Scalable Routing Overlay Networks

Akihiro Nakao
The University of Tokyo
7-3-1 Hongo, Bunkyo-ku
Tokyo, 113-0033, Japan
nakao@iii.u-tokyo.ac.jp

Larry Peterson
Princeton University
35 Olden Street
Princeton, NJ, 08540, U.S.A.
llp@cs.princeton.edu

Andy Bavier
Princeton University
35 Olden Street
Princeton, NJ, 08540, U.S.A.
acb@cs.princeton.edu

ABSTRACT

Routing overlays have become a viable approach to working around slow BGP convergence and sub-optimal path selection, as well as to deploy novel forwarding architectures. A common sub-component of a routing overlay is a *routing mesh*: the route-selection algorithm considers only those *virtual links* inter-node links in an overlay—in the routing mesh rather than all N^2 virtual links connecting an N-node overlay. Doing so reduces routing overhead, thereby improving the scalability of the overlay, as long as the process of constructing the mesh doesn't itself introduce overhead.

This paper proposes and evaluates a low-cost approach to building a topology-aware routing mesh that eliminates virtual links that contain duplicate physical segments in the underlying network. An evaluation of our method on PlanetLab shows that a conservative link pruning algorithm reduces routing overhead by a factor of two without negatively impacting route selection. Additional analysis quanti es the impact on route selection of defining an even sparser mesh on top of the topology-aware routing mesh, documenting the cost/benefit tradeoff that is intrinsic to routing. It also shows that constructing a sparser routing mesh on the topology-aware routing mesh—rather than directly on the Internet—itself benefits from having the reduced number of duplicate physical segments in the underlying network, which improves the resilience of the resulting routing mesh.

Categories and Subject Descriptors

C.2 [**Computer-Communication Networks**]: General

General Terms

Design, Algorithm

Keywords

Overlay Networks, Routing, Infrastructure

1. INTRODUCTION

Routing overlays have recently been proposed as an approach to improving the robustness, performance, and functionality of packet delivery over the Internet [4, 6, 7, 8, 14, 15, 27, 30]. To improve robustness, overlays rapidly find an alternative path when the current BGP-provided route fails [16, 26]. To improve performance, overlays select the best of multiple alternative paths. To improve functionality, overlays run application-specific code on intermediate nodes [30]. Routing overlays accomplish these goals by monitoring the available paths, resulting in a fundamental trade-off between how aggressively the overlay monitors the network (i.e., how many alternatives it monitors and how frequently it probes them), and the size to which the overlay is able to scale.

Existing routing overlays typically treat the Internet as a black box, relying on end-to-end performance measurements (primarily latency probes) to select routes. In RON [4], for example, each node actively monitors each of the other nodes, watching for opportunities to use an indirect path through an intermediate node should the direct path fail or suffer from lesser performance. Evidence indicates that such an N^2 approach is able to scale to approximately 50 overlay nodes.

In general, one can view a routing overlay as constructing a substrate for routing packets, often called a *routing mesh*, and then monitoring the *virtual links* on this mesh to select the best route between any pair of nodes. RON uses a fully connected mesh, where each virtual link corresponds to the default Internet path between a pair of nodes. An alternative is to first build a more sparse mesh—one with fewer than N^2 edges—and then actively monitor only the edges in this mesh to select routes. If one could construct a sparse mesh that eliminates virtual links (mesh edges) that are never selected by the routing algorithm, then probing costs will go down and scalability will improve without harming the overlay's ability to select good routes. If the overlay wishes to further prune the mesh, ideally such pruning will trade off goodness for sparseness in a reasonable way.

This raises the question of how to best select which virtual links to keep in the routing mesh. An approach used by several single-source application-level multicast overlays [6, 7, 8, 14] is to construct a self-organizing routing mesh. Like RON, however, such systems also treat the Internet as a black box, and depend on end-to-end performance metrics to determine which nodes should peer with each other in the mesh.

Another approach is to exploit static topology information about the Internet to build a routing mesh that is representative of the underlying physical network [19]. The resulting *representative mesh* attempts to eliminate virtual links (mesh edges) that have duplicate physical segments in the underlying network. A representative mesh can be defined at different levels of granularity—e.g., at the autonomous system (AS) level or at the router level—but in general, selecting virtual links that share as few underlying physical links as possible both reduces redundant traffic on the physical links and eliminates fate sharing in the case of link failure.

This paper describes and evaluates a system, called PLUTO, that builds a representative mesh to be used by routing overlays. We extend a preliminary analysis of a representative mesh [19] by introducing PLUTO in two stages. First, we present a conservative heuristic to identify and remove mesh edges that do not contribute to route selection; we call these edges *redundant* since the routing algorithm is likely to select other indirect paths. This heuristic uses only passive measurements and seldom-changing topology information such as AS-level topology and geographical information; it does not itself add to monitoring overhead. Experiments on PlanetLab [24] show that using this heuristic, PLUTO is able to reduce monitoring costs with marginal negative impact on route selection.

In the second stage, we explore the cost/benefit trade-off in more detail. Specifically, we evaluate the impact of pruning the mesh constructed in the first stage even further, but at the cost of impacting the overlays ability to select the best possible paths.

2. REPRESENTATIVE MESH

A factor in the scalability of any routing overlay is the total amount of overhead—link probes and route update dissemination—that it introduces into the network. The intuition is that any reduction of total overhead produced by an overlay of a particular size provides an opportunity to scale the overlay to more nodes.

Generally speaking, reducing the number of virtual edges in the routing mesh is the simplest way to cut down the overhead. More formally, suppose overlay networks use ping to measure network properties periodically at the rate of p per peer and employ a link state algorithm to disseminate the update at the rate of r per peer. Also suppose the overlay node i has n_i neighbors in a mesh with N overlay nodes ($1 \leq i \leq N$). The total ping traffic P and link state traffic R in the entire mesh are given as follows. [1]

$$P = \Sigma_{i=1}^{N} p n_i = \alpha p N^2 \quad (1)$$

$$R = \Sigma_{i=1}^{N} r n_i^2, \quad \alpha^2 r N^3 \leq R \leq \alpha r N^3 \quad (2)$$

where $\alpha = E/N^2$ is the edge-reduction parameter indicating reduction in the number of virtual links $E = \Sigma_{i=1}^{N} n_i$ versus the fully connected mesh. ($\alpha = 1$ means the graph is fully connected.) For example, a method that sets α to be inversely proportional to N makes P and R proportional to $O(N)$ and $O(N^2)$, respectively.

Although it is straightforward to create a sparser routing mesh by removing virtual edges at random, this approach might eliminate a link with good network properties; in the worst case, the network might become partitioned. The question is how to remove virtual links to define a sparse but useful routing mesh. This problem is immediately complicated by the fact that routing overlays sometimes aim to achieve resilience. Even if a virtual link performs poorly, we may wish to retain it in the mesh for use when other links fail. It is important to cover as many independent routes as possible in the underlying network, which contradicts our goal of eliminating as many virtual links as possible. Supporting routing overlays such as RON complicates this problem further, since unlike for single source multicast overlays, we need to support resilience for all possible pairwise connections; an unnecessary virtual link for one could be a crucial one for others.

Accordingly, the insight into solving this problem is to eliminate a direct virtual link when there are alternate multi-hop virtual links bearing similar network characteristics. Since the goal is scalability, inexpensive information must be exploited: either static or passive information that the Internet has already collected for its own operation. The mesh building strategy uses inexpensive hints to find sets of three virtual links, such that one virtual link is almost identical in the underlying physical topology to the concatenation of the other two virtual links. For each such set, the strategy removes the former virtual link since it is expected to bear similar network properties to the combination of the latter two. As a result, duplicate segments in the underlying network are removed from the routing mesh, yielding a mesh that resembles the underlying topology as much as possible. This routing mesh is referred to as a *(physically) representative mesh*. Although the mesh may introduce more hops in the optimal route, it is often not necessary for every node along the path to forward packets. We will discuss this in more detail in Section 4.6.

[1]The first inequality in (2) follows from $\Sigma_{i=1}^{N} n_i^2 \geq (\Sigma_{i=1}^{N} n_i)^2/N = E^2/N$ since $\Sigma_{i=1}^{N}(n_i - n)^2 \geq 0$ where $n = \Sigma_{i=1}^{N} n_i/N = E/N$. The second follows from $\Sigma_{i=1}^{N} n_i^2 \leq NE$. We express P and R in terms of α and N instead of E and N for the sake of simplicity in our discussion.

3. PLUTO ARCHITECTURE

This section describes PLUTO (PlanetLab Underlay Topology services for Overlay Networks), which defines a two-layer routing hierarchy. The bottom layer supports a *topology discovery service* that reports static topology information about the underlying network, typically extracted from information that the Internet has already collected for its own operation. The upper layer builds a representative mesh using the topology discovery service. Although PLUTO was designed and implemented on PlanetLab, we believe that is useful for generic overlay networks.

3.1 Topology Discovery

PLUTO implements two topology discovery operations. The first,

```
Path = GetASPath(src, dst)
```

returns the *verified AS path* traversed by packets sent from IP address src to IP address dst. Note that this operation maps a pair of network prefixes to the sequence of AS numbers that connect them, much like a BGP routing table maps a network prefix to an AS path. Also, this operation must limit the src and dst to addresses of overlay nodes because PLUTO needs a point-of-presence within an AS in order to resolve this query. The second operation,

```
PG = GetASGraph()
```

returns the *peering graph* (PG) for the Internet. This graph represents the coarse-grain (AS-level) connectivity of the Internet, where each vertex in PG corresponds to an AS, and each edge represents a peering relationship between ASes. The Internet does not currently publish the complete PG, but it is easy to construct an approximation of the PG by aggregating BGP routing tables from multiple vantage points in the network, as is currently done by sites like RouteViews [3] and FixedOrbit [1]. That is, an edge exists between any two vertices X and Y in PG if some BGP routing table contains a path in which ASes X and Y are adjacent.

Figure 1 sketches the implementation of these two operations. We assume each AS that hosts one or more overlay nodes has a BGP router that feeds BGP updates to a PLUTO service. A given overlay node sends one of the above queries to the local PLUTO service, which answers the query immediately if it can, otherwise it redirects the query to an appropriate PLUTO service that can answer the query. The answer to the query is then cached in the local PLUTO service. For example, suppose the overlay node IP_x in the figure issues a GetASPath query to learn the AS path between IP_y and IP_z. The PLUTO service in AS X determines that IP_y is located in AS Y using the translation service described below, and redirects this query to the PLUTO service in AS Y since this instance of PLUTO should be able to answer the query about AS paths originating from AS Y.

Note that while PLUTO assumes that a BGP feed is available in every AS, this is difficult in practice. Thus, for the sake of evaluating PLUTO, we developed an "AS Traceroute" service as a part of the GetASPath operation. The implementation translates a traceroute result into an AS path using the translation service, thereby emulating a BGP-based implementation of GetASPath. Since AS paths have no cycles, we modified the traceroute program so that it performs a binary search for AS boundaries, sending multiple UDP

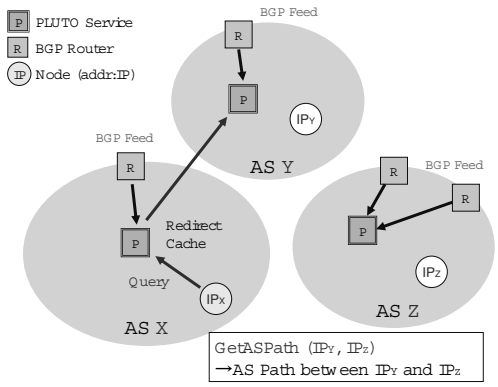


Figure 1: PLUTO topology discovery sub-services

probing packets in parallel. Since AS hop count is much less than router hop count, this method suppresses probe traffic significantly. PLUTO also caches AS paths derived in this way, and updates this information infrequently.

In addition to these two topology-related operations, PLUTO also provides three translation services. The first,

GeoInfo = IP2Geo(addr)

returns geographical information GeoInfo corresponding to the given IP addr. This information includes the longitude/latitude coordinates where the corresponding machine is located. Currently, we assume the overlay nodes report their own geographical locations with their IP addresses to PLUTO.

In this paper, we often compute geographic distance between two points using longitude and latitude. This distance D (along the great circle) is easily derived as $D = R \cdot \arccos[\sin\theta_1 \cdot \sin\theta_2 + \cos\theta_1 \cdot \cos\theta_2 \cdot \cos(\phi_1 - \phi_2)]$, where R is the radius of the Earth, (θ_1, ϕ_1), (θ_2, ϕ_2) are (latitude,longitude) coordinates (in radian) of two points.

The second operation,

ASN = IP2AS(addr)

returns the AS number ASN that the given IP addr belongs to. This can be done by selecting the route in a BGP table that matches a given IP address, and then inferring that the last AS number on the path in that route is the AS that contains the node with the given address. We take a simpler approach than that of Mao et al. [18] and make this inference using a collection of BGP tables to improve the accuracy. We also take the Internet Exchange Points (IXP) addresses into account [2]: if a given IP address corresponds to an IXP and if that IXP has an AS number, report it; otherwise report that it is an IXP without an AS number.

The last operation,

PoPInfo = AS2PoP(ASN)

returns point-of-presence (PoP) information PoPInfo for the given AS number ASN. This information includes the list of IP addresses and geographical locations of PoPs of the AS of ASN. PLUTO currently registers PoP geographical locations of 68 large ASes obtained from Rocketfuel[29]. This information is expected to be updated very infrequently.

3.2 Mesh Construction

We implement a mesh construction service, called PLUTO-Mesh, on top of topology discovery operations just described. The service constructs an AS-level representative mesh using passive and static information. We currently assume every node has a list of the other nodes in the overlay network at the time the mesh is created, although it would be straightforward to periodically update the mesh to incrementally incorporate new nodes. However, the intent is that the mesh not change frequently, but instead, the routing overlay running on top of the mesh run its own membership protocol to determine which nodes and links in the mesh are currently "up". That is, rapidly adjusting to nodes joining and leaving the mesh is viewed as part of the routing problem rather than the mesh construction problem. For overlays that include both a stable core of nodes a dynamic set of end-user nodes, a hybrid approach would be required.

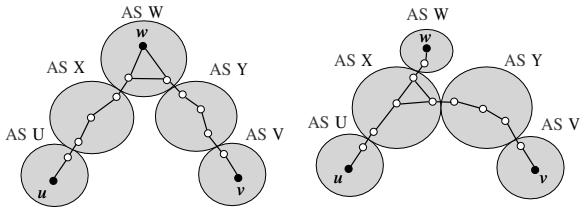
3.2.1 AS-Level Pruning

PLUTO-Mesh identifies and removes topologically redundant virtual links (at the AS level) between overlay nodes, or said another way, retains only those edges that we can determine to be independent in the underlying AS-level network. It does this in a distributed fashion. That is, an instance of PLUTO-Mesh runs on each overlay node, and determines which other overlay nodes are neighbors in the mesh based on calls to the GetASPath (and optionally GetASGraph) operations. The entire global mesh could be formed by aggregating these neighbor sets, but many routing overlays maintain only immediate neighbor sets at each node and do not need the global topology.

Our approach is limited to edges that can be removed without building a global picture of the network. An alternative strategy would be for a central authority to collect global network information, build the entire mesh, and distribute it throughout the overlay. We opt for a localized approach for reasons of scalability and cost—e.g., it is practical to update the mesh whenever AP paths change—although the resulting mesh may not be as sparse as that produced by a centralized algorithm.

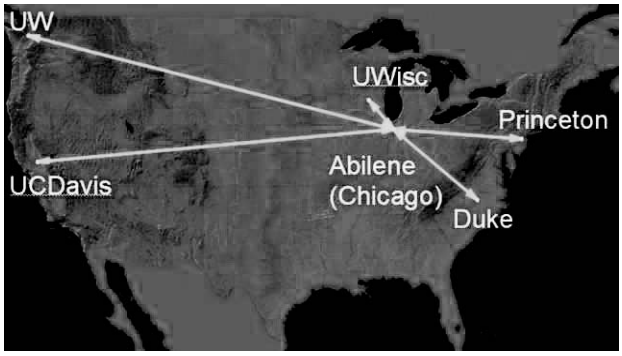
Our algorithm prunes an edge from the local node u to remote node v if the AS path from u to v includes AS W, such that there is an overlay node w that is located within AS W. This scenario is illustrated in Figure 2(a). Although each node u runs this algorithm locally, it will not cause a resulting mesh to be partitioned, since it prunes the edge (u, v) only when it finds a physically similar alternate path to reach v.

In addition, we prune edge (u, v) should an intermediate node w reside in an AS that is directly connected to the path from u to v, as illustrated in Figure 2(b). PLUTO Mesh uses GetASGraph to narrow the scope of candidate intermediate nodes. When the links into and out of the AS that contains w are poor, our algorithm may prune a better direct edge from u to v. In order to avoid this situation, we optionally find out more precise information. For ex-

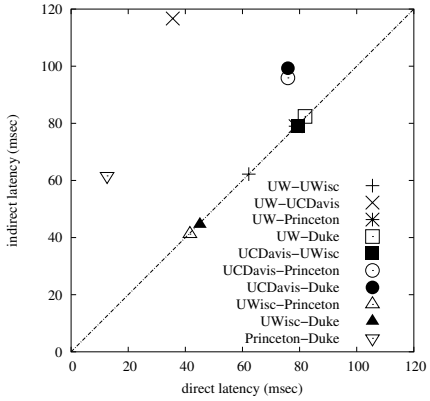


(a) Topology A (b) Topology B

Figure 2: Black dots u, v, and w denote overlay nodes and the white dots denote routers. Virtual link (u, v) is redundant and can be removed from the mesh, since edges (u, w) and (w, v) connect u to v.



(a) Simple PLUTO Mesh (N=6)



(b) Latency comparison between direct edge (u, v) and indirect edge (u, w, v) where w is a node in Abilene

Figure 3: Mesh Including Cross-Country Transit AS (Abilene AS 11537)

ample, we estimate latency from u to w and from u to v using the latency estimator discussed in 5.1, and do not prune the edge if the difference is greater than some threshold. Note that this scenario of pruning requires complex coordination between nodes so that the network may not be partitioned. In this paper, we focus only on the scenario show in Figure 2(a).

3.2.2 Geo-Based Pruning

Although our algorithm is very straightforward, AS-level information may be too coarse-grain to produce a satisfactory result, especially when we have large ASes spanning a continent. In Figure 2(a), if AS W is a cross-country AS, the default path (u, v) and its indirection path (u, w, v) may not have many duplicate route hops, and so may not bear similar network properties to (u, v). We have also implicitly assumed that there is only one overlay node per AS and have not dealt with the case where there are multiple nodes in (especially) large ASes. One simple solution is to let one of the nodes be a representative of the nodes in its AS. However, since the AS in question can cover a large area, this approach will not always work well.

To resolve these problems, we implement a *discriminator* that: (1) preserves good direct links, and (2) selects among multiple nodes in a single AS. The basic idea behind the discriminator is to use geographical information provided by the IP2Geo translation service. It uses the geographical location of the nodes and PoPs of cross-country ASes from the AS2PoP service, and leverages two assumptions: (1) large ASes are well-connected, hence, geographical distance within AS should be highly correlated with latency, and (2) regional ASes are connected to the geographically nearest PoPs of a large AS.

Preserving Good Links

Our strategy is to prune the default path (u, v) when we expect the path (u, w, v) has similar network properties, using only AS-level information. Figure 3(a) illustrates the idea with a simple mesh created using the algorithm defined up to this point. It contains six nodes spanning six ASes: UW, Princeton, Duke, UCDavis, Abilene, and UWisc. In this case, Abilene is a large AS spanning the U.S.; we select an overlay node in the middle of the country. This mesh forms a tree rooted at Abilene, since every AS path between ASes other than Abilene goes through Abilene; our algorithm has pruned a direct link (u, v) and replaced it by the indirect path (u, w, v), where in this case w is a node in Abilene.

Figure 3(b) compares the direct latency between (u, v) and the indirect latency of (u, w, v), showing only the latency between pairs where we prune the direct links. In other words, this figure compares the latency before/after pruning (u, v). As can be seen, except for a couple of anomalies, direct latency and indirect latency are almost the same. We see anomalous cases for (Princeton, Duke) and (UW, UCDavis) where indirect latency is much higher than the direct latency.

The reason is that at the router level, packets travel through the PoPs of Abilene that are closer to the end points. For example, between UW and UCDavis, packets go through PoPs at Seattle and Sunnyvale, while between Princeton and Duke, they go through a PoP in Washington D.C. Therefore, if we rely on only AS-level information, we might prune good direct links when we have a large cross country AS along the route. It is necessary to avoid getting rid of these direct links.

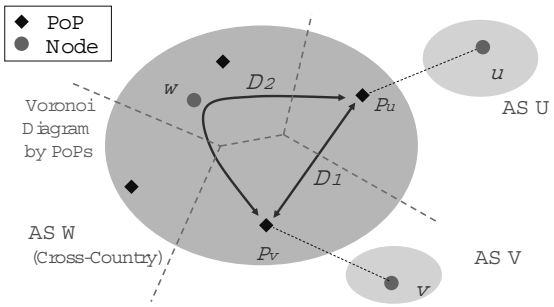
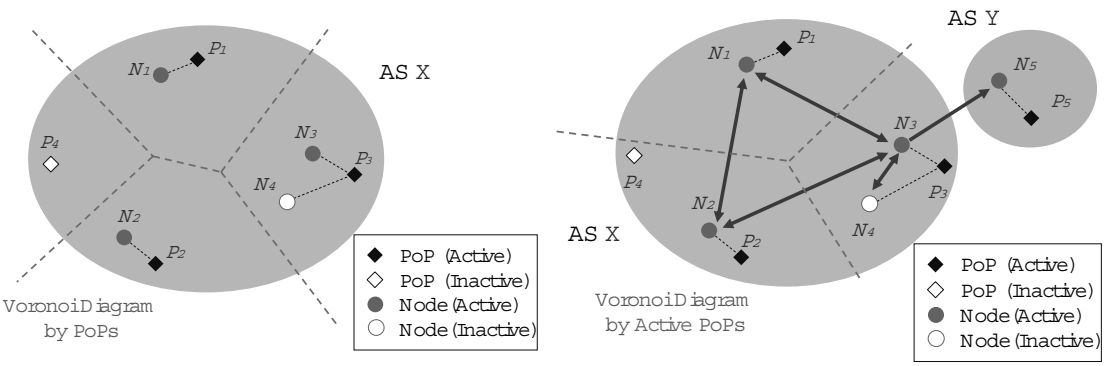


Figure 4: Mesh algorithm 1 (preserving good direct links)

Figure 4 illustrates how our discriminator works to preserve good direct links. Suppose we consider removing direct link (u, v) because we believe indirect link (u, w, v) has similar network properties. In order to assess the latency difference between (u, v) and (u, w, v), we first draw a voronoi diagram using geographical locations of PoPs. The voronoi diagram partitions the space such that each cell contains exactly one PoP and every point in a given cell is closer to that cell's PoP than to any other PoP. We then identify the nearest PoPs in W, P_u and P_v for the nodes u and v, respectively. As mentioned, we assume that regional ASes connect to the nearest PoP of the cross-country AS they subscribe to; this means, for



(a) Step1: classifying nodes

(b) Step2: construct virtual edges

Figure 5: Mesh algorithm 2 (discriminating multiple nodes)

example, that packets from u will enter AS W at P_u. We next compare the geographical distance D_1 and D_2, and if D_2 differs significantly from D_1, we elect to keep the direct edge (u, v). Here, we assume that a large AS should be well connected, so geographical distance should reflect latency. More specifically, we define packet traversal time difference $\Delta t = |D_1 - D_2|/v$ where v is effective light speed in fiber discussed in more detail in Section 5.1.2, and if $\Delta t > \Delta t^*$, we keep the direct edge. We currently set Δt^* to 5 msec, which successfully removes the anomalous points in Figure 3(b).

Discriminating Multiple Nodes

Another question is how we connect nodes when we have multiple nodes in the same AS. Figure 5 shows how our discriminator solves this problem in two steps. Note that every node runs the same algorithm locally.

[Step1: Classifying Nodes] Suppose we have 4 nodes N_1 to N_4 and 4 pops P_1 to P_4 in AS X. First, we subdivide the AS into voronoi cells by PoPs. PoPs P_1 and P_2 each have 1 node, P_3 has 2 nodes, and P_4 has no nodes. We then classify nodes as either *active* or *inactive*: the node closest to the PoP in each voronoi cell is considered active and all others are inactive. Similarly, we call a PoP active when it has at least one node; PoPs with no overlay nodes in their cells are considered inactive.

[Step2: Construct Virtual Edges] Once we identify active and inactive nodes, we connect edges between all the active nodes in the same AS, and also interconnect active nodes and inactive nodes sharing the same PoP. In addition, we construct an edge from an active node A to another active node B (in a different AS), when A's PoP has B's PoP in its region. At this point, we subdivide AS X into voronoi cells by active PoPs. Figure 5(b) shows that active node N_3 connect edge to N_5 since P_3 has N_5's PoP P_5 in its region.

PLUTO-Mesh uses AS2PoP to obtain PoP locations for a given AS. If it determines that a particular AS is not among the registered large ASes, PLUTO uses the node with the smallest IP address among the nodes in the same AS as a effective PoP location for that AS.

4. EVALUATION

This section reports on experiments designed to evaluate PLUTO-Mesh's ability to help routing overlays reduce redundant traffic without sacrificing performance and resilience. Our experiments use RON [4] as the example routing overlay, since designing a mesh to support a routing overlay like RON is more challenging than supporting a single-source multicast overlay (e.g., [6, 7, 8, 14]). Ideally a routing mesh should optimize all possible pairwise connections; RON is so aggressive in monitoring link behavior that one can view it as approximating the optimal route selection strategy. The experiments show that our mesh successfully removes superfluous virtual links and suppresses unnecessary traffic without degrading the performance of the routing overlay. A point of emphasis: we use RON as a worst-case benchmark of overlay routing overhead; one would not necessarily run the unmodified RON algorithm on top of the PLUTO-Mesh for reasons discussed at the end of this section.

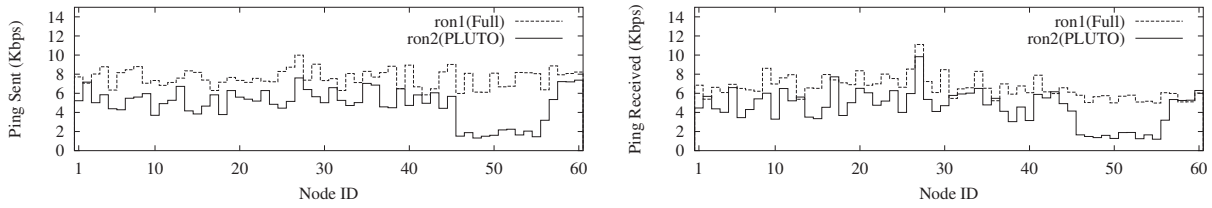
4.1 Redundancy Metrics

It is undesirable for multiple virtual links in a routing mesh to share underlying physical links, because shared physical links represent a single point of congestion or failure among seemingly disjoint virtual paths. We define two metrics that together attempt to capture the quality of a mesh composed of virtual links: *resilience* and *duplicates*. The *resilience* metric indicates how many disjoint minimum spanning trees of virtual links we can iteratively "extract" from the mesh. The *duplicates* metric then quantifies the disjointedness of the underlying physical paths by counting how many pairs of virtual links of the mesh share a physical link. We define two variants of the *duplicates* metric, counting both duplicate AS hops and router hops. For any two meshes with the same *resilience*, we prefer the one with the smaller *duplicates* metric.

4.2 Experimental Setup

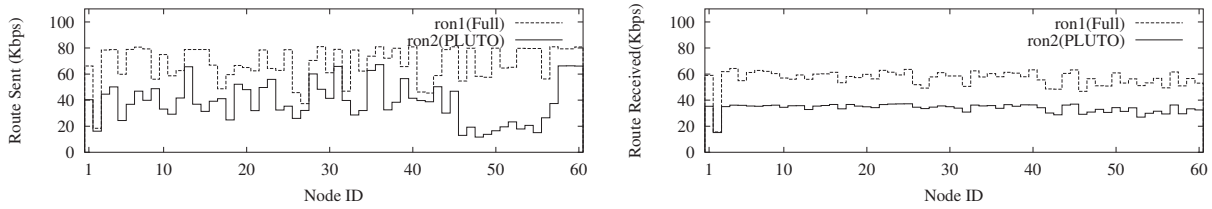
To evaluate the mesh constructed by PLUTO, we run one instance of RON (`ron1`) on a fully connected mesh, and *at the same time on the same set of nodes*, a second instance of RON (`ron2`) on top of PLUTO-Mesh. We run both instances of RON for 40 minutes across 60 PlanetLab nodes geographically distributed throughout the U.S. We then compare routing tables and the amount of traffic both RON instances have generated.

We modified RON slightly for our experiments. Although the original RON implementation exchanges link-state information among all nodes in the overlay, it calculates only single-hop indirection routes as alternate routes based on the observation that single hop indirection gives us the best alternate path most of the time. We cannot rely on single hop indirection with PLUTO-Mesh since each node has a limited number of virtual neighbors. Our modified RON calculates the optimal multi-hop indirection route. Note that



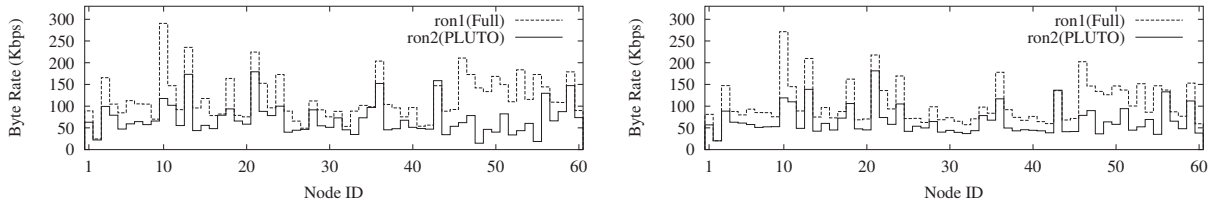
(a) Ping Sent (b) Ping Received

Figure 6: Ping traffic comparison



(a) Route Sent (b) Route Received

Figure 7: Routing update traffic comparison



(a) Byte Sent (b) Byte Received

Figure 8: Total traffic comparison

when run on the fully connected mesh, this method will find routes that are at least as good as those found by the original RON.

4.3 Mesh Sparseness

For the example set of 60 PlanetLab nodes, PLUTO-Mesh calculates a representative mesh with 1968 directed virtual links, as opposed to 3540 (=60x59) in the fully connected virtual topology. This reduction in the number of virtual links (α=55.6%) greatly suppresses the traffic of probes and route dissemination. Figures 6, 7, and 8 compare the various traffic (ping, route dissemination, and total traffic respectively) that both RON instances generate per node. Note that we use the routing overlay itself to bridge the disconnectivity between the public Internet and Internet2, and the total traffic also includes traffic resulting from disseminating routes through the overlay. According to these graphs, on average `ron2` carries about 50% of the traffic of `ron1`. Some nodes (especially those at network cross-roads) have significantly less traffic since these nodes do not have many virtual links.

4.4 Mesh Quality

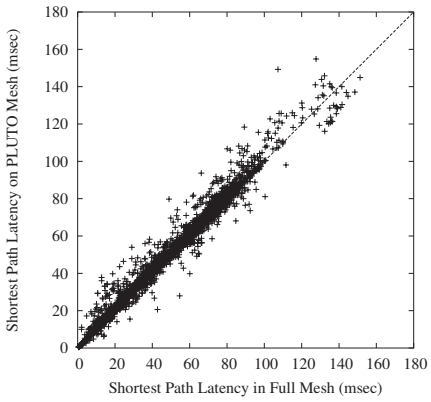
Next, we examine whether we have sacrificed performance by reducing the number of virtual links. To do this, we compare the optimal latency and bandwidth between every pair of nodes on `ron1` and `ron2`. The bandwidth of each virtual link is calculated using TCP congestion control equations [11, 21] and the measured latency and loss rate. This is the same as in the original RON, and although it may not represent real throughput, it is a reasonable comparison point. We use the modified Dijkstra algorithm to calculate bandwidth-optimized paths [10, 28].

As Figures 9(a) and 9(b) show, the performance degradation from using PLUTO-Mesh is minimal. Figure 9(a) shows there is strong correlation between the optimal latency in `ron1` and that in `ron2`. Note 61% of routes on `ron1` are better than the Internet. Figure 9(b) shows the cumulative distribution of a Relative Delay Penalty (RDP) and a Relative Bandwidth Penalty (RBP). RDP is calculated as a ratio of the latency difference (`ron2-ron1`) to the reference latency (`ron1`). RBP is defined as a ratio of the bandwidth difference (`ron1-ron2`) to the reference bandwidth (`ron1`), and indicates how much bandwidth we lose in `ron2` versus `ron1`.

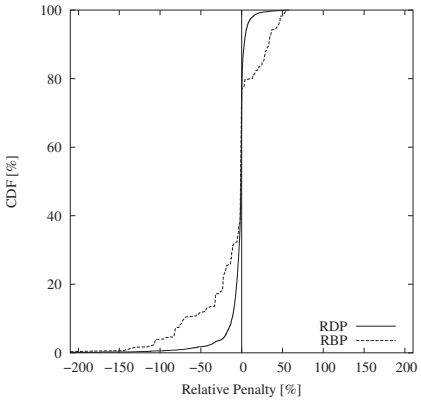
Note that sometimes, `ron2` (on PLUTO-Mesh) can achieve better performance than `ron1` (on the complete mesh). We believe this is partly because there exists some fluctuation in route convergence; also, because of the high-bandwidth usage incurred from running two instances of RON, a denser mesh has a higher chances of losing route updates.

4.5 Mesh Redundancy

We evaluate the mesh resilience by extracting MSTs from both graphs. Our off-line analysis shows that the fully connected mesh contains 20 MSTs (i.e., *resilience*=20), while our mesh includes 5 MSTs (*resilience*=5). Although this metric may not show the actual resilience of the mesh, it imply that our mesh still leaves us the ample possibility of finding alternate routes in case some of the links go down. However, it is not fair to compare the *dupli-*



(a) Pair-wise shortest path latency comparison



(b) Relative Delay/Bandwidth Penalty

Figure 9: Mesh quality comparison (latency and bandwidth)

cates metric between the complete mesh and PLUTO-Mesh since their *resilience* is significantly different. We explore the validity of PLUTO-Mesh in this regard in Section 5.

4.6 Discussion

The efficacy of any topology-aware mesh construction algorithm depends on the underlying topology of a given overlay network. PLUTO-Mesh is no exception. This paper has evaluated our mesh construction algorithm in PlanetLab, using about 60 nodes geographically distributed in the U.S. This set includes both nodes at the edge of the network, as well as at cross-roads inside the network (e.g., nodes co-located with Internet2 PoPs). Generally speaking, our approach works better the more overlay nodes there are close to PoPs inside the underlying network. This should not come as a surprise since more interior nodes results in more redundant edges that can be removed. As an aside, although we used PlanetLab nodes where node distribution is biased toward GREN (Global Research and Educational Network) [5], our mesh algorithm is independent of which kind of network the overlay belongs to. That is, the advantages of running a routing overlay may vary, but the value of our edge reduction strategy is more dependent on the breadth of overlay coverage than the specific link characteristics.

It appears that the more overlay nodes there are, the greater the opportunity to prune edges. In previous work [19], simulations on data collected from RouteViews and PlanetLab BGP feeds show that the parameter $\alpha = E/N^2$ decreases linearly as the number of nodes N increases, as shown in Figure 10. That is, with 1000 nodes, we expect PLUTO-Mesh to achieve about a 70% reduction (α=30%) in routing overhead. There is yet another way to interpret

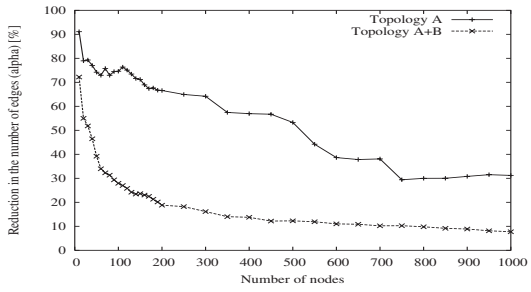


Figure 10: The reduction in the number of edges $\alpha = E/N^2$ as a function of the number of overlay nodes

the results presented in this paper. If our mission is to strategically pick a set of nodes to construct a routing overlay, we need to consider not only scalability but also node placement. Our results implies that it is obviously beneficial to include "inside" nodes rather than "edge" nodes, and to run our mesh algorithm to significantly reduce the overhead of routing.

Finally, a potential objection to sparse routing meshes is that they could increase the number of overlay hops in the optimal route. This may introduce extra delay and points of failure in a RON-like routing overlay that aims to achieve resilience and performance. We expect such routing overlays to use PLUTO in a smart way. The insight is that if (a part of) the optimal route consists of the default Internet route, the routing overlay does not have to forward the packets through the intermediate overlay nodes. PLUTO can use the AS information to help the overlay break down the optimal route into a smaller number of forwarding hops.

For example, suppose RON on PLUTO chooses the optimal overlay path (s, x, y, z, d), where s, x, y, z, d are overlay nodes, and this translates into an AS path (S, X, Y, Z, D). If RON can verify (with PLUTO's help) that the Internet route (s, y) translates into an AS path (S, X, Y) and (y, d) into (Y, Z, D), then it does not need to forward the packet through all the 3 intermediate hops x,y,z, but instead it can use only one intermediate hop y to forward packets to d. The point is that the overlay is always free to use the default Internet routes.

5. COST/BENEFIT TRADE-OFF

The PLUTO mesh provides clear benefits to routing overlays by enabling them to reduce unnecessary traffic. Our expectation is that this strategy is sufficient to support overlays containing a few hundred overlay nodes, provided they are reasonably distributed over the Internet. However, if the goal of a routing overlay designer is to scale to a thousand nodes, then the PLUTO mesh may not be sparse enough. For example, RON scales to about 50 nodes (2500 edges) on the fully connected mesh. Assuming that the number of edges in the routing mesh is a limiting factor in the overlay's scalability, we expect that a RON-like overlay could scale to a thousand nodes if the underlying mesh had only a few thousand edges. However, this would require that the edge reduction coefficient α be smaller than 1%, and our results indicate that the PLUTO mesh can achieve only around 70% edge reduction for 1000 nodes. Further edge pruning is clearly necessary to enable routing overlays like RON to scale to such dimensions. Specifically, it is necessary to drastically reduce

α, say, inversely proportional to N so the resulting sparse mesh will have an edge count that is linear in the number of nodes.

This section evaluates the cost and benefit of further pruning the PLUTO mesh to form an even more sparse graph. A more sparse mesh obviously reduces the traffic for both active probing and route dissemination, but at the cost of impacting the overlay's ability to select the best possible paths. To make the overlay topology even more sparse, we need to rely on more than AS topology and geographical information. The problem is that this information needs to be inexpensively obtained, otherwise we have traded one set of expensive probes for another. Toward this end, we develop the PLUTO Minimum Latency Estimator (MLE) to predict the packet round trip times between an arbitrary pair of nodes using only passive and infrequently updated information. PLUTO MLE infers the AS path between any two nodes based on the available BGP data, and then estimates the latency between the nodes using the geographic length of the inferred path. Our method does not actively probe the network at all, yet provides latency estimates that are on-par with other approaches like GNP [20].

Using PLUTO MLE, we then construct more sparse meshes by building k minimum spanning trees on the PLUTO mesh, and evaluate the sparseness, quality and redundancy of the resulting meshes. A key advantage of the resulting k-MST mesh is that its edge count is linear in the number of nodes; we expect that such a mesh could enable a routing overlay like RON to scale to a thousand nodes. We show that pruning a representative mesh into a sparse mesh gracefully degrades an overlay's ability to select the best possible paths. We also show that building a sparse mesh on top of the PLUTO Mesh provides better results than building the sparse mesh directly on the fully connected virtual topology.

5.1 Minimum Latency Estimator

The PLUTO Minimum Latency Estimator (MLE) estimates the latency between an arbitrary pair of nodes using the geographical locations of both end-points and PoPs in large ASes, along with existing BGP feeds from various vantage points. Previous research [22] has shown the difficulty of estimating latency using only the geographical locations of end-points, especially when the end-points are far away. We believe that the resulting inaccuracy is mainly caused by the fact that a packet trajectory between two end nodes is not geographically a straight line between them, but rather a zig-zag curve deflected by the various ASes it traverses. Our basic idea is to first infer the AS Path between arbitrary pair of end nodes, and then to sum up PoP-to-PoP geographical distances along the AS path to infer the end-to-end latency. Note that the reason that we must first infer the AS path between end nodes is that we do not have the BGP tables for every AS; if we happen to have a BGP feed at the AS containing one of the end-points, we can simply use the real AS path. Next we explain our strategy for inferring paths using only a limited number of BGP feeds, and then describe how to use this information to estimate the latency between nodes.

5.1.1 AS Path Inference

A simple method for inferring AS paths using only a handful of BGP feeds is to construct an AS connectivity graph (e.g., via the GetASGraph operation), and then to infer an AS path by calculating the shortest path on this connectivity graph. However, the problem is that this approach does not account for BGP policies that often result in selection of a path other than the one with the fewest AS hops. Previous work [19] shows that only 50% of 5500 examined paths are predicted by this shortest path method. Note that, in reality, prediction is much worse than 50%, since the connectivity graph is so dense that there are multiple shortest paths with the same AS hop count; the matching scheme simply verifies that the actual AS path is in the set of multiple shortest paths calculated from the BGP routes. In other words, the AS graph contains pairs of nodes having several hundred shortest paths of the same length. Inferring the actual path involves guessing which is the actual path from hundreds of candidates. Factoring in the AS peer relationship using Gao's method [13], and attempting to exclude *illegal* prediction (such as a customer AS carrying traffic for a provider AS), the prediction accuracy is only slightly better than using the naïve approach. We believe that the low accuracy of this simple path prediction method results from the fact that policies are so deeply embedded in BGP routing.

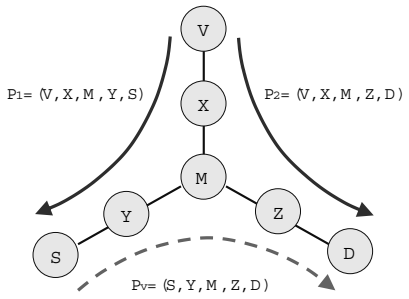
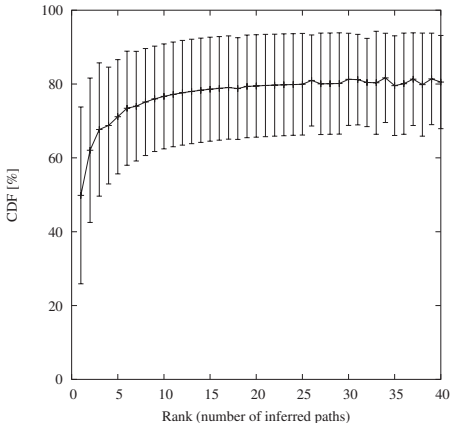


Figure 11: AS Path Inference

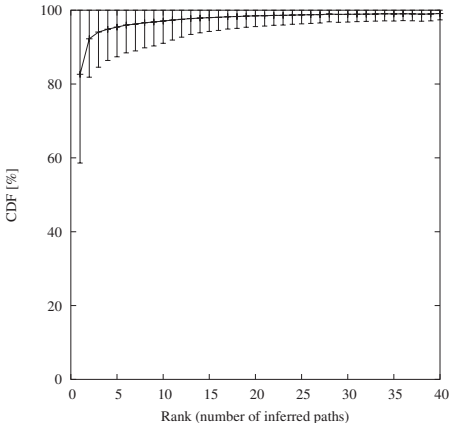
Our AS path inference approach is based on the following two insights. First, about 70% of the AS paths observed in the BGP tables obtained from RouteViews are symmetric. Second, it seems reasonable to assume that AS paths are usually *transitive*, meaning that if the AS path from A to C is (A, B, C), then the paths from A to B and from B to C are likely to be (A, B) and (B, C), respectively. Of course, this is not necessarily true, because AS paths are determined by network prefixes, rather than AS numbers.

We use these insights to infer AS paths as follows. Suppose we are trying to infer an AS path between a source node s (in AS S) and a destination node d (in AS D), by combining information from a number of different vantage points where we can access BGP tables. For each vantage point v in some AS V, we calculate the AS path from that vantage point to s, $P_1 = (V, X, M, Y, S)$, and also the AS path from that vantage point to d, $P_2 = (V, X, M, Z, D)$. Note that the AS paths are the same until they reach some AS M, at which point they diverge. Then we simply infer that the AS path from s to d is $P_v = (S, Y, M, Z, D)$, as shown in Figure 11. We repeat this process across all vantage points to obtain a set of AS path candidates, remove duplicates from the set, and then rank them in order of increasing AS hop counts.

To evaluate this strategy, we have examined BGP tables from 56 distinct vantage points using RouteViews and PlanetLab BGP feeds. For each vantage point in some AS X, we use the other 55 vantage points to infer the AS path from AS X to every single network prefix (there are typically 150,000 such prefixes) and then compare the results with the actual paths given by the BGP tables from AS X. Figure 12 shows the success of our method. The solid line in each subfigure shows the average across all vantage points, and the error bars show the standard deviation for individual vantage points. First, Figure 12(a) shows how often the actual path is within the top r ranked inferred paths. For example, on average, the actual AS path was one of the top five ranked candidates about 71% of the time, but the success of the method has standard deviation of 15%. We have found that the vantage points for which the method did not work as well are located in Europe and Asia, where we do

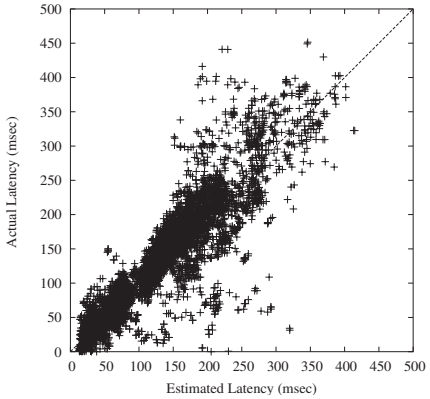


(a) Perfect Matching

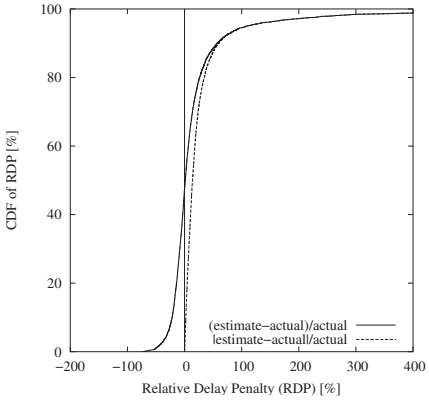


(b) Ambiguous Matching

Figure 12: AS Path Inference



(a) Comparison between estimated latency and actual latency



(b) CDF of Relative Delay Penalty (RDP) and its absolute variant

Figure 13: Minimum Latency Estimation

not have a concentration of vantage points. However, for vantage points in the U.S., our method often gives good results.

Second, we note that, for the purpose of predicting latency, it is not necessary to infer the *actual* AS path, but only one with *similar* geographic properties—namely, that traverses the same major PoPs. Based on this insight, we evaluate the candidate sets using the criterion of *ambiguous matching*, where we consider that an inferred AS path matches the actual AS path even if they differ by one AS. Figure 12(b) shows that this ambiguous matching ranks a path that is similar to the actual path within the top two over 90% of the time, and within the top five over 95% of the time. This result indicates that the top few AS paths inferred by our method should be useful for predicting latency.

5.1.2 Minimum Latency Estimation

Using the AS path inference method described in the previous section, and the geographical locations of major PoPs, we have implemented a minimum latency estimator. We have obtained PoP locations of 68 major ASes at the city level from Rocketfuel [29] and translated the city names into longitude/latitude coordinates. For a given pair of nodes, we first infer the AS path between these two nodes, and then connect PoPs along the AS path to infer the packet trajectory between them. More specifically, when we find an AS path (S, X, A, Y, B, Z, D) between nodes s and d, where A and B are ASes whose PoP locations are known, we first construct a graph connecting s to the geographically nearest PoP in AS A, each PoP in A connecting to the nearest PoP in AS B, and d connecting to the geographically nearest PoP in AS B. All PoPs in the same AS are also connected by edges.

We next find the shortest path from s to d in this graph, calculate the geographical distance along this path, and then divide it by the effective speed of light to get the estimated latency between the nodes. The effective speed of light is calculated as $v = \beta \cdot c$, where c is the light speed in vacuum and $\beta < 1$ denotes the light speed factor; usually β needs to be determined using some training set of data. Since we infer a candidate set of AS paths for a given pair of nodes, and Figure 12 empirically shows that the probability of the candidate of rank r being similar to the actual AS path decreases exponentially as r increases, we take an exponentially weighted sum of the contribution of each AS path in the candidate set. Formally, the estimated latency is calculated as $rtt_e = \Sigma_{i=1}^{n} k \cdot e^{-i/\lambda} \cdot r_i$ where r_i is round-trip-time (RTT) estimate from i-th ranked candidate, n is the number of candidates, λ is the damping factor (currently set to 1), and $k = (e^{1/\lambda} - 1)/(1 - e^{-n/\lambda})$

is the normalization factor. Note that the probability of finding the actual AS path by using prediction of ith rank is expressed as $k \cdot e^{-i/\lambda} \cdot$. We have not throughly explored the space of finding the right probability distribution function, but this method works reasonably well.

We have compared the latency estimate produced by our method with the actual latency reported by `ping` between 8192 pairs of 107 PlanetLab sites. Figure 13(a) compares the estimated RTT (rtt_e) and the actual RTT (rtt_a), where the estimated latency is adjusted by the light speed factor $\beta = 0.42$ derived from the actual data sets. The plot shows a strong correlation between rtt_e and rtt_a. Figure 13(b) shows the cumulative distribution of a Relative Delay Penalty (RDP), defined as $(rtt_e - rtt_a)/rtt_a$, and its absolute variant $|rtt_e - rtt_a|/rtt_a$. This plot shows that about 90% of pairs have less than 50% absolute errors, which is comparable to what other approaches such as GNP [20] achieve.

5.2 k-MST on PLUTO Mesh

We now prune the PLUTO mesh to make it more sparse by extracting a *k-minimum spanning tree (k-MST)* [32], using the PLUTO MLE just described. We also show that a k-MST mesh built on top of the PLUTO Mesh contains fewer duplicate physical links than a k-MST mesh built directly from the fully connected virtual topology.

Of course building a k-MST is only one way to construct a sparse mesh, but it has some desirable properties. First, a k-MST mesh contains a number of edges that is linear in the number of nodes, which we expect will enable routing overlays to scale to thousands of nodes. Second, a k-MST can be constructed in a distributed fashion [32]. Third, since our definition of *resilience* is based on the number of MSTs that can be iteratively extracted from a given representative mesh, it is natural to investigate k-MST as an example mesh.

Note that the MST algorithm operates on undirected graphs but our PLUTO-Mesh generates a directed (asymmetric) representative mesh. For this reason, we symmetrize a PLUTO-Mesh such that if the direct edge (u, v) is replaced with the indirect edge (u, w, v), we make sure (v, u) is also replaced with (v, w, u). From now on, we assume that the mesh we discuss is a symmetric directed graph, or simply an undirected graph.

5.2.1 Mesh Sparseness

The benefit from using a k-MST mesh in terms of reduction in the number of virtual links is obvious. Since each MST is disjoint with respect to the others, the total number of virtual links in the k-MST mesh is $k(N-1)$. Using 55 PlanetLab nodes located in the U.S., we have built the PLUTO Mesh and tried to construct a k-MST on top. It turns out that this particular PLUTO mesh only has a *resilience* of 3, meaning it contains only 3-MSTs. However, by running RON on these k-MST (with $k \leq 3$) meshes, we also found that the end-to-end overlay hop count exceeds RON's default maximum of 8. In order to both avoid stressing RON's limits and to examine the quality of denser meshes, we extend the definition of k-MST for ($k > 3$) as follows.

Intuitively, after extracting the maximum number of k-MST out of a given graph, we try to further extract MST's from the residual connected subgraphs to add locally efficient redundant mesh edges. The pseudo code below calculates the *extended* k-MST, G_k, as follows:

1: $R_0 = G$
2: **for** $i = 0, \ldots, k-1$ **do**
3: $(R_i^0, \ldots, R_i^m)$=**subgraphs**(R_i)
4: $F_i = \cup_{j=0 \ldots m}$**MST**$(R_i^j)$
5: $G_{(i+1)} = G_i \cup F_i$
6: $R_{(i+1)} = R_i - F_i$
7: **done**

where **MST**(R) represents the minimum weight spanning tree of graph R, and **subgraphs**(R) returns a vector of connected subgraphs in R. In the i-th iteration of the loop (lines 2-7), we compute the MST F_i of the residual graph R_i from the previous step. If the residual graph R_i is disconnected, we compute the MST of connected sub-graphs R_i^j and union them together into F_i (line 4). The $(i+1)$-th residual graph $R_{(i+1)}$ is defined as a difference between R_i and F_i. (line 6). Note that G_k is a composite of MST's, that is, the union of MST's from each step (line 5). Table 1 shows the number of virtual links in the extended k-MST from our experiment. Note that in all cases, the number of links in our mesh is no greater than the number of links in the standard k-MST.

k	1	2	3	4	5	6	7	PLUTO Mesh
edge count	54	108	162	215	268	318	365	1364
k(N-1)	54	108	162	216	270	324	378	N/A

Table 1: Edge count in k-MST

For different k-MST meshes, Figure 14(a) compares the three types of traffic statistics that we have examined for PLUTO Mesh in Section 4.3: ping, route-updates, and total (including ping, route updates, and route-updates forwarding). Each plot shows the ratio of total traffic generated on the k-MST meshes to the total traffic on the fully-connected mesh. As the plot shows, the amount of traffic decreases in proportion to k, that is, to the number of virtual links in the mesh.

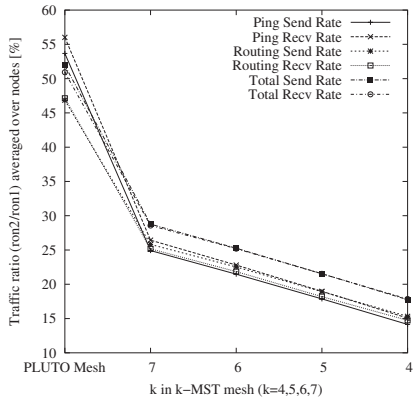
5.2.2 Mesh Quality

Using the same set of 55 PlanetLab nodes as in the previous experiment, we have compared the optimal latency and bandwidth of the fully connected mesh and the k-MST meshes built on top of the PLUTO Mesh. Each experiment runs about 60 minutes. For each k-MST mesh ($k = 4, \ldots, 7$), we run two instances of RON at the same time using the same set of nodes, one on the fully-connected mesh (`ron1`) and the other on the k-MST mesh (`ron2`).

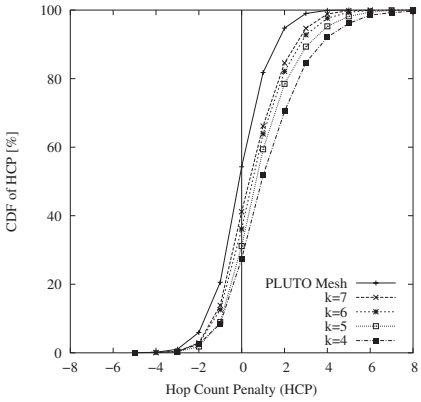
Figure 14(b) shows the cumulative distribution of the Hop Count Penalty (HCP). HCP defines the difference in hop count between the latency optimized paths chosen by `ron2` (on the k-MST mesh or PLUTO mesh) and `ron1` (on the complete mesh). The plot shows that `ron2` on the PLUTO mesh incurs essentially no HCP relative to `ron1`; half the paths have more hops on `ron1` and half have more on `ron2`. However, the k-MST meshes incur more HCP as k decreases. For example, only 5% of paths on the PLUTO mesh add more than 2 extra hops compared to paths in the complete mesh, while this is true for about 30% of paths on the 4-MST mesh.

Figure 15(a) shows the cumulative distribution of the Relative Delay Penalty (RDP) for end-to-end latency. RDP is calculated as a ratio of the latency difference (`ron2 - ron1`) to the reference latency (`ron1`). Not surprisingly, we see that we sacrifice some performance as k decreases: while 90% of paths in the PLUTO mesh have an RDP of less than 50%, only 80% of paths in the 4-MST mesh achieve an RDP of less than 50%. We also observe that about 3% of the paths on sparser graphs have 50% better optimal paths than on the complete graph. We conjecture this is caused by the same reasons described in Section 4.4.

Likewise, Figure 15(b) shows the cumulative distribution of the Relative Bandwidth Penalty (RBP) of the bandwidth optimal paths.

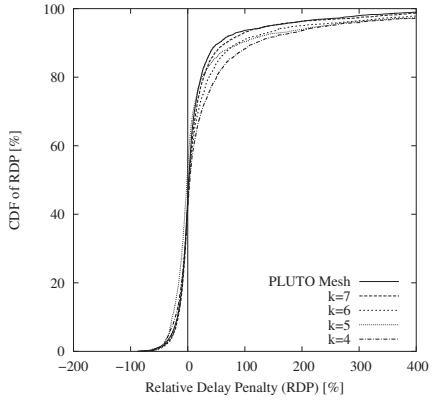


(a) Ratio of traffic generated on k-MST on PLUTO mesh to traffic on complete mesh

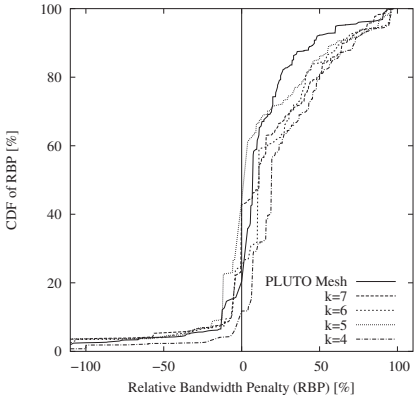


(b) Hop Count Penalty (HCP) (ron2-ron1)

Figure 14: Traffic and overlay-hop-count comparison between complete mesh and k-MST mesh on PLUTO Mesh



(a) Relative Delay Penalty (RDP)



(b) Relative Bandwidth Penalty (RBP)

Figure 15: Performance comparison between complete mesh and k-MST mesh on PLUTO Mesh

RBP is calculated as a ratio of the bandwidth difference (`ron1 -ron2`) to the reference bandwidth (`ron1`), and represents the bandwidth sacrificed when we reduce the number of virtual links. The plot shows that 90% of paths on the PLUTO mesh and 80% of paths on the 4-MST mesh retain more than 50% of the possible bandwidth.

5.2.3 Mesh Redundancy

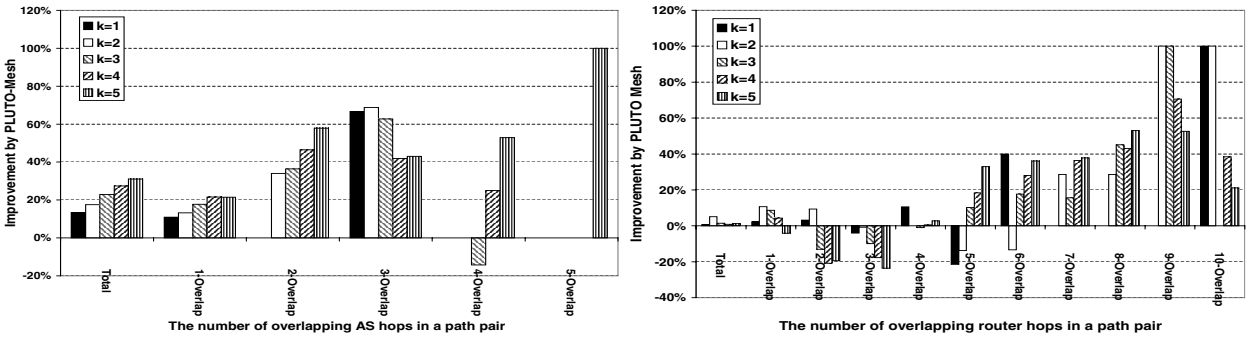
We now examine the redundancy metrics defined in Section 4.1. Obviously, the *resilience* metric of a k-MST mesh is k, since by its definition, it includes k edge-disjoint MSTs in the graph. To evaluate the *duplicates* metric, we compare two k-MST meshes among 60 nodes, one constructed on top of the fully connected virtual topology (`kmst1`), and the other on top of the PLUTO Mesh (`kmst2`) using PLUTO MLE to estimate latency. For each mesh, we use traceroute along all virtual links in the mesh to obtain router and AS sequences. Figure 16 shows the *duplicates* metric at both AS granularity and router granularity.

Figure 16(a) shows the improvements in the *duplicates* metric at the AS-level ((`kmst1-kmst2`)/`kmst1`). Recall that the AS level *duplicates* metric counts the pairs of virtual links that share one or more AS hops. As bars in the "Total" category shows, the k-MST mesh built on PLUTO-Mesh contains 13% (k=1) to 31% (k=5) fewer virtual edge pairs that share any AS hops—that is, it makes this many virtual edge pairs disjoint. Further breaking down the result according to the number of overlapping ASes in the non-disjoint virtual edge pairs, we see that the benefits of PLUTO-Mesh are distributed across all overlap counts.

Finally, Figure 16(b) shows the improvements at the router level. The "Total" category shows that the increase in disjoint router-level paths is less than 5%, but the break-down by the number of overlapping router hops shows that PLUTO-Mesh generally reduces the *duplicates* metric for longer overlapping paths, while it increases the *duplicates* metric for shorter overlaps. This result implies PLUTO-Mesh turns virtual link pairs sharing longer router hops into those sharing shorter hops, and thus improves the disjointedness of the resulting mesh. Note that PLUTO-Mesh achieves this improvement only using AS-level topology and geographical information.

5.3 Summary

This section explores the trade-off between α, representing the number of virtual links in a sparse routing mesh, and the mesh's quality. One reason we have chosen a k-MST mesh as an example sparse routing mesh is because, with such a mesh, the edge reduction parameter α is inversely proportional to N. We have shown



(a) AS level

(b) Router level

Figure 16: The improvements in the duplicates metric

that using PLUTO MLE we can construct a k-MST mesh on top of the PLUTO Mesh without actively probing the Internet. The quality of the k-MST mesh, measured in both latency and throughput, degrades gracefully with k. As a result, aggressive reductions in α can be achieved while still producing good results for routing overlays.

We have also shown that a key benefit of building a sparse routing mesh on top of the PLUTO Mesh is to reduce the number of paths that have duplicate physical segments. Naïvely constructing a k-MST mesh directly on top of a fully connected mesh (i.e., the Internet) leads to a significant number of shared physical links among the seemingly disjoint paths. A k-MST mesh built using the PLUTO Mesh as a middle layer has fewer duplicate physical segments in the underlying network, thus improving the resilience of the resulting routing mesh at minimal cost.

Finally, we recognize that an overlay with a thousand nodes is likely to stress route dissemination algorithms. It is easy to imagine overlays adopting less aggressive dissemination strategies at large scale since their failure modes are different than when used in traditional physical networks (e.g., the overlay can always fall back to use the default Internet path between a pair of nodes). How to scale routing protocols to work on a large overlay is a subject of future work.

6. RELATED WORK

Overlays commonly maintain subgraphs of the complete graph as a routing mesh. As opposed to RON [4]'s complete routing mesh scheme, several single-source application-level multicast overlays (e.g., ESM [7, 8], YOID [23], Overcast [14], Bullet [15], and SplitStream [6]) define more sparse routing meshes and sometimes build multicast trees on top of them. Most related to our approach may be Interleaved Spanning Tree [32], which proposed a distributed algorithm to define sparser mesh for generic overlay networks. However, all of these mesh building strategies treat the Internet as a black box, relying on expensive performance measurements before constructing a routing mesh; none capture the idea of duplicate segments in the underlying network. To the best of our knowledge, our work is the first attempt to define an infrastructure for constructing topology-aware representative meshes in a cost-effective manner.

Several research efforts have tried to inexpensively predict the end-to-end latency for a given pair of nodes. IDMaps [12] explored the feasibility of a public infrastructure to provide end-to-end distance information. GNP [20], Lighthouses [25], ICS [17], Virtual Landmarks [31], and PIC [9] proposed a coordinate-based strategy to predict distance by treating the Internet as a high-dimensional geometric space. Our PLUTO MLE is novel in that it uses only information that is passive (BGP tables) and fairly static (geographical location of PoPs of major ASes) to achieve reasonably good latency estimation.

7. CONCLUSIONS

This paper describes a distributed service that constructs a topologically-representative mesh using only passive measurements and static topology information. By eliminating mesh edges that are not likely to be selected by a high-level routing overlay, we are able to reduce the impact of unnecessarily probing the network to find good routes, and in the process, improve the scalability of routing overlays. Experiments show that when redundant edges are conservatively removed, route selection does not suffer but the overhead of probing the network and disseminating routing information is reduced by a factor of two. Additional analysis quantifies the impact on route selection of more aggressively removing mesh edges, documenting the cost/benefit trade-off that is intrinsic to routing.

One of the main insights of this work is that while treating the underlying Internet as a black-box is appealing in many ways, information about the Internet's topology is readily available, either for free or at very low cost, and that using this information can have a dramatic effect on building a routing overlay that is both cost-conscious (scalable) and effective.

8. REFERENCES

[1] Fixed Orbit. http://www.fixedorbit.com/.

[2] Packet Clearing House. http://www.pch.net.

[3] Route Views Project. http://antc.uoregon.edu/route-views/.

[4] D. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Resilient Overlay Networks. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP)*, pages 131–145, Chateau Lake Louise, Banff, Alberta, Canada, October 2001.

[5] S. Banerjee, T. G. Griffin, and M. Pias. The Interdomain Connectivity of PlanetLab Nodes. In *Proceedings of the Passive and Active Measurement Workshop (PAM2004)*, Antibes Juan-les-Pins, France, April 2004.

[6] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: High-bandwidth Multicast In Cooperative Environments. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 298–313. ACM Press, 2003.

[7] Y.-H. Chu, S. G. Rao, S. Seshan, and H. Zhang. Enabling Conferencing Applications on the Internet using an Overlay

Multicast Architecture. In *Proceedings of the ACM SIGCOMM Conference*, pages 1–12, August 2001.

[8] Y.-H. Chu, S. G. Rao, and H. Zhang. A Case For End System Multicast. In *Proceedings of the ACM SIGCOMM Conference*, pages 1–12, June 2000.

[9] M. Costa, M. Castro, A. Rowstron, and P. Key. PIC: Practical Internet Coordinates for Distance Estimation. In *In Proceedings of the 24th International Conference on Distributed Computing Systems 2004*, March 2004.

[10] Dexter C. Kozen. *The Design and Analysis of Algorithms*. Springer-Verlag, 1992.

[11] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based Congestion Control for Unicast Applications. In *Proceedings of the ACM SIGCOMM*, pages 43–56, 2000.

[12] P. Francis, S. Jamin, C. Jin, Y. Jin, V. Paxson, D. Raz, Y. Shavitt, and L. Zhang. IDMaps: A Global Internet Host Distance Estimation Service. In *Proceedings of the IEEE INFOCOM Conference*, 1999.

[13] L. Gao. On Inferring Autonomous System Relationships in the Internet. In *Proceedings of IEEE Global Internet Symposium*, November 2000.

[14] J. Jannotti, D. Gifford, K. Johnson, M. Kaashoek, and J. Jr. Overcast: Reliable Multicasting with an Overlay Network. In *Proceedings of the Fourth USENIX Symposium on Operating System Design and Implementation (OSDI)*, October 2000.

[15] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh. In *Proceedings of the 19th ACM symposium on Operating Systems Principles (SOSP)*, pages 282–297. ACM Press, 2003.

[16] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian. Delayed Internet Routing Convergence. In *Proceedings of the ACM SIGCOMM Conference*, pages 175–187, 2000.

[17] H. Lim, J. C. Hou, and C.-H. Choi. Constructing Internet Coordinate System Based on Delay Measurement. In *Proceedings of the 2003 ACM SIGCOMM conference on Internet measurement*, pages 129–142. ACM Press, 2003.

[18] Z. M. Mao, J. Rexford, J. Wang, and R. Katz. Towards an Accurate AS-Level Traceroute Tool. In *Proceedings of the ACM SIGCOMM 2003 Conference*, August 2003.

[19] A. Nakao, L. Peterson, and A. Bavier. A Routing Underlay for Overlay Networks. In *Proceedings of the ACM SIGCOMM 2003 Conference*, August 2003.

[20] T. S. E. Ng and H. Zhang. Predicting Internet Network Distance with Coordinates-Based Approaches. In *Proceedings of the IEEE INFOCOM Conference*, June 2002.

[21] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: A simple model and its empirical validation. In *ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 303–314, Vancouver, CA, 1998.

[22] V. N. Padmanabhan and L. Subramanian. An investigation of geographic mapping techniques for internet hosts. In *Proceedings of the ACM SIGCOMM 2001 Conference*, 2001.

[23] S. R. Paul Francis. Your Own Internet Distribution, 2001. http://www.aciri.org/yoid/.

[24] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A Blueprint for Introducing Disruptive Technology into the Internet. In *Proceedings of the HotNets-I*, 2002.

[25] M. Pias, J. Crowcroft, S. Wilbur, T. Harris, and S. Bhatti. Lighthouses for Scalable Distributed Location. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems*, February 2003.

[26] Y. Rekhter and T. Li. A Border Gateway Protocol 4, March 1995. RFC 1771.

[27] S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat, G. Voelker, and J. Zahorjan. Detour: A Case for Informed Internet Routing and Transport. *IEEE Micro*, 19(1):50–59, January 1999.

[28] J. L. Sobrinho. Algebra and Algorithms for QoS Path Computation and Hop-by-Hop Routing in the Internet. *IEEE/ACM Transactions on Networking*, 10(4):541–550, 2002.

[29] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP Topologies with Rocketfuel. In *Proceedings of the ACM SIGCOMM Conference*, pages 133–145, August 2002.

[30] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet Indirection Infrastructure. In *Proceedings of the ACM SIGCOMM Conference*, pages 73–85, August 2002.

[31] L. Tang and M. Crovella. Virtual Landmarks for the Internet. In *Proceedings of the 2003 ACM SIGCOMM conference on Internet measurement*, pages 143–152. ACM Press, 2003.

[32] A. Young, J. Chen, Z. Ma, A. Krishnamurthy, L. Peterson, and R. Y. Wang. Overlay Mesh Construction Using Interleaved Spanning Trees. In *Proceedings of the IEEE INFOCOM Conference*, March 2004.