of i's $(2 \cdot \sqrt{n})$ other clients. Each node ID is a 2-byte integer and we must give *both* the destination node ID and the one-hop node ID, so a recommendation message is $4 \cdot (2 \cdot \sqrt{n})$ bytes in size.

5.1 Implementation

To evaluate the performance of the routing algorithm in both emulation and on the Internet, we built a prototype implementation of our routing algorithm as a stand-alone Java application. The 4000-line implementation uses Java's standard NIO facilities in a simple, event-driven architecture. To compare against the original routing algorithm in RON, our implementation can operate in a mode that uses RON's default all-pairs link-state routing algorithm. We refer the reader to [3] for details on the operation of RON.

For both the simulation and experimental deployments, we configured the parameters to include a probe period of 30 seconds, with a 5-probe timeout. We use a 30 second routing interval for RON and a 15 second routing interval for our new routing algorithm. We use half the routing interval because, in the absence of rendezvous failures, our algorithm takes two routing intervals to find optimal one-hop routes using current probing data.

Configuration parameter	**Full-mesh (RON)**	**Quorum System**
routing interval (**r**)	30s	15s
probing interval (**p**)	30s	30s
#probes for failure	5	5

Note that while deployments may choose different timescales for routing and probing to react more quickly to failures or to reduce overhead, the bandwidth required scales linearly with probing and update frequency. As a result, the relative cost of full link-state exchange and our algorithm remains the same regardless of the actual frequency used.

[9] While this is, in many ways, "merely" an implementation detail, the verbosity of the original system's link-state representation resulted in routing messages being about twice as large as necessary. This difference becomes more important when trying to use our improved algorithm to scale such a system to hundreds of nodes.

6. EVALUATION

The goal of our evaluation is twofold. First, we evaluate the actual bandwidth consumed by the quorum routing algorithm, both under normal operation and in the presence of failures, to confirm that the system scales as expected. Second, we evaluate the availability and latency optimization achieved by the algorithm, in comparison to the normal Internet paths and to a conventional full-mesh overlay routing system, to show that the system meets or exceeds the performance of earlier systems despite its greatly reduced bandwidth requirements.

We perform this evaluation using a real-world deployment on PlanetLab, performed on March 29, 2008, and using an in-system emulation. Our deployment consisted of 140 geographically distributed PlanetLab nodes, chosen from among PlanetLab's least-loaded nodes. We allowed the system to run for 136 minutes, with every node in the overlay performing probing and routing. While this overlay size is smaller than our goal of running the algorithm on all PlanetLab nodes, the remainder were too unstable for measurement. The 140 nodes we did measure still provides ample room for optimization—saving a factor of $\frac{1}{4}\sqrt{140}$ could still reduce bandwidth to one third of what the naive approach would require.[10] Moreover, the savings would continue to grow with more nodes.

In particular, we expect that one of the most effective applications of our routing algorithm will be to the Skype scenario that we outlined in Section 2. In that scenario, because we would be more interested in optimizing latency *on average* rather than recovering from Internet failures, we could afford to do much less frequent measurement and route computations. On an overlay with 10,000 nodes our algorithm, modified appropriately, would give a 50-fold reduction in per-node communication.

Network environment characterization We first seek to understand whether and how the PlanetLab environment will stress our algorithm—in particular, whether it has sufficient concurrent failures to stress the rapid rendezvous failover mechanism and whether it affords sufficient opportunities for availability optimization. We believe this to be the case.

Figure 8 shows the distribution of the number of concurrent link failures per node. For each source, we average, over all measurement intervals, the number of destinations that were unreachable via the direct Internet path for five consecutive probes. Almost all nodes had, on average, fewer than 40 concurrent link failures. Most nodes had relatively good connectivity, and a few nodes had very bad connectivity. Since the nodes with bad connectivity are also rendezvous servers, the nodes in their rows and columns may have to find failover rendezvous servers. The next section shows that these failovers did not substantially increase our algorithm's bandwidth consumption. In Section 6.2, we show that despite these connectivity problems, our algorithm is nearly always able to quickly find the optimal one-hop routes in the overlay.

6.1 Overhead: bandwidth comparison

Scaling in the absence of failures. We first observe the average per-node bandwidth required to operate in steady state as the number of nodes increases. We perform this measurement using emulation. To maximize the fidelity of our emulation results, the emulation uses the same implementation as the one deployed on the Internet. The emulated nodes run on one physical machine. Each run lasts

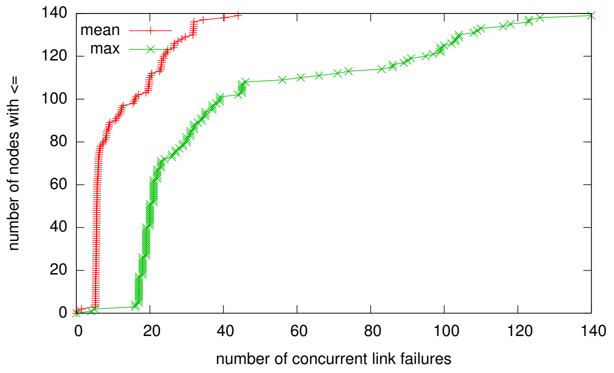


Figure 8: CDF of the number of concurrent link failures per node for an overlay of 140 nodes on PlanetLab. Each data point is an average (or max), of the number of destinations that cannot be reached per source.

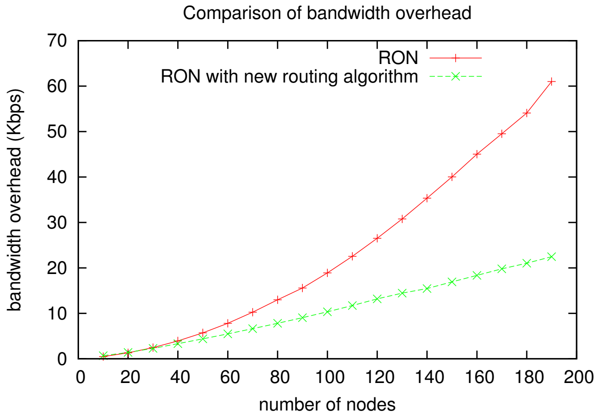


Figure 9: Comparison of average per-node routing traffic (incoming and outgoing), for 5 minutes of running an emulation with no node or link failures.

[10] The factor of $\frac{1}{4}$ arises from (a) using a routing interval that is half as large, and (b) sending messages to $2\sqrt{n}$ nodes in each round.

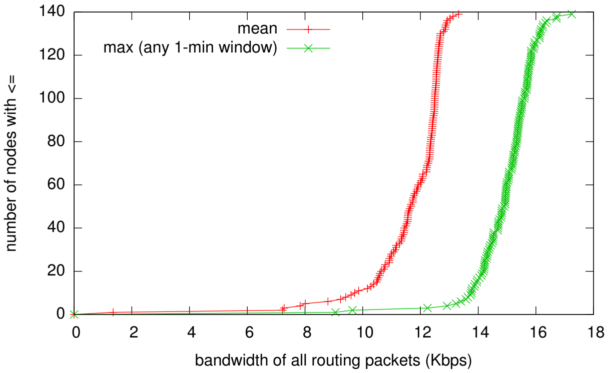


Figure 10: CDF of the per-node routing traffic (incoming and outgoing) on the PlanetLab deployment.

five minutes; we determine the bandwidth as the average bandwidth used after all nodes have joined the network. Figure 9 shows the scalability improvement of our routing algorithm over the naive routing algorithm in emulation, without any node or link failures.

With the reported routing and probing intervals, the theoretical bandwidth of total probing traffic (incoming and outgoing) for RON, with or without our new routing algorithm, is $49.1n$ bps. The theoretical bandwidth of total routing traffic for RON (incoming and outgoing) is

$$1.6n^2 + 24.5n \text{ bps},$$

and for our new routing algorithm is

$$6.4n\sqrt{n} + 17.1n + 196.3\sqrt{n} \text{ bps}.$$

The theoretical scaling numbers match closely with the in-system emulation results. For example, the routing traffic (incoming and outgoing) for 140 nodes would be 34.8 Kbps for the link-state algorithm, and 15.3 Kbps using ours.

Scaling in the real world. We next examined the bandwidth used in the deployment on 140 PlanetLab nodes. We examine both the average and maximum bandwidth used by any node, since the maximum bandwidth would represent a critical barrier to scalability if it ever exceeded a node's capacity. Figure 10 shows the CDF of the amount of bandwidth required by each node in the system. The maximum bandwidth is calculated using 1-minute intervals over the 136 minute deployment.

The average bandwidth required by our routing algorithm on PlanetLab is slightly *less* than that that consumed in emulation and in theory. In particular, for 140 nodes on PlanetLab the routing overhead is 13.5Kbps while the theoretical routing bandwidth is 15.3 Kbps. This occurs because some of the routing packets are lost, but do not need to be re-transmitted because of the system's redundancy. While a few nodes used more bandwidth at some point during the run, the maximum increase was under 30%. The extra bandwidth was consumed primarily by nodes finding additional failover rendezvous servers during severe network failures. Despite some quite serious failures experienced on PlanetLab during this time, no node in the overlay used more than 17 Kbps during any 1-minute interval, and the average remained under 13 Kbps.

Employing our algorithm for overlay routing reduced substantially the bandwidth required for routing. The failover mechanisms and quorum construction properly spread the load of being a rendezvous server across the nodes in the network, ensuring that no node ever had to handle greatly in excess of its expected load.

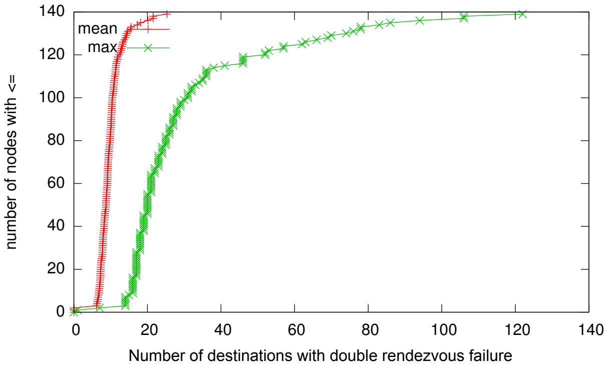


Figure 11: The number of destinations (x-axis) for which a node (y-axis) experiences failures to both of the destination's default rendezvous nodes, calculated at 1-minute intervals.

6.2 Effectiveness

While our routing algorithm will eventually find the optimal one-hop route for all destinations, communication failures between nodes and their rendezvous servers could result in added delay while these nodes attempt to find failover rendezvous servers (see Section 4). In this section, we investigate how these issues affect the overall reliability and usefulness of the overlay, finding that the impact is minimal: even a node with *very* poor connectivity receives routing recommendations for every other node in under 25 seconds on average.

6.2.1 *Rendezvous node failures*

The system will find the optimal route to a destination node within one probing and routing interval *unless* it experiences failures to both of the default rendezvous servers for that destination. A concurrent double failure requires the server to find a new rendezvous node for the destination using the mechanisms described in Section 4. Figure 11 shows that the median node in the deployment experiences almost no double failures, and that 98% of the nodes have fewer than 10 concurrent double failures on average.

We conclude that the redundancy provided by having two default rendezvous servers for each destination suffices to find the optimal route to the vast majority of the destinations in each routing interval, even in the presence of node and link failures. In the next section we investigate the exceptions to this, looking at the effectiveness of the failover mechanisms.

6.2.2 *Optimal routes and update freshness*

We next measured, at 30 second intervals, the amount of time since a node received the last recommendation to each destination in the overlay. (In other words, how long the system takes to find the optimal path to any destination.) Figure 12 shows that nodes typically receive an update for each destination every 8 seconds. This wait is shorter than might be expected because, in the absence of failures, nodes receive recommendations from *two* rendezvous nodes, and the reports from those nodes are not synchronized. As a result, the recommendations arrive uniformly distributed in the 15 second routing interval. In addition, for destinations in the same row (column), nodes receive routing recommendations from all $\sqrt{n}-1$

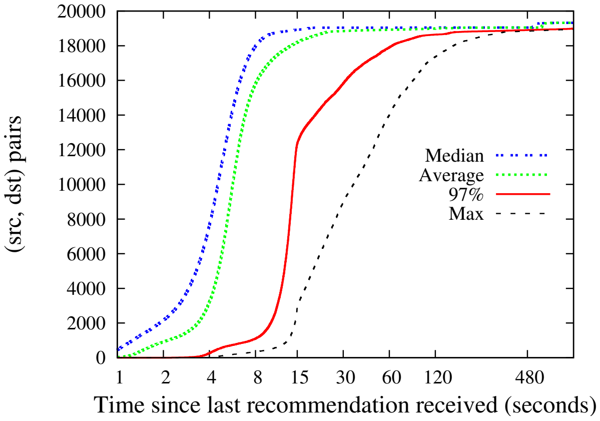


Figure 12: Route freshness (x-axis, shown on log scale) for all source and destination pairs (y-axis), calculated at 30 second intervals. The median line is calculated, for each (src, dst) pair, as the median route freshness across all 30 second intervals.

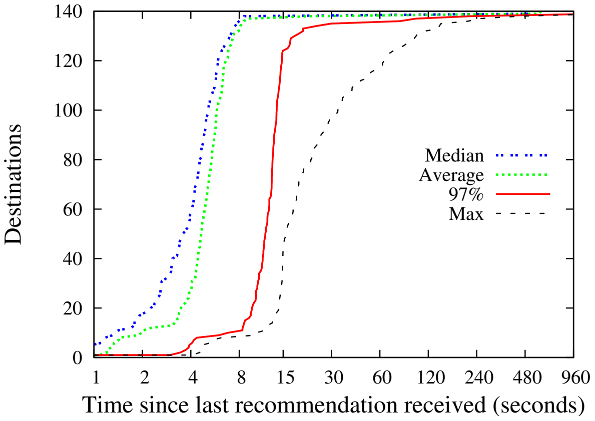


Figure 13: Route freshness (x-axis, shown on log scale) to all destinations (y-axis) from a node with *good* connectivity. This node had an average of only 5.2 concurrent link failures (max: 16).

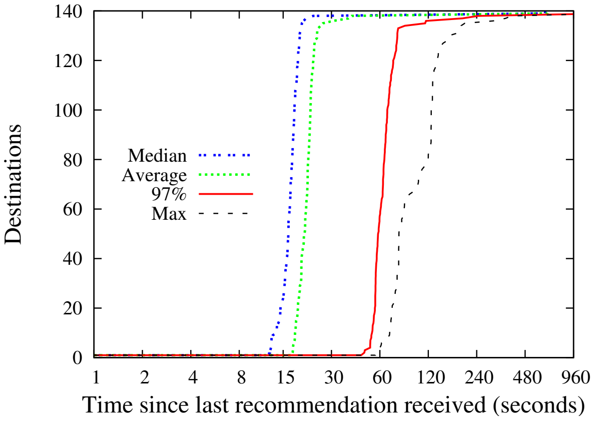


Figure 14: Route freshness (x-axis, shown on log scale) to all destinations (y-axis) from a node with *bad* connectivity. This node had an average of 44 concurrent link failures (max: 123).

nodes (including the destination) in that row (column), resulting in ≤ 4 second freshness for these destinations.

In our implementation, when a rendezvous server sends recommendations to its clients, it uses any measurements sent to it within the last 3 routing intervals, or $3 \cdot r = 45$ seconds. We do this to provide extra redundancy in case of dropped link-state messages from some of the rendezvous clients.[11] As a result, the information on which these routes are based, at the time the recommendation is received, is at most $p + 3 \cdot r = 75$ seconds old.

Because of these factors, freshness using our algorithm is high. 97% of the time, a "typical" (median) path's freshness is under 12 seconds. The median path experienced a worst-case freshness of only 30 seconds over the duration of the measurements. This worst case occurred despite the particularly high load and latency that PlanetLab was experiencing during the experiment.

To explore these results in more detail, we examine the behavior of two specific nodes: one "well-connected" node that had an average of only 5.2 concurrent failures, and one "poorly connected" node that experienced 44 concurrent link failures on average. From this observation, we hope to understand what effect, if any, the rare instances of staleness would have on the system's performance.

Figure 13 shows the distribution of update freshness (in log scale) for the well-connected node. Figure 14 shows the same distribution for the poorly connected node.

A node with good connectivity receives routing recommendations for every destination, on average, every 8 seconds (Figure 13). The 97% line shows that for nearly all destinations, over 97% of the time (in 30 second intervals), this node received a recommendation within 30 seconds.

Even a poorly connected node will, 97% of the time, receive updates for nearly all destinations within one minute (Figure 14). Note that some amount of staleness is not unique to the quorum algorithm: A full-mesh link state algorithm such as used in RON is also susceptible to link failures or packet loss that reduces routing freshness. Both systems' performance could likely be improved by making link-state announcements reliable, at the cost of additional complexity and some bandwidth.

Evaluation summary. *The grid quorum based routing algorithm effectively and rapidly finds optimal one-hop overlay routes (Figures 12–14) even in the presence of numerous link failures and high packet loss (Figure 8). It achieves this effectiveness while scaling far better than prior overlay routing systems (Figure 9). We therefore conclude that both the quorum system and the failover mechanisms are effective both in theory and practice in the Internet environment represented by PlanetLab.*

7. CONCLUSION

In this paper, we explored a novel algorithm for one-hop full-mesh routing based on grid quorum systems. Our algorithm informs all nodes in the system of their optimal one-hop route to every other node using only $\theta(n\sqrt{n})$ communication instead of the $\theta(n^2)$ communication required by previous systems. We presented solutions to the practical problems of efficiently handling non-square numbers of overlay participants and coping with communication failures. Applying this algorithm to an existing overlay network application, we found through simulation and deployment on PlanetLab that the theoretical scaling results hold in practice: A Resilient Overlay

[11] If we had also included timestamps, we could later choose the most up-to-date best hop recommendation.

Network (RON) required less than half the bandwidth to run on 140 nodes using our algorithm (15Kbps) than the earlier full-mesh variant (35Kbps). Despite the many failures encountered during our deployment tests, the algorithm rapidly and efficiently determined optimal routes for the overlay nodes.

We believe that the algorithm presented in this work offers an exciting step in scaling full-mesh networks and overlays, as well as possible generalizations to settings such as dense multi-hop wireless networks, social networks, and potentially even networks such as multiprocessors. Our results provide a path for future work in applying overlay systems to new classes of applications that are either too large for prior techniques or could not afford their overhead. In addition, this work remains open to further algorithmic refinement in a number of directions.

Challenges for larger overlays: While these systems have mostly aimed at smaller, trusted confederations of nodes (e.g., nodes in VPNs, special-purpose overlay nodes, or collaborative applications), our results show a way to scale these overlays to new classes of applications. For instance, our prototype can already support the number of nodes found in many peer-to-peer scenarios (e.g., a few hundred nodes in a BitTorrent swarm [5]). Such an extension, however, would also invite a number of new research challenges. For instance, earlier overlays dealt with many difficult security issues by assuming that nodes were mutually trusting. While such an assumption is reasonable in the case of a ten-party video conference, it certainly does not hold in a 300 party peer-to-peer swarm. Future work must address the question of how these networks can resist attacks against the routing mechanisms (e.g., malicious rendezvous nodes) and the data plane.

Integration with other scaling techniques. As we noted, there are several other techniques that researchers have proposed to improve the scalability of routing overlays. An attractive area of future study is how well these and our techniques can be combined, perhaps to create near-optimal routing overlays that scale to thousands or tens of thousands of nodes.

Appendix

A. IS IT POSSIBLE TO DO BETTER?

We wish to give a lower bound on the amount of per-node communication required to find the optimal one-hop route between all pairs of nodes in a dense overlay. Our result applies to any algorithm that finds optimal routes by doing direct comparisons of every alternative 1-hop path between two nodes. Both our routing algorithm and RON's are in this class of algorithms.

Each pair of alternative one-hop paths corresponds to a diamond in the overlay graph. Our technique for showing this bound uses a counting argument for the maximum number of such comparisons a node can make when receiving some edge weights. We begin with a definition and two lemmas.

Definition 1. A *diamond*, denoted $a-b-c-d$, is an undirected graph with edges (a,b), (b,c), (c,d), and (d,a).

Lemma 2. *There are* $3\binom{n}{4}$ *unique diamonds in the complete graph.*

Proof. Every choice of four nodes $\{a,b,c,d\}$ gives three possible diamonds, $a-b-c-d$ (square), $a-b-d-c$ (hourglass), and $a-c-b-d$ (bow tie). □

Lemma 3. *Every set of e edges forms at most* e^2 *diamonds.*

Proof. By induction on the number of edges. In the base case, $e=4$ can form at most 1 diamond.

By the inductive hypothesis, any e edges form at most e^2 diamonds. Now we show that the $e+1$st edge, (a,b), can be part of at most $2e$ *new* diamonds involving itself and any of the earlier e edges. Any two edges on four distinct nodes can be part of exactly two diamonds. For each of the earlier e edges, if the earlier edge is of the form (c,d), with no nodes in common, then it can contribute at most two new diamonds. If the edge is of the form (c,a), with one node in common, then it will only form a new diamond if there are edges (c,d) and (d,b) among the earlier e. Thus, we charge this new diamond to edge (c,d), and have already bounded the number of such in the earlier part of our argument. We conclude that with $e+1$ edges there can be at most $e^2+2e \leq (e+1)^2$ diamonds. □

Theorem 4. *Any algorithm that requires that each diamond's edge weights be found at some node has* $\Omega(n\sqrt{n})$ *per-node communication.*

Proof. Suppose every node receives the weights of e edges. In total, all nodes together would only be able to compare ne^2 diamonds, by Lemma 3. Since there are $\Theta(n^4)$ diamonds (Lemma 2), e needs be $\Omega(n\sqrt{n})$. □

It is an open question whether the lower bound holds for all algorithms: perhaps algebraic techniques, such as those used for fast matrix multiplication, could result in reduced communication. Regardless, this result tells us that, to improve upon the algorithm given in this paper, vastly different techniques, or additional assumptions, would be required.

Acknowledgments

The authors thank Jacob Scott, Himabindu Pucha, Vyas Sekar, and Vijay Vasudevan for helpful discussions. David Andersen and Amar Phanishayee were funded in part by the National Science Foundation under award CNS-0716273. David Karger was supported by a grant from the National Science Foundation. Additionally, David Sontag was supported by a NSF Graduate Research Fellowship, and Amar Phanishayee by an IBM Research Fellowship.

References

[1] I. Abraham, A. Badola, D. Bickson, D. Malkhi, S. Maloo, and S. Ron. Practical locality-awareness for large scale information sharing. In *Proc. 4th International Workshop on Peer-to-Peer Systems (IPTPS)*, Ithaca, NY, Feb. 2005.

[2] Akamai-sureroute. Akamai 1Q 2002 results. `http://www.akamai.com/en/html/about/press/press343.html`, Apr. 2002.

[3] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Resilient Overlay Networks. In *Proc. 18th ACM Symposium on Operating Systems Principles (SOSP)*, pages 131–145, Banff, Canada, Oct. 2001.

[4] S. Y. Cheung, M. H. Ammar, and M. Ahamad. The grid protocol: A high performance scheme for maintaining replicated data. *IEEE Transactions on Knowledge and Data Engineering*, 4(6):582–592, 1992.

[5] B. Cohen. Incentives build robustness in BitTorrent. In *Workshop on Economics of Peer-to-Peer Systems*, Berkeley, CA, USA, June 2003.

[6] J. Cowling, D. R. K. Ports, B. Liskov, R. A. Popa, and A. Gaikwad. Census: Location-aware membership management for large-scale distributed systems. In *Proceedings of the 2009 USENIX Annual Technical Conference*, San Diego, CA, USA, June 2009. USENIX.

[7] W. Cui, I. Stoica, and R. H. Katz. Backup path allocation based on a correlated link failure probability model in overlay networks. In *IEEE International Conference on Network Protocols (ICNP)*, Paris, France, Nov. 2002.

[8] S. M. Das, H. Pucha, and Y. C. Hu. Performance comparison of scalable location services for geographic ad hoc routing. In *Proc. IEEE INFOCOM*, Miami, FL, Mar. 2005.

[9] T. Fei, S. Tao, L. Gao, and R. Guerin. How to select a good alternate path in large peer-to-peer systems. In *Proc. IEEE INFOCOM*, Barcelona, Spain, Mar. 2006.

[10] R. Ferreira, M. Ramanathan, A. Awan, A. Grama, and S. Jagannathan. Search with probabilistic guarantees in unstructured peer-to-peer networks. In *Proc. 5th IEEE Intl. Conf. on Peer-to-Peer Computing (P2P)*, 2005.

[11] K. P. Gummadi, H. V. Madhyastha, S. D. Gribble, H. M. Levy, and D. Wetherall. Improving the reliability of Internet paths with one-hop source routing. In *Proc. 6th USENIX OSDI*, San Francisco, CA, Dec. 2004.

[12] H. lee, J. L. Welch, and N. H. Vaidya. Location tracking using quorums in mobile ad hoc networks. *Ad Hoc Networks*, 1:371–381, 2003.

[13] C. Lumezanu, R. Braden, D. Levin, and N. S. aand Bobby Bhattacharjee. Symbiotic relationships in Internet routing overlays. In *Proc. 6th USENIX NSDI*, Boston, MA, Apr. 2009.

[14] D. Malkhi, M. Reiter, and R. Wright. Probabilistic quorum systems. In *Proc. 16th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 267–273, 1997.

[15] A. Nakao, L. Peterson, and A. Bavier. A routing underlay for overlay networks. In *Proc. ACM SIGCOMM*, Karlsruhe, Germany, Aug. 2003.

[16] A. Nakao, L. Peterson, and A. Bavier. Scalable routing overlay networks. *Operating Systems Review*, 40(1):49–61, Jan. 2006.

[17] M. Piatek, T. Isdal, A. Krishnamurty, and T. Anderson. One hop reputations for peer to peer file sharing workloads. In *Proc. 5th USENIX NSDI*, San Francisco, CA, Apr. 2008.

[18] S. Savage, A. Collins, E. Hoffman, J. Snell, and T. Anderson. The End-to-End Effects of Internet Path Selection. In *Proc. ACM SIGCOMM*, pages 289–299, Cambridge, MA, Sept. 1999.

[19] J. Stribling. Planetlab - all pairs pings. `http://pdos.csail.mit.edu/~strib/pl_app/`, Nov. 2005.

[20] Y. Zhang, N. Duffield, V. Paxson, and S. Shenker. On the constancy of Internet path properties. In *Proc. ACM SIGCOMM Internet Measurement Workshop*, San Fransisco, CA, Nov. 2001.