Scalable Routing Overlay Networks

Akihiro Nakao and Larry Peterson
Department of Computer Science
Princeton University
{*nakao,llp*}*@cs.princeton.edu*

Abstract

Routing overlays have become a viable approach to working around slow BGP convergence and sub-optimal path selection. A common sub-component of a routing overlay is a *routing mesh*: the route-selection algorithm considers only *virtual links* in the mesh rather than all N^2 paths connecting an N-node overlay, thereby reducing routing overhead and improving the scalability of the overlay. This paper proposes and evaluates an 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, thereby improving scalability. Additional analysis quantifies the impact on route selection of more aggressively removing mesh edges, documenting the cost/benefit tradeoff that is intrinsic to routing.

1 Introduction

Routing overlays have recently been proposed as an approach to improving the robustness and performance of packet delivery over the Internet [31, 6, 9, 10, 16, 17, 8]. To improve robustness, overlays rapidly find an alternative path when the current BGP-provided route fails [30, 18]. To improve performance, overlays select the best of multiple alternative paths. Routing overlays accomplish these two 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 [6], 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 cost will go down and scalability will improve, but without harming the routing overlay's ability to select good routes.

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 [9, 10, 16, 8] 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.

This paper puts forward another approach, which is to exploit static topology information about the Internet to build a routing mesh that is representative of the underlying physical network [22]. The resulting *representative mesh* attempts to eliminate virtual links (mesh edges) that have duplicate physical segments in the underlying network. We can define a representative mesh 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 introduce 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; it does not itself add to monitoring overhead. Experiments on PlanetLab [28] 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 PLUTO Architecture

This section describes PLUTO (PlanetLab Underlay Topology services for Overlay Networks), which defines a two-layer routing hierarchy [22]. 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.

2.1 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 [5] 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.

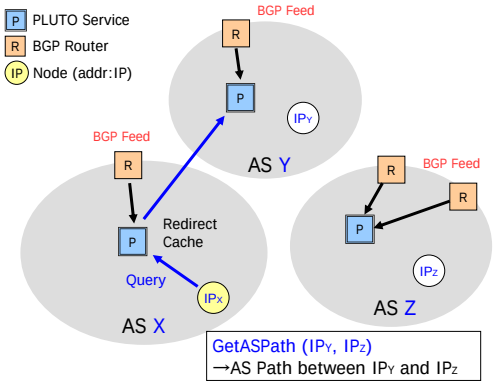


Figure 1: PLUTO topology discovery sub-services

For example, suppose the overlay node 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 path originating from AS Y.

Note that while PLUTO assumes that a BGP feed is available in every AS that hosts a node, actually achieving 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 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.

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. Another option is to implement a global mapping from IP addresses to geographical locations; we discuss this option in Section 5.3. 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 [20] 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 [4]: 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)

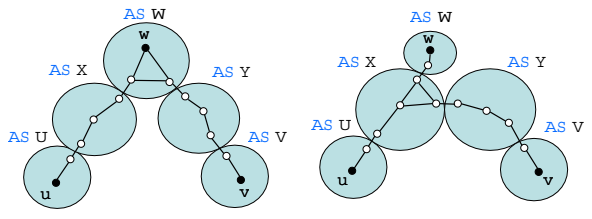
returns point-of-presence (PoP) information PoPInfo for the given AS number ASN. This information includes the list of IP addresses and geographical locations of PoPs of the AS of ASN. PLUTO currently registers PoP geographical locations of 68 large ASes obtained from Rocketfuel[33]. This information is expected to be updated very infrequently.

2.2 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 more a part of the routing problem, than the mesh construction problem.

2.2.1 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



(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.

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, however, 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, although the resulting mesh may not be as sparse as that produced by a centralized algorithm.

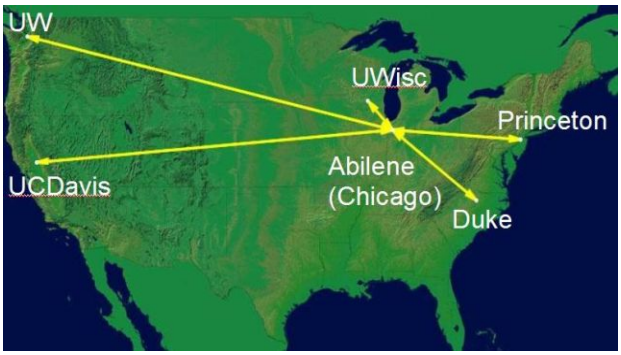
Our algorithm prunes an edge from the local node u to remote node v if the AS path from u to v includes AS W, such that there is a node $w \in N$ that is located within AS W. This scenario is illustrated in Figure 2(a). Although each node u runs this algorithm locally, it will not cause a resulting mesh 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 example, we estimate latency from u to w and from u to v by latency estimator discussed in 4.1, 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).

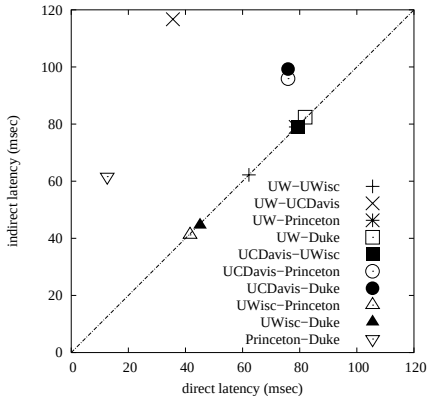
2.2.2 Geo-Based Pruning

Although our algorithm is very straight-forward, AS-level information may be too coarse-grain to produce a satisfactory result, especially when we have large ASes spanning a continent. 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. We need to add extra steps to complete our mesh construction algorithm, resolving the two problems that these simplifying assumptions hid.

First, in Figure 2(a), if AS W is a cross-country AS, default path (u, v) and its indirection path (u, w, v) may not have many duplicate route hops, hence, may not bear similar network property to (u, v). Second, our algorithm has not clarified the case where we have multiple nodes in the same AS. One simple solution is to let one of the nodes as a representative of the nodes in its AS. However, since the AS in



(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)

question can cover a large area, we need a more generic strategy. If we have multiple nodes in a large AS, we need to discriminate these multiple nodes further.

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 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) shows 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.

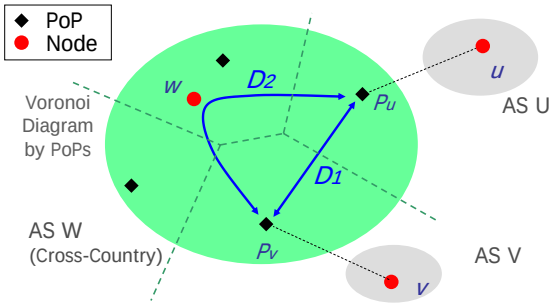
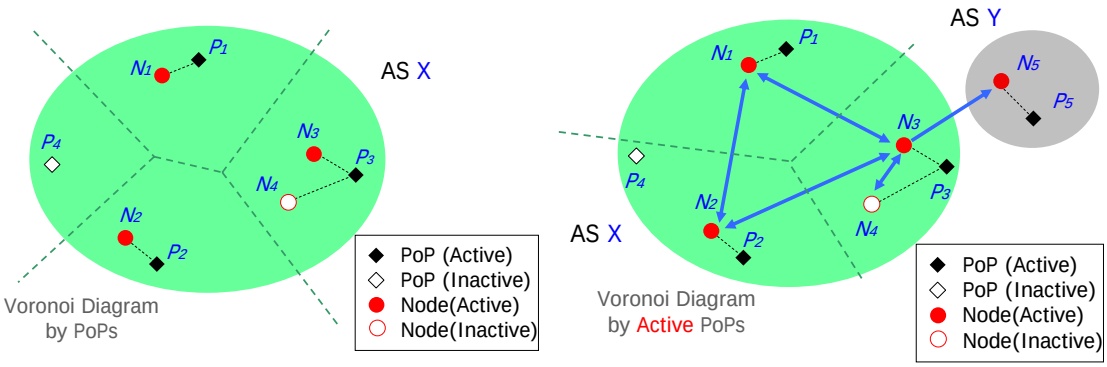


Figure 4: Mesh algorithm 1 (preserving good direct links)

Figure 4 illustrates how our discriminator works to preserve good direct links. Suppose we consider removing direct link (u, v) since we believe indirect link (u, w, v) has similar network properties. In order to assess 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. We now use the first assumption that says regional ASes are connected to the nearest PoP of the cross-country AS they subscribe to. 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 use the second assumption that says 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 the



(a) Step1: classifying nodes (b) Step2: construct virtual edges

Figure 5: Mesh algorithm 2 (discriminating multiple nodes)

Section 4.1.2, 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 works for 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, 4 pops P_1 to P_4 in the AS X. First, we subdivide the AS into voronoi cells by PoPs. PoP P_1 and P_2 each have 1 node, P_3 has 2 nodes, and P_4 has no node. 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.

3 Evaluation

This section reports on experiments designed to evaluate how much redundant traffic we can reduce with the mesh constructed by PLUTO, without sacrificing performance and resilience. Designing a mesh to support a routing overlay like RON [6] is more challenging than supporting a single-source multicast overlay like [9, 10, 16, 8], since it needs to optimize all possible pairwise connections and an unnecessary virtual link for one node could be a crucial one for the other. Also, RON is so aggressive in monitoring link behavior that one can view it as approximating the optimal route selection strategy. Therefore, we use RON as an example of a routing overlay that runs on top of the mesh we construct. The experiment shows that our mesh successfully removes redundant virtual links and suppresses unnecessary traffic without degrading the performance of the routing overlay.

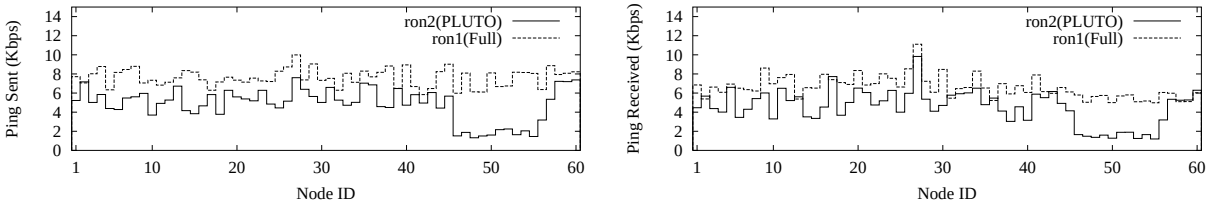
3.1 Redundancy Metrics

Before presenting the results, we note that there are two types of redundant virtual links. Undesirable virtual links contain duplicate underlying physical links, while desirable virtual links are (maximally) edge-disjoint in terms of underlying physical links. For a given representative mesh, we define two metrics: *duplicate* indicates how many duplicate links we have in the underlying topology, and *resilience* corresponds to how many disjoint minimum spanning trees (MSTs) we iteratively "extract" from the mesh. We further define two variants of the *duplicate* metric: one counts the number of duplicate router hops, and the other counts the number of duplicate AS hops. For any two meshes with the same number of virtual links, we prefer the one with the smaller *duplicate* metric.

3.2 Experimental Setup

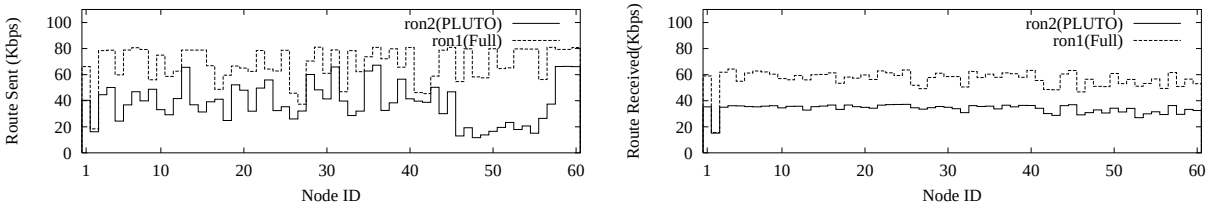
To evaluate the mesh constructed by PLUTO, we run one instance of RON (`ron1`) on a fully connected mesh, and *at the same time on the same set of nodes*, a second instance of RON (`ron2`) on top of PLUTO-Mesh. We run both instances of RON for 40 minutes across 60 PlanetLab nodes geographically distributed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

To conduct this experiment, we modified RON slightly. The original implementation calculates only single-hop indirection routes as alternate routes (although it exchanges link-state information among participating nodes) based on the observation that single hop indirection gives us the best alternate



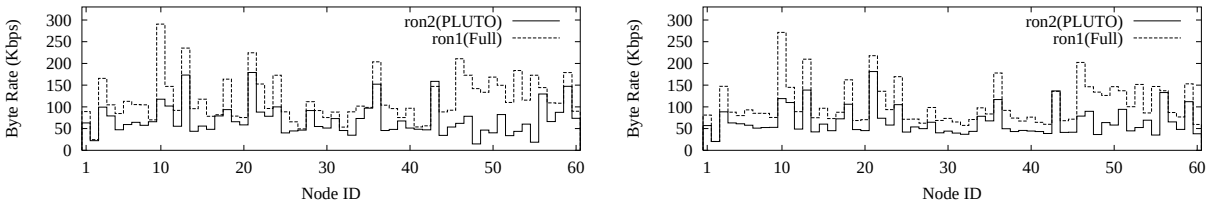
(a) Ping Sent (b) Ping Received

Figure 6: Ping traffic comparison (full-mesh/sparse-mesh)



(a) Route Sent (b) Route Received

Figure 7: Routing update traffic comparison (full-mesh/sparse-mesh)



(a) Byte Sent (b) Byte Received

Figure 8: Total traffic comparison (full-mesh/sparse-mesh)

path most of the time. Our modified RON calculates the optimal multi-hop indirection route, since in PLUTO-Mesh, each node has a limited number of virtual neighbors, meaning that we cannot strictly rely on single hop indirection.

3.3 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 55.3% reduction in the number of virtual links 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 total traffic does not represent only the sum of ping and route dissemination traffic, but also includes the traffic for forwarding routes, since we use our routing mechanism itself to disseminate routes in order to get over disconnectivity between the Internet1 and Internet2. According to these graphs, we reduce about 50% on average (over nodes) of the traffic in `ron2` comparing to `ron1`. Some nodes (especially those on crossroad) have significantly less traffic since these nodes do not have many virtual links.

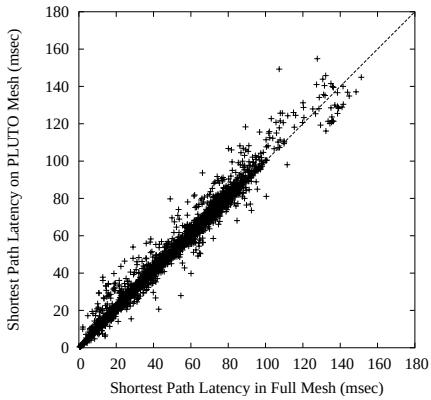
3.4 Mesh Quality

Next, we examine how much performance we have sacrificed by reducing the number of virtual links. Taking a snapshot of routing tables of RON, we have examined difference in optimal latency and bandwidth between every pair of nodes on fully connected topology and on our representative mesh. The bandwidth of each virtual link is calculated using TCP congestion control equations [25, 13] 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 modified Dijkstra algorithm to calculate bandwidth-optimized paths [32, 12].

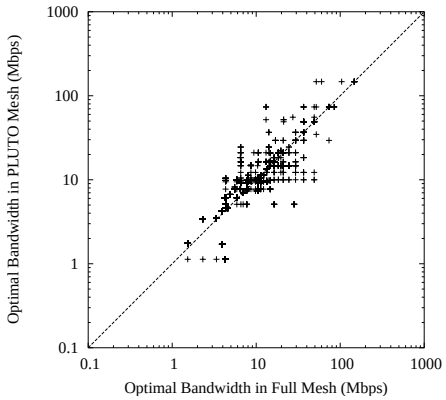
As Figures 9(a) and 9(b) show, the performance degradation is minimal. Note that sometimes, PLUTO-Mesh can achieve better performance than the complete mesh. We believe this is partly because there exists some fluctuation in route convergence, and also because of the high-bandwidth usage incurred from running two instances of RON, a denser mesh has a higher chances of losing route updates.

3.5 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 implies 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 *duplicate* metric between the complete mesh and PLUTO-Mesh since the number of virtual



(a) Every pair-wise shortest path latency comparison (Full-Mesh/ PLUTO-Mesh)



(b) Every pair-wise optimal bandwidth comparison (Full-Mesh/PLUTO-Mesh); calculated from loss and RTT by TCP equation

Figure 9: Mesh quality comparison (latency and bandwidth)

links in both graphs are significantly different. We will look into the validity of PLUTO-Mesh in this regard in the next section.

4 Cost/Benefit Trade-off

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 further information 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 arbitrary pair of nodes using only passive and infrequently updated information.

Using PLUTO MLE, we then construct more sparse meshes from the PLUTO Mesh, and evaluate the sparseness, quality and redundancy of the resulting meshes. 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 even though we could build sparse mesh starting from fully connected virtual topology, building sparse meshes on top of the PLUTO Mesh has even better properties.

4.1 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 [26] has shown that estimating latency using only geographical locations of end-points is not very accurate, especially when the end points are far away. We believe that this 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 ASes it traverses. The basic idea is to first infer the AS Path between arbitrary pair of end nodes, and then to add PoP-to-PoP geographical distance along the AS path to infer the end-to-end latency. If we obtained BGP feed at every single AS in the world, we would not have to perform the first part of the strategy.

However, even when we had all the BGP information, it would be useful to have a service that can infer arbitrary AS path using only a limited number of BGP feeds. In this section, we propose a method to infer AS paths using existing BGP feed from various vantage points.

4.1.1 AS Path Inference

We discuss inferring AS paths using only a handful of BGP feeds. A simple method one might come up with is to construct a connectivity graph (via the GetASGraph operation), in which nodes are ASes and edges are BGP neighbors, and then to infer an AS path by calculating the shortest path on this connectivity graph. However, there is a shortcoming in this approach. Since the connectivity graph represents only *possible* communication channels, the shortest AS hop count path on this graph does not necessarily reflect the BGP policy even though the shortest AS hop count path would be selected as the best one without the BGP artifacts.

Our previous work [22] shows that only 50% of 5500 examined paths are predicted by this shortest path method. One must 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. Our matching scheme is to check if the actual AS path is in the set of multiple shortest paths calculated from the BGP routes.

In our experiment, we sometimes encounter a pair of nodes having over several hundreds of shortest paths of the same length. If we had to infer the actual path using this method, we would have to guess which is the actual path selected

from many candidates. Considering a single policy factor AS peer relationship using the method in [15] and attempting to exclude *illegal* prediction (such as a customer AS is carrying traffic for a provider AS), the prediction accuracy is only slightly better than by the naive approach. We believe the low accuracy of this method results from the fact that policies are so deeply embedded in BGP routing.

For this reason, we take a different approach to inferring AS paths. Our new approach is based on the following two observations and tries to capture the idea of BGP policies as much as possible. First, about 70% of the AS paths observed in RouteViews BGP tables are symmetric. Second, although we do not have any quantitative examination of this, we assume that AS paths are *transitive*, meaning that if we observe an AS path like (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.

Suppose we are trying to infer an AS path between a source node s and a destination node d, using k vantage points $v_i(i = 0...k-1)$ where we can access BGP tables. For each vantage point v, we first get the AS path P_1 from v to s, $P_1 = (V, X, M, Y, S)$ and the AS path P_2 from v to d, $P_2 = (V, X, M, Z, D)$, where s, d and v reside in AS S, AS D and AS V, respectively, and AS paths P_1 and P_2 share the AS hops till they reach AS M. After finding AS M, we infer the AS path from s to d to be $P_v = (S, Y, M, Z, D)$ as shown in Figure 10. We repeat this process for all of k vantage points $v_i(i = 0...k-1)$ to obtain AS paths $P_{v_i}(i = 0...k-1)$. We then rank them into a set $P[r](r = 0...l \leq k-1)$ in the increasing order of AS hop counts, consolidating the same paths.

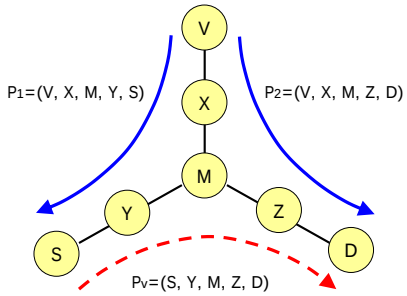


Figure 10: AS Path Inference

To evaluate this strategy, we have examined BGP tables from 100 vantage points from RouteViews and PlanetLab BGP feeds. For each BGP table of AS X, we use only the vantage points that do not belong to AS X and infer the AS path from AS X to every single network prefix (there are typically 150,000 such prefixes). Figure 11 shows that how many of 150,000 AS paths in each BGP table can be successfully inferred (y-axis) when we use the first r rank of path inference $P[r]$ (x-axis). As shown in Figure 11(a), for most AS paths, we can achieve 60% accuracy within rank 5, except for a few cases that fall behind less than 50%. We have found that these low-accuracy curves belong to ASes located in Europe or Asia, where we do not have concentration of vantage points. However, for the most BGP tables from the vantage points located in U.S., we can achieve higher accuracy of inferring AS paths.

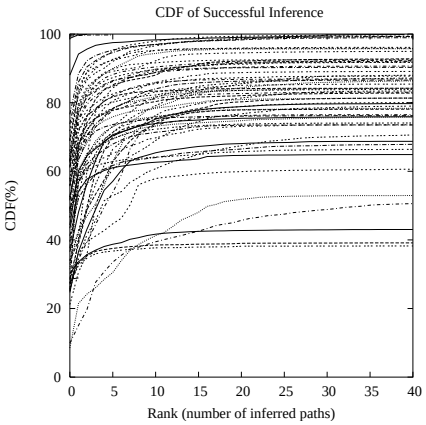
In addition, we have modified our matching method from *perfect matching*, where a predicted AS path must be hop-by-hop identical to the actual path, into *ambiguous matching*, where only one AS in the inferred path is different from the actual path. Figure 11(b) shows that this ambiguous matching achieves over 90% within rank 5 for most vantage points. The reason we conducted this experiment is because we expect that the one AS difference is not very important for our strategy of estimating latency.

4.1.2 Minimum Latency Estimation

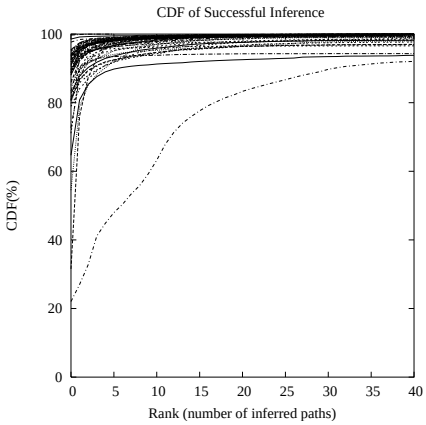
Using AS path inference and the geographical location of major PoP locations, we have implemented a minimum latency estimator. We have obtained PoP locations of 68 major ASes at the city level from Rocketfuel [33] and translated the city name into longitude/latitude coordinate. For a given pair of nodes, 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 such that s connects an edge to the geographically nearest PoP in AS A, then each A's PoP connects an edge to the nearest PoP in AS B, then B's nearest PoP to d connects an edge to d, and all PoPs in the same AS interconnect their edges themselves.

We next find the shortest path from s to d in this graph and calculate the geographical distance along the shortest path and divide it by the effective light speed $v = \alpha \cdot c$ to get the estimated latency between the nodes, where c is the light speed in vacuum and $\alpha < 1$ denotes the light speed factor. Usually α needs to be determined using some training set of data. However, as long as we use the estimated latency for comparison purpose as in getting a set of k-nearest neighbors for a given node, the absolute value of α does not matter. Since we infer several multiple AS paths ranked by the number of AS hops for a given pair of nodes, and Figure 11 empirically shows that the probability of $P[r]$ being an actual AS Path decreases exponentially as rank r, we take exponentially weighted sum of the contribution from each AS path. Thus, the estimated latency is calculated as $rtt_e = \Sigma_{i=0}^{n} k \cdot e^{-(i+1)/\lambda} \cdot r_i$ where r_i is round-trip-time (RTT) estimate from i-th rank path $P[i]$, n is the maximum number of ranks, λ is damping factor (currently set to 1), and $k = (e^{1/\lambda} - 1)/(1 - e^{-(n+1)/\lambda})$ is normalization factor.

Since we have geographical coordinates of all the PlanetLab sites, we have examined ping RTT and estimated RTT

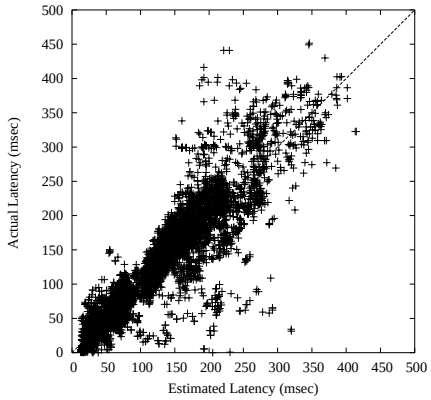


(a) Perfect Matching (6 curves from the bottom are from the vantage points at 195.66.232.239(UK), 195.66.232.254(UK), 203.194.0.12(AU), 194.85.4.249(RU), 141.142.12.1(US;IL) and 213.140.32.144(NL))

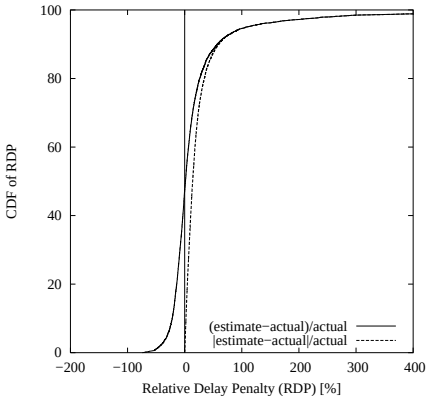


(b) Ambiguous Matching (The bottom curve is from 194.85.4.249 (RU))

Figure 11: AS Path Inference



(a) Comparison between estimated latency rtt_e (x-axis) and actual latency rtt_a (y-axis)



(b) CDF of Relative Delay Penalty (RDP) $(rtt_e - rtt_a)/rtt_a$ and its absolute variant $|rtt_e - rtt_a|/rtt_a$

Figure 12: Minimum Latency Estimation

calculated by our estimator between 8192 pairs of 107 PlanetLab sites. Figure 12(a) compares the estimated rtt_e and the actual RTT rtt_a where estimated latency is adjusted by the light speed factor $\alpha = 0.42$ using a linear fitting to best fit with the actual data sets. The plot shows that there exists a strong correlation between rtt_e and rtt_a. Figure 12(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. There are points on the plot that are off the $Y = X$ line (perfect estimate), indicating when our AS path inference or latency estimation algorithm gives faulty results.

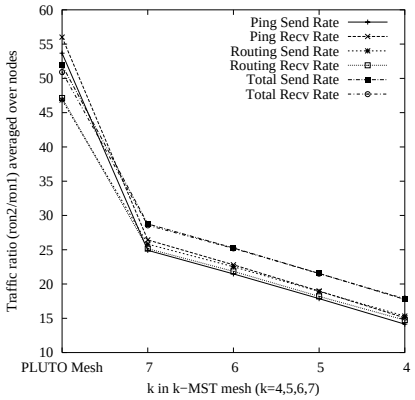
4.2 k-MST on PLUTO Mesh

We now prune our representative mesh to make it more sparse by extracting *k-minimum spanning tree (k-MST)* [35], using the PLUTO MLE just described. We also show that even though we could build a k-MST mesh starting from fully connected virtual topology, one constructed on top of the PLUTO-Mesh has better properties.

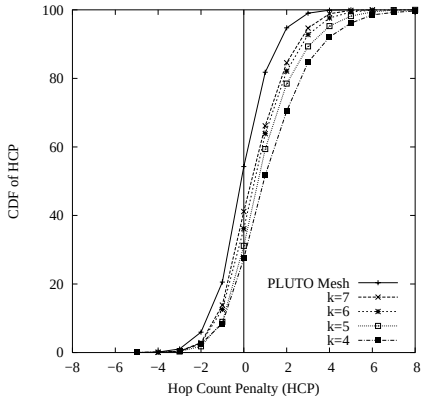
Note that although we pick k-MST as an example method for constructing a more sparse mesh, there could be many other alternatives. However, since our definition of *resilience* is based on the number of MSTs that can be iteratively extracted from a given representative mesh, it is natural to investigate k-MST as an example mesh. Such k-MST can be constructed in a distributed fashion [35]. Also note that MST is well-defined for undirected graphs and 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.

4.2.1 Mesh Sparseness

The benefit from using 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 vir-

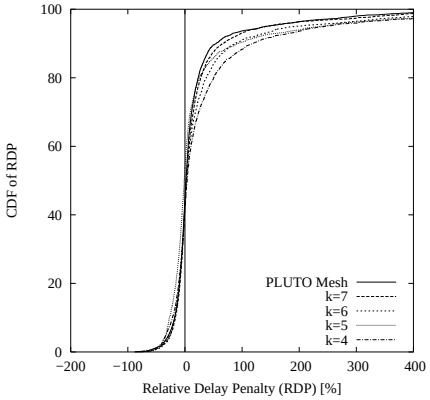


(a) Ratio of traffic generated on k-MST on PLUTO mesh to traffic on complete mesh

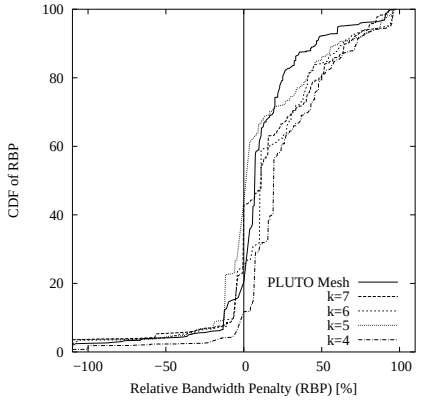


(b) Hop Count Penalty (ron2-ron1)

Figure 13: 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 14: Performance comparison between complete mesh and k-MST mesh on PLUTO Mesh

tual links in the k-MST mesh is $k(N-1)$. Using 55 PlanetLab nodes located in U.S., we have built PLUTO-Mesh and tried to construct k-MST on top. It turns out that this particular PLUTO mesh happens to include maximally 3-MSTs (*resilience*=3). However, by running RON on these k-MST ($k <= 3$) meshes, we also found that end-to-end overlay hop count becomes too large, exceeding RON's default maximum of 8. In order to avoid such unreasonable setting and to examine the quality of denser meshes, we extend the definition of k-MST for ($k > k^* = 3$) 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 *extended* k-MST, G_k, as follows:

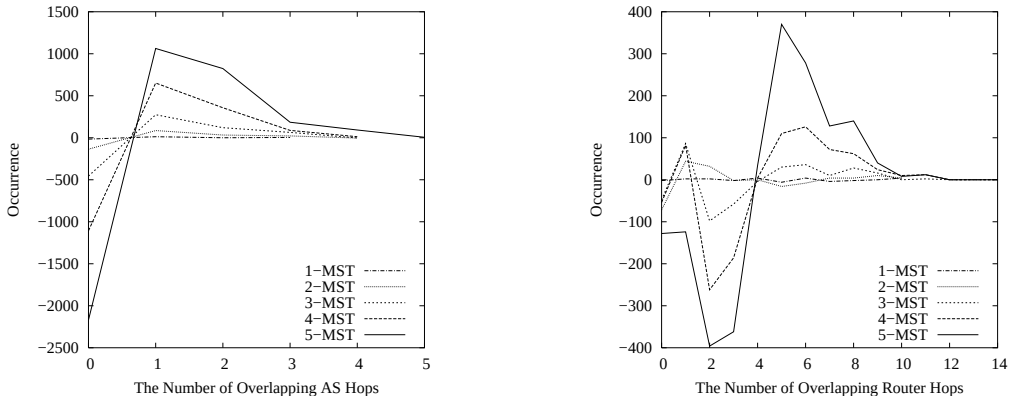
1: $R_0 = G$
2: **for** $i = 0, \ldots, k-1$ **do**
3: $(R_i^0, \ldots, R_i^m)$=**subgraphs**(R_i)
4: $F_i = \cup_{j=0 \ldots m}$**MST**$(R_i^j)$
5: $G_{(i+1)} = G_i \cup F_i$
6: $R_{(i+1)} = R_i - F_i$
7: **done**

where **MST(**R**)** represents the minimum weight spanning tree of graph R, and **subgraphs(**R**)** returns a vector of connected sub-graphs 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.

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

Figure 13(a) compares three types of traffic statistics that we have examined for PLUTO Mesh in Section3.3: ping, route-updates, and total (including ping, route updates, and route-updates forwarding), with different k-MST meshes. Each plot shows the ratio of traffic generated on the k-MST



(a) AS Level Redundancy (b) Router Level Redundancy

Figure 15: Mesh redundancy (occurrence of duplicate hops in every pair of paths)

meshes averaged across overlay nodes, to the traffic on the fully-connected mesh. As the plot shows, the amount of traffic decreases in proportional to k, hence, the number of virtual links in the mesh.

4.2.2 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 k-MST mesh (`ron2`).

Figure 14(a) shows cumulative distribution of the Relative Delay Penalty (RDP) of end-to-end latency. RDP is calculated as a ratio of the latency difference (`ron2 - ron1`) to the reference latency (`ron1`). While 90% of paths on PLUTO mesh have RDP of less than 50%, only 80% of paths on 4-MST mesh achieve an RDP of less than 50%. We see that we sacrifice more performance as k decreases. We also observe that about 3% of paths on sparser graphs have 50% better optimal paths than on complete graph. We conjecture this is caused by the same reason described in 3.4.

Figure 13(b) shows the cumulative distribution of Hop Count Penalty (HCP). HCP is defined as the difference between overlay node hop count per latency optimized path on k-MST meshes on top of the PLUTO Mesh (plus the PLUTO Mesh itself) and on the complete mesh. The plot shows that k-MST meshes (and the PLUTO Mesh) incurs HCP and the k-MST with greater k suffers from larger HCP. Only 5% of paths on the PLUTO mesh add more than 2 extra hops, while about 30% of paths on 4-MST mesh observe the same extra overhead, compared to the paths on complete mesh.

Figure 14(b) shows the cumulative distribution of Relative Bandwidth Penalty (RBP) of the bandwidth optimal paths. RBP is calculated as a ratio of the bandwidth difference (`ron1 - ron2`) to the reference bandwidth (`ron1`), which represents how much (relative) bandwidth is sacrificed when we reduce the number of virtual links. The plot shows 10% of paths on PLUTO mesh and 20% of paths on 4-MST mesh have lost more than 50% of the possible bandwidth.

4.2.3 Mesh Redundancy

We now examine the redundancy metrics defined in Section 3.1. Obviously, the *resilience* metric of k-MST mesh is k, since by its definition, it includes k edge-disjoint MSTs in the graph.

To evaluate the *duplicate* 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 for latency values. These two meshes have *resilience* metric of 5. Figure 15 shows the *duplicate* metric at both AS granularity and router granularity. We have conducted all-pairs traceroutes to obtain a router sequence and an AS sequence for every pair of nodes in the meshes. Figure 15(a) shows the difference (`kmst1-kmst2`) in the occurrence of duplicate AS hops of a given length between all pairs of routes. For instance, compared to `kmst2`, `kmst1` has about 1000 more pairs of routes that have duplicate AS hops of length 1. As this graph shows, `kmst2` achieves less duplicate AS hops of all the lengths. An interesting observation with this graph is that there are more pairs of routes in `kmst2` that have duplicate AS hops of length 0. This means that `kmst2` has over 2000 more edge-disjoint pairs of routes than `kmst1`.

Finally, Figure 15(b) shows a similar plot for the difference (`kmst1-kmst2`) in the occurrence of duplicate router hops of a given length between all pairs of routes. As can be seen in the graph, `kmst1` has more long duplicate router hops, while `kmst2` has more of short duplicate router hops. This implies that `kmst2` has reduced the number of duplicate router hops as a whole. Although we do not have a clear reason why we have increased the number of shorter duplicate router hops in `kmst2`, we believe that it may have resulted from the fact that our method uses only static and passive information, namely AS paths and geographical information. We should also note that `kmst2` has more edge-disjoint pairs of routes at router

granularity as well.

5 Discussion

This section discusses several issues raised by this work.

5.1 Scalability

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.

It is also the case that the more overlay nodes there are, the greater the opportunity to prune edges. Of course, it matters how widely the nodes are distributed over the network. If we assume overlays are formed from a randomly distributed participants, we expect the chances of having nodes at network cross-roads will increase, as the number of ASes where overlay participants resides grows. Using a peering graph with 15,396 ASes and 69,496 peering edges extracted from RouteViews and PlanetLab BGP feeds, we have evaluated reduction of virtual links when there are 1,000 overlay nodes (each node resides in a distinct AS) sampled randomly [22]. We found that the pruning algorithm in Figure 2(a) can reduce the number of virtual links by 70%, and the one in Figure 2(b) can reduce it by 90%. This implies that assuming a random distibution of the overlay participants, our scheme is expected to achieve 70% of routing overhead reduction.

There is yet another way to interpret 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, as a aside, although we used PlanetLab nodes where node distribution is biased toward GREN (Global Research and Educational Network) [7], 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 how broadly the overlay covers that network than the specific link characteristics.

5.2 Routing Optimization

One of the main challenges in routing using end-systems is to overcome the extra latency overhead in routing and forwarding packets, since the computation for routing and forwarding takes place at application level. In our experiments on PlanetLab, we did observe such undesirable extra overhead especially when the computational and network resources on the node responsible for forwarding packets become scarce, since PlanetLab is a shared test-bed. This is especially true if we use multiple-hop indirection as in our experiments using modified RON. Since one of the goals of routing overlay is to achieve better performance than the BGP-provided route, this may become a critical issue. However, here is an optimization to compensate for this problem. For instance, in RON, each node calculates its route using link-state information disseminated among participants. If the best route that a node calculate to a destination at AS level is identical to the actual AS path, then we do not need to forward the packets through end-systems, but simply let the Internet deliver them.

5.3 IP2Geo

Although it is not unreasonable to have each overlay node report its geographical location (e.g., city/state, longitude/latitude), it is desirable to have a service to translate an IP address into geographical location. Although there are both academic efforts [26] and various commercial products attempting to provide this mapping service [2, 3], to the best of our knowledge, there is no satisfactory open IP2Geo mapping service available to date. Here is a short report on our feasibility study toward IP2Geo service.

As pointed out in [26], it is important to cluster IP addresses that are supposedly located in close vicinity. It is also important to identify IP addresses in use, since there is no value in trying to identify the location of unused IP addresses. First, we cluster routable IP addresses using BGP tables into smaller networks than /24's. Using BGP tables from RouteViews and PlanetLab nodes, we identified a little over 7 million such networks. Note that usually BGP tables includes larger networks like /8's and /16's, so we divided those network into /24's since these large networks simply do not advertise smaller networks to outside.

Although there are 7 million routable small networks, it is not necessarily true that all of these networks are in use. For this reason, we tried reverse DNS lookups for almost all the IP addresses. Since PTR records [21] are managed hierarchically by the `in-addr.arpa` domain, we have done hierarchical lookups where only when upper-octet network `x.in-addr.arpa` exists, we further perform lookups for lower-octet networks `y.x.in-addr.arpa`. [1] We have used PlanetLab nodes to distribute the load of reverse DNS lookups and identified about 1.85 million /24's, 265.88 million individual IP addresses in use. Although this is a conservative estimate of the number of networks in use, since there are existing hosts that do not have PTR records in DNS, we believe this is a good start for bootstrapping open IP2Geo

[1] A few networks whose lookups timeout (not returning non-existing domain error, NXDOMAIN (RCODE=3) [21] but simply timeouts) sometimes include lower-octet networks that do exist. For instance, lookups for 214/8, 215/8 (DoD) always timeout, but several /16's under this /8's do exist.

service.

Second, after clustering and identifying networks in use, we still need to locate their geographical location for about 1.85 million networks. Figure 16 shows /24 networks bitmap showing which network has valid IP addresses in it. We are currently using the traditional technique of using traceroute to obtain DNS name of router nearby the target, and from DNS name, we can often infer its geographical location [26, 33]. We have also found that the results from reverse DNS lookups we performed give us clue as to where the particular IP addresses is located. For example, most IP addresses assigned to DSL, PPP and cable modems in major ISPs do have geographical locations, such as `lsanca1-ar3-008-034.biz.dsl.gtei.net` (Los Angels, CA). Since these addresses are not necessarily responsive to ping or traceroute, this is viable method to identify the location of those IP addresses. Also, geographical locations of IP addresses in educational institutions and large companies tend to be easily identifiable by looking at their DNS name.

Finally, we combine this DNS approach with BGP information. If the BGP table shows an AS has relatively small number of peers, then we infer that this is a regional ISP, so we can cluster /24's networks obtained from our DNS record method even further and thus we will be able to greatly suppress the amount of active probing such as traceroute.

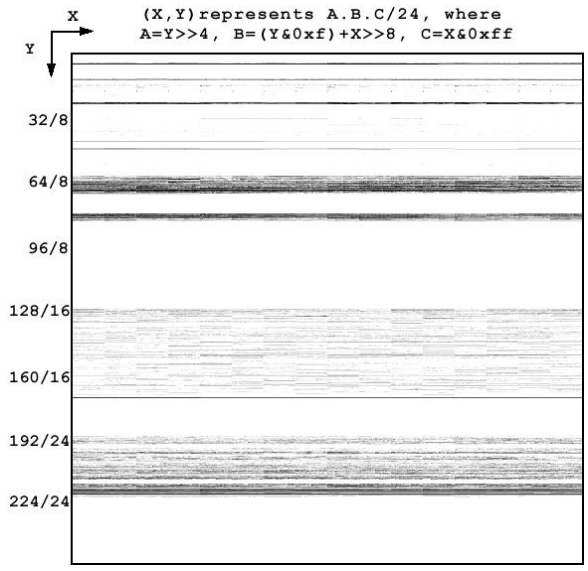


Figure 16: Networks that have at least one valid PTR records. Each dot in this 4096×4096 pixel image represents a /24 network. Y-axis, x-axis represent upper and lower 12bits of /24's network addresses, respectively. The darkness (8-bit gray-scale) of each dot represents how many IP addresses have PTR records in the /24 network.

5.4 Divert

We have designed another orthogonal idea called *divert* [23] to make the routing overlay scalable. In our design, we made a clear distinction between *desktop* and *overlay* nodes. A desktop is a user's machine that subscribes to a routing overlay network. Desktops are usually under full control of the user. All privileged operations, such as loading kernel modules and setting up firewall rules, are allowed on desktop machines. On the other hand, overlay nodes are shared, protected, and restricted, intermediate resources, where privileged operations are prohibited, or modified to be restricted or protected in some way. Our approach moves route calculation to dedicated overlay nodes, where and each desktop (end-system) associates itself with a nearby overlay node, for instance, within the same AS. This effectively allows routing overlay networks to aggregate traffic on behalf of a larger collection of end-systems, hence make overall system much more scalable. It also means routing overlay networks can support thinner end-systems, for example mobile nodes that cannot afford the measurement burden routing overlay networks impose. We have implemented divert using RON in PlanetLab and demonstrated it works with several applications such as `ssh`.

6 Related Work

It is not uncommon that an overlay network maintains a subgraph of the complete graph as a routing mesh. As opposed to RON [6]'s complete routing mesh scheme, several single-source application-level multicast overlays ESM [9, 10], YOID [27], Overcast [16], Bullet [17], SplitStream [8] define more sparse routing meshes and sometimes build multicast trees on top of them. Interleaved Spanning Tree [35] proposed an distributed algorithm to define sparser mesh for generic overlay networks, and this is probably the most related to our approach. However, these mesh building strategies are still treating the Internet as a black box and rely on expensive performance measurements before constructing a routing mesh, and does not capture the idea of duplicate segments in the underlying network. To the best of our knowledge, our work is the first to attempt defining an infrastructure to construct topology-aware representative mesh in a cost-effective manner. As for estimating latency, there were similar research efforts to inexpensively predict the end-to-end distance for a given pair of nodes. IDMaps [14] explored the feasibility of a public infrastructure to provide end-to-end distance information. GNP [24], Lighthouses [29], ICS [19], Virtual Landmarks [34], and PIC [11] proposed a coordinate-based strategy to predict distance treating the Internet as a high-dimensional geometric space. Our PLUTO MLE is novel in that it uses only passive (BGP tables) and fairly static (geographical location of PoPs of major ASes) and achieves 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.

Another contribution is that we have developed subservices such as PLUTO MLE to infer minimum latency using only passive and static information, and AS Traceroute server to efficiently obtain and cache AS paths, between an arbitrary pair of nodes. Although these services are developed as a supplemental service or for evaluation purpose only, these services potentially evolve into stand-alone generic service by themselves.

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-conscience (scalable) and effective.

References

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

[2] GEOBYTES. http://www.geobytes.com/.

[3] IP2Location. http://www.ip2location.com/.

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

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

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

[13] 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.

[14] 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.

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

[16] 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.

[17] 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.

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

[19] 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.

[20] 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.

[21] P. Mockapetris. Domain Names - Implementation and Specification, November 1987. RFC 1035.

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

[23] A. Nakao, L. Peterson, and M. Wawrzoniak. A Divert Mechanism for Service Overlays. Technical Report TR–668–03, Department of Computer Science, Princeton University, February 2003.

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

[25] J. Padhye, V. Firoiu, D. Towsley, and J. Krusoe. 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.

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

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

[28] 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.

[29] 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.

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

[31] 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.

[32] 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.

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

[34] 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.

[35] 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.