Heuristics for Internet Map Discovery

Ramesh Govindan, Hongsuda Tangmunarunkit
USC/Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292, USA

Abstract— **Mercator is a program that uses hop-limited probes—the same primitive used in *traceroute*—to infer an Internet map. It uses *informed random address probing* to carefully exploring the IP address space when determining router adjacencies, uses source-route capable routers wherever possible to enhance the fidelity of the resulting map, and employs novel mechanisms for resolving *aliases* (interfaces belonging to the same router). This paper describes the design of these heuristics and our experiences with Mercator, and presents some preliminary analysis of the resulting Internet map.**

I. INTRODUCTION

Obtaining a router-level map of the Internet has received little attention from the research community. This is perhaps unsurprising given the perceived difficulty of obtaining a high-quality map using the minimal support that exists in the infrastructure. However, as we show in this paper, it is useful, and possible, to get an approximate map of the Internet. Such a map is a first step in trying to understand some of the macroscopic properties of the Internet's physical structure. Other potential uses of Internet maps have been described elsewhere [17] and we do not repeat them here.

This paper documents a collection of heuristics, some well-known and some obscure, for inferring the router-level map of the Internet. Of the many possible definitions of the word *map*, the one we choose for the purposes of this paper is: a graph whose nodes represent routers in the Internet and whose links represent *adjacencies* between routers. Two routers are adjacent if one is exactly one IP-level hop away from the other. In a Section IV-A, we discuss the implications of this definition, and describe how well our map collection heuristics allow us to infer the complete Internet map. Inferring the number, or IP addresses, of Internet hosts is an explicit non-goal.

Perhaps the only ubiquitously available primitive to infer router adjacencies is the *hop-limited probe*. Such a probe consists of a hop-limited IP packet, and the corresponding ICMP response (if any) indicating the expiration of the IP *ttl* field (or other error indicators). The *traceroute* tool uses this primitive to infer the path to a given destination. Generally speaking, all earlier mapping efforts [17], [4], [6] have computed router adjacencies from a sequence of traceroutes to different Internet destinations. The destinations to direct the traceroutes are usually derived from one or more *databases* (e.g. such as routing tables, the DNS, or a precomputed table of host addresses). Finally, with one exception, all earlier mapping efforts have attempted to map the Internet by sending hop-limited packets from a single location in the network. Section II describes in greater detail these efforts at mapping the Internet.

In this paper, we describe the heuristics employed by our Internet mapper program, Mercator. Written entirely from scratch, Mercator *requires no input*, *i.e.,*, it does not use any external database in order to direct hop-limited probes. Instead, it uses a heuristic we call *informed random address probing*; the targets of our hop-limited probes are informed both by results from earlier probes, as well as by IP address allocation policies. Such a technique enables Mercator to be deployed anywhere because it makes no assumptions about the availability of external information to direct the probes. Mercator also uses *source-routing* to direct the hop-limited probes in directions other than radially from the sender. This enables Mercator to discover "cross-links"—router adjacencies that might otherwise not have been discovered. As we describe later, these heuristics can result in several *aliases* (interface IP addresses) for a single router. Mercator also contains some heuristics for resolving these aliases. Section III describes these heuristics in greater detail, and discusses the limitations of the resulting map.

In Section IV, we discuss our experiences with a deployment of mapper. We describe various techniques we have used to validate the map resulting from this deployment. Finally, we present some preliminary estimates of the size of this map, and analyze some graph-theoretic properties of the map. Finally, Section V summarizes our main contributions, and indicates directions for future work.

II. RELATED WORK

Several Internet mapping projects have, or are currently attempting to, obtain a router-level map of the Internet. The earliest attempt we know of [13] traced paths to 5000 destinations from a single network node. These destinations were obtained from a database of Internet hosts that, in 1995, had sent electronic mail to a particular organization. In addition, a small number (11) of these destinations were used as intermediate nodes in *source-routed* traceroutes to the remaining destinations. Although the use of source routing can result in greater map fidelity, it is unclear how complete the resulting map is, given that the chosen destinations essentially represent an arbitrary subset of hosts in the Internet.

More recently, researchers [4], [17] have used BGP backbone routing tables in order to determine the destinations of traceroutes. For each prefix in the table, they repeatedly generate a randomly chosen IP address from within that prefix. From traceroutes to each such address, they determine router adjacencies, building a router adjacency graph in this manner. As we show in Section III, this alone does not result in an Internet map as we have defined it. In particular, these techniques may miss backup links (for which [17] proposes—but does not report results of—tracing from several locations in the Internet). Furthermore, these traceroutes may discover two or more interfaces belonging to the same router; these projects do not propose techniques for resolving such aliases. Finally, the *skitter* tool [6] uses a database of Web servers to determine traceroute targets.

Techniques for collecting other representations of Internet, such as the AS (Autonomous System) topology, have also been documented in the literature. In one approach, traces of backbone routing activity over a period of several days have been used to infer inter-AS "links" [12]. Each link represents an inter-ISP peering or a customer-ISP connection. Instantaneous dumps of backbone routing tables have also been used to infer AS-level links [3].

In some cases, router support can be used to determine router adjacencies. For example, Intermapper [7] builds a list of router adjacencies by recursively interrogating routers' SNMP [5] MIBs. A similar technique can also be used for the Internet's multicast overlay network, the MBone [18]. Routers on the MBone support an IGMP query that returns a list of neighbors.

Our interest in heuristics for Internet mapping are motivated by the desire to understand network structure better. In this paper, we present some preliminary results from analyzing the resulting map. More generally, a high quality Internet map can be used to validate compact topology models such as those proposed in [9]. Such models, or the resulting Internet map itself, can be used as input to simulations [1]. Moreover, high quality Internet maps can also be used to validate hypotheses about scaling limits of real networks [15].

III. Mapping Heuristics

In this section, we discuss the design of several heuristics for mapping the Internet. This design is driven by several goals and requirements, which we discuss first.

A. The Challenge

The challenge we set ourselves at the outset of this project was to find a collection of heuristics that would allow us to map the Internet:

From a single, arbitrary, location,

Using **only** hop-limited probes.

Why is this an interesting formulation of the mapping problem?

We chose the first restriction for two reasons. First, deployment of the mapping software then becomes trivial, especially if the heuristics do not require a specific topological placement (*e.g.,* at exchange points or within backbone infrastructures). In fact, the results described in this paper were obtained by running the Mercator software on a workstation at the edge of a campus network. Second, while it is certainly feasible to design a distributed mapping scheme, we chose to defer the complexity of implementing such a scheme until after we had explored the centralized mapping solution. Upon first glance, requiring our mapping software to run from a single node might seem too restrictive. It would appear that the single perspective provided by this restriction could result in large inaccuracies in the map. As we show later in Section III-D, the use of source routing can help alleviate these inaccuracies.

The second restriction—using *only* hop-limited probes—makes only minimal assumptions about the availability of network functionality. This restriction follows from the first, allowing the mapping software to be deployed anywhere in the network. It also implies that we explicitly chose *not* to use any external databases (the DNS, routing table dumps) to drive map discovery. Not only does this choice pose an academically interesting question (How can we map an IP network starting with nearly zero initial information?), but it can also lead to more robust heuristics for map discovery. As we show later, not all backbones have routing table entries for all Internet addresses (Section III-E, and not all router interfaces are populated in the DNS (Section IV-C).

There are several secondary requirements that inform the design of map discovery heuristics.

Obviously, the resulting map must be *complete.* Clearly, this requirement conflicts with the single-location restriction; using hop-limited probes from a single location cannot possibly reveal all router adjacencies. In Section IV-A, we argue that our heuristics can give us nearly complete maps of the *transit* portion of the Internet.

The map discovery heuristics must not impose significant probing *overhead* on the network.

Informally, the heuristics must not result in significantly slower map discovery compared to existing approaches. Rather than quantify this requirement, we chose to sacrifice rapidity of map discovery in favor of completeness and reduced overhead wherever necessary. This choice has interesting consequences, as described in Section IV-A.

B. Informed Random Address Probing

Mercator uses hop-limited probes to infer router adjacencies, but does not use external databases to derive targets for these probes. In the absence of external information, one possible heuristic—with obvious convergence implications—is to infer adjacencies by probing paths to addresses randomly chosen from the entire IP address space [17]. In this section, we describe a different heuristic, *informed random address probing*.

The goal of this heuristic is to guess which portions of the entire IP address space contain addressable nodes. An addressable node is one which has IP-level connectivity to the public Internet. Because IP addresses are assigned in prefixes, Mercator makes *informed* guesses about which prefixes might contain addressable nodes. From within each such *addressable prefix*, it then uniformly randomly selects an IP address as the target for one or more hop-limited probes (described in Section III-C). Mercator uses two techniques to guess addressable prefixes:

1. Whenever it sees a response to a hop-limited probe from some IP address , Mercator assumes that some prefix of must contain addressable nodes. In this sense, Mercator is *informed* by the map discovery process itself.
2. If is an addressable prefix, Mercator guesses that the neighboring prefixes of (*e.g.,* `128.8/16` and `128.10/16` are the neighboring prefixes for `128.9/16`) are also likely to be addressable[1]. This technique is based on the assumption that address registries delegate address spaces sequentially.

In the following paragraphs, we describe the details of informed random address probing.

In a given instance of Mercator, repeated application of these two techniques leads to a gradually increasing *prefix population.* In order for both the above techniques to work, the prefix population must be seeded with at least one prefix. Mercator uses the IP address of the host it is running on to infer this seed prefix. Then, technique 2 above ensures that this prefix population eventually covers the entire address space. Technique 1 attempts to ensure that addressable prefixes are explored early, leading to more rapid map discovery. Technique 1 alone is insufficient for complete map discovery. For some choices of seed prefix, using this technique alone might only result in a campus-local map.

For technique 1, we need a heuristic that infers the length of an addressable prefix from a router's IP address . In classful terms [16], if is a class A (class B) address, Mercator assumes that 's prefix length is 8 (respectively 16). This heuristic is based on pre-CIDR [11] address allocation policies. If is a class C adddress, Mercator assumes that is a prefix of length 19. This assumption is based on the practice of some top-level address registries in allocating CIDR blocks of length not greater than 19. For simplicity, we chose relatively coarse-grained guesses about the size of addressable prefixes. As a result, only a small portion of an addressable prefix might actually contain addressable nodes. However, because we probe randomly (and repeatedly, as described in Section III-C) within , the choice of the prefix length does not affect the completeness of the resulting map, only the rapidity of the discovery.

Finally, we need a heuristic that determines how often technique 2 is invoked, and which addressable prefix's neighbor is chosen. If, within some window application of technique 1 has not resulted in an increase in the prefix population, Mercator selects a neighbor of some existing prefix . In our implementation, is chosen to be 3 minutes. is chosen from among those prefixes in the population at least one of whose neighbors is not in the population. Among these prefixes, is selected by a lottery scheduling [20] algorithm. The number of lottery tickets for each prefix is proportional to the fraction of *successful probes* (Section III-C) on that prefix. This heuristic attempts to explore neighbors of those prefixes that are more densely populated with addressable nodes.

C. Path Probing

To discover the Internet map, Mercator repeatedly selects a prefix (in a manner described later in this subsection) from within its population, and probes the *path* to an address selected uniformly from within that prefix. Like traceroute, Mercator sends UDP packets to with successively increasing *ttl*s. To minimize network traffic, the path probe is self-clocking—the next UDP packet is not sent until a response to the previous one has been received. The probing stops either when is reached, or when a probe fails to elicit a response, or a loop is detected in the path. Unlike traceroute, the latter two termination conditions are appropriate for Mercator since our interest is in inferring router adjacencies.

A path probe results in a sequence of routers

Note that Mercator does not rely on these neighboring prefixes being topologically related—*e.g.,* assigned by the same ISP—to .

such that responded with an ICMP *time exceeded* for a UDP probe with ttl 1, and so on. From this sequence, Mercator inserts into its map nodes corresponding to , *etc.*, and links , *etc.*, if these nodes and links were not already in the map.

To reduce path probing overhead, not all path probes start at ttl 1. Rather, from the results of each path probe for prefix , Mercator computes the furthest router in that path that was already in the map at the time the probe completed. Subsequent path probes to start at the ttl corresponding to . If the first response is from , Mercator continues the path probe, otherwise it *backtracks* the path probe to ttl 1. This technique allows Mercator to avoid, where possible, rediscovering router adjacencies. Moreover, it reduces the probing overhead on routers in the vicinity of the host on which Mercator executes.

In what order are prefixes selected for probing? Rather than selecting prefixes in round-robin fashion, Mercator uses the lottery scheduling algorithm [20] where each prefix has lottery tickets proportional to the fraction of successful probes addressed to the prefix. A probe is deemed successful if it discovers at least one previously unknown router. With lottery scheduling, then, Mercator's probing is biased towards recently created prefixes densely populated with addressable nodes. This heuristic attempts to speed up map discovery.

Finally, Mercator is designed to allow multiple path probes to proceed concurrently. This configurable number can be used to tradeoff rapidity of map discovery for increased overhead.

D. Source-Routed Path Probing

Intuitively, in a shortest-path routed network, one might expect that path probing (Section III-C) results in a tree rooted at the host running Mercator. For two reasons, however, path probing actually discovers a richer view of the topology:

Inter-domain routing in the Internet is policy-based [19]. Policies can result in widely divergent paths to two topologically contiguous routing domains (Figure 1(a)).

Mercator continuously probes each addressable prefix over several days. It can therefore potentially discover *backup* paths to addressable prefixes (Figure 1(b)).

Even so, because Mercator attempts to discover the Internet topology from a single location, it may miss some "cross-links" in the Internet map. An obvious approach to increasing the likelihood of discovering all links in the Internet map is to run Mercator from different nodes in the network.

In this section, we describe a different solution to the problem, *source-routed path probing*. Essentially, our technique directs path probes to already discovered routers *via* source-route capable routers in the Internet (Section 1(c)). Not all routers in the Internet are source-route capable. In fact, only a small fraction (about 8%, see Section IV) support source routing. In some cases, these routers are located at small ISPs who have neglected to disable this capability in their routers. However, a number of large backbone provider routers are also source-route capable. Presumably, these providers use source-routed traceroutes to diagnose connectivity problems in their infrastructures.

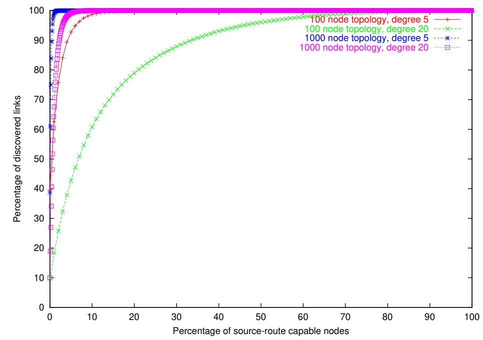
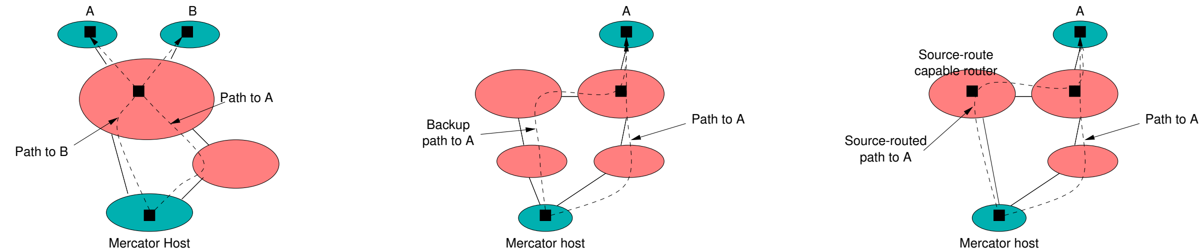


Fig. 2. In relatively sparse random networks, only a few source-route capable nodes (< 5%) are sufficient to discover 90% of the links. The one plot in this figure that doesn't satisfy this criterion is a dense random graph (the 100 node graph with a degree 20). This figure serves to illustrate the intuition behind the use of source-routed path probing.

Can such a small number of source-route capable routers help us discover cross-links? Roughly speaking, each source-route capable router can give us the same perspective as an instance of Mercator running at the same location. How effective a small fraction of routers is in helping us discover the entire map depends on several factors including the structure of the Internet, the location of the host running Mercator, and the placement of these source-route capable nodes. To see whether there is even a *plausible* argument for the efficacy of source-routed path probing, we conducted the following experiment. We generated random graphs of different sizes with different average node degrees. For each such graph, we computed the percentage of links discovered as a function of the fraction of source-route capable nodes randomly placed in the network. We found that (Figure 2) for a relatively sparse graph (one with a small ratio of the average node degree to the total number of nodes), 5% of source-route capable routers is enough to discover 90% of the links.

To detect source-route capability in a router , we attempt to send a source-routed UDP packet *via* to a randomly chosen routers from our map. This UDP packet is directed to a randomly chosen port and will usually



(a) Routing policy can lead to richer topology views....

(b) ... as can backup paths

(c) Source-routing provide a different perspective for probing

Fig. 1. Continuous path probing does not only result in a shortest-path tree routed at the host running Mercator. The ellipses represent autonomous systems. Where available, source routing can lead to richer topology views.

elicit an ICMP `port unreachable` message from the target router. If we do receive such a response, we mark source-route capable. We conduct this test for *every* router discovered by the path probing heuristic. Furthermore, for each router , we repeatedly test for source-route capability (to limit overhead, only a small number of such tests are run concurrently). We do this because a given test may fail for reasons other than the lack of source-route capability: may be unreachable at the time the test was conducted, may not have a route to , and so on.

Once a router is deemed source-route capable, we repeatedly attempt to send path probes via to randomly chosen *routers* from our map. This method of choosing probe targets is different from that used by path probing (Section III-C). We chose this approach only to avoid targeting hosts and setting off alarms—many host implementations automatically log source-routed packets. Even so, our probing activity was noticed by several system administrators (Section IV-B). In order to reduce probing overhead, the source-routed path probe starts at . The initial hop-count is inferred from the response obtained when was first discovered. This initial hop-count guess might be incorrect (*e.g.,* because the path to has changed), but this does not affect the *correctness* of our map discovery. As before, we send hop-limited probes with successively increasing *ttl*s, terminating only if we reach the destination, a loop is detected in the path, or a small number (three, in our implementation) of consecutive hop probes have failed. This last criterion was empirically chosen to carefully balance the overhead of probing against the rapidity of map discovery.

E. Alias Resolution

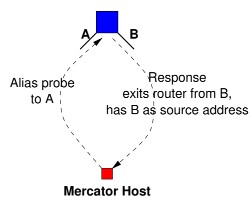
To simplify the preceding discussion, we said that path probes (source-routed or otherwise) discover *routers*. Path probes actually discover *router interfaces.* Thus, a single Mercator instance might discover more than one interface belonging to the same router (*i.e.,* multiple *aliases* for the router). For example, because of policy differences, paths from the Mercator host to two different destinations can intersect (Figure 1(a)). Similarly, the primary and backup paths to a destination might overlap (Figure 1(b)). Finally, a source-routed path probe can, because it probes from a different perspective, discover additional router interfaces (Figure 1(c)).

Resolving these aliases is an important step for obtaining an accurate map. Unfortunately, these aliases cannot be resolved by examining the syntactic structure of interface addresses. This is because a router's interfaces may be numbered from entirely different IP prefixes; this commonly occurs at administrative boundaries. The DNS is equally ineffective for these purposes; where interfaces are assigned DNS names, different interfaces are assigned different names. At administrative boundaries, different interfaces of a router may actually be assigned names belonging to different DNS domains.

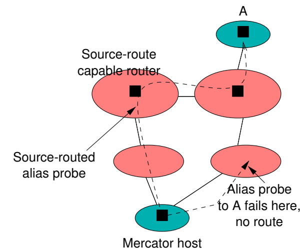
Our solution leverages a *suggested* (but not required) feature of IP implementations [2]. Suppose a host addresses a UDP packet to interface A of a router (Figure 3(a)). Suppose further that that packet is addressed to a non-existent port. In what follows, we call such a packet an *alias probe*. The corresponding ICMP *port unreachable* response to this packet will contain, as its source address, the *address of the outgoing interface for the unicast route towards* . In Figure 3(a), this interface is B. We have verified this behavior of IP stacks belonging to at least two major router vendors.

This feature suggests a simple heuristic for alias resolution. Send an alias probe to interface X. If the source address on the resulting ICMP message (assuming there is one) is Y, then X and Y are aliases for the same router.

This heuristic was also suggested in earlier work [13]. However, Mercator uses two additional refinements necessary to correctly implement alias resolution.



(a) The alias probe



(b) The source-routed alias probe

Fig. 3. Repeated alias probing can resolve router aliases. In some cases, source-routed alias probes are necessary for alias resolution.

The first refinement repeatedly sends alias probes to an interface address. To see why this is necessary, consider the following scenario. Suppose, as in Figure 3(a), a router has two interfaces A and B. Suppose further that, at some instant, an alias probe to A returns A itself. This implies that, *at that instant*, the router's route to the sending host exits interface A. It is possible that, at some later instant, an alias probe to B returns B. This can happen if, at the instant the second probe is sent, the router's route to the sending host exits interface B. Mercator therefore repeatedly sends alias probes to every known interface. To limit probing traffic, only a small number of alias probes are run concurrently. The efficacy of such repeated alias probing is based on the observation that there exist *dominant* paths [14] and routes [12] in the Internet. There is a high likelihood that, eventually, a large fraction of the alias probes will be returned via the router's dominant path to the sending host.

The second refinement uses source-routed alias probes. Figure 3(b) explains why this is necessary. In practice, large backbones do not have *complete* routing tables. Thus, the backbone(s) to which the sending host is eventually connected may not be able to forward an alias probe to its eventual destination. To circumvent this, Mercator attempts to send alias probes *via* source-route capable routers. This random, combinatorial search, of all source-route capable routers is a time-consuming process. However, in the absence of other information about where one might find a route to a given interface, we believe this is the only option left to us.

F. Mercator Software Design

Although some of the probing primitives that Mercator uses are available in the *traceroute* program, we chose to implement Mercator from scratch. Doing so allowed us the flexibility of trying different probing heuristics (*e.g.*,, different termination conditions for source-routed path probing, Section III-D). It also enabled us to carefully tune the overhead imposed by probing without sacrificing the rapidity of map discovery (*e.g.*,, the path probing optimization described in Section III-C).

Mercator is implemented on top of *Libserv*, a collection of C++ classes written by one of the authors that provides non-blocking access to file system and communication facilities. This allows Mercator to have several outstanding probe packets, and to independently tune the number of outstanding path probes, source-routed path probes, and alias probes. Together with Libserv, Mercator is about 14,000 lines of C++ code.

Mercator periodically checkpoints its map to disk. It is completely restartable from the latest (or any) checkpoint. We made several heuristic revisions during the implementation of Mercator, and this feature allowed us to avoid re-discovering sections of the Internet topology. It also allowed us to recover from buggy implementations of heuristics.

IV. Experiences and Preliminary Results

Although the heuristics described in Section III seem plausible, several questions remain unanswered: Is the resulting Internet map complete? What techniques might we use to validate the resulting map? How well do the heuristics work? What does the map look like? This section attempts to answer some of these questions.

A. Implications of our Methodology

A careful examination of the heuristics described in Section III reveals several important implications of our methodology. The "map" discovered by Mercator is *not* a topology map—it does not enumerate all router interfaces, and does not depict shared media.

First, because it uses path probing, Mercator cannot discover all interface addresses belonging to a router. Instead, it discovers only those interfaces through which paths from the Mercator host "enter" the router. Source-routed path probing can help increase the number of interfaces discovered, but our use of source-routed path probing does not guarantee that all routers are discovered.

Second, Mercator does not implement heuristics for discovering shared media. To do this, it would have to infer the subnet mask assigned to router interfaces. Unfortunately, two potential probing techniques for inferring subnet masks do not work very well. The ICMP query that returns the subnet mask associated with an interface is not widely implemented. Sending ICMP echo requests to different broadcast addresses (corresponding to different subnet masks) and inferring the subnet mask from the broadcast request that elicits the most responses [17] does not work either—this capability is now disabled in most routers in response to *smurf* attacks. It may be possible, however, to infer shared media by isolating highly meshed sections of our Internet map; this remains future work.

Mercator's map is *not* an instantaneous map of the Internet. To reduce probing overhead, Mercator limits the number of outstanding path probes. As a result, it takes several weeks (Section IV-B) to discover the map of the Internet. During this interval, of course, new nodes and links may have been added; Mercator is unable to distinguish these from pre-existing nodes. For this reason, the resulting map is a time-averaged picture of the network. Furthermore, Mercator discovers a time-averaged *routed* topology. If a router adjacency is used as a backup link, and that backup link is never traversed during a Mercator run, that adjacency will not be discovered. Similarly, if two routers are physical neighbors on a LAN, but never exchange traffic between themselves (*e.g.*,, for policy reasons), that adjacency is never discovered.

Finally, Mercator does not produce a *complete* map of the Internet. In particular, Mercator does not discover details of stub (campus) networks. Even though a single run of Mercator sends a large number of path probes, a given campus network is not probed frequently enough to discover the entire campus topology. As well, we believe that many campuses have source-routing turned off, so Mercator is unable to discover cross-links. However, we have a higher confidence in the degree of completeness of the Mercator map with respect to the *transit* portion of the Internet. This is because our probes traverse the transit portion more frequently (and this is true to a greater extent of the *core* of the Internet). We could obtain a more complete map by running multiple instances of Mercator and correlating the results. This is left for future work.

B. *Experiences*

We implemented the heuristics of Section III and ran one instance of Mercator on a PC running Linux. When configured with a limit of 15 concurrent path probes, Mercator takes about 3 weeks to discover nearly 150,000 interfaces and nearly 200,000 links. These numbers are greater than the corresponding numbers obtained from [4], serving as a simple validation of the completeness of our run. Before we discuss how we validated our Internet map, and what that map reveals in terms of the macroscopic Internet structure, we describe our early experiences in running Mercator and analyzing some of the data.

How well do our techniques for inferring addressable prefixes work? To answer this question, we first obtained the list of prefixes contain in a backbone routing table[2]. We then compared this with the prefixes inferred by Mercator. Only 8% of the prefixes in the routing table were not "covered" by any prefix inferred by Mercator. Conversely, 20% of prefixes inferred by Mercator did not have at least one overlapping prefix in the routing table. This latter figure is not surprising; our assumption that the "neighbor" of an addressable prefix is also addressable (Section III-B) can result in an overestimation of the addressable space. These numbers validate both the efficacy of the heuristic and the near completeness of our exploration of the address space.

How well does path probing work? There is a perception that many ISPs disable traceroute capability through their infrastructures. If this were true of the major US backbones, we would clearly not be able to see any routers belonging to these in our map. In fact, we do. There is also a perception that some ISPs configure their routers to not decrement TTLs across their infrastructures. If this were true, our map would contain many routers with a large outdegree (*i.e.,* the entire ISP appears to be one large router). We have found one possible instance of this, but no evidence that this practice is widespread. We conjecture that traceroute availability is higher than supposed by many researchers, because ISPs use this capability to debug their infrastructures.

We also found some instances of obvious misconfiguration. Mercator received at least one ICMP *time exceeded* message whose source address was a Class D (multicast) address. It was also able to successfully probe the path to some net 10 addresses; such addresses are reserved for private use and should not be globally routable [11].

Our experience with alias resolution has been mixed. Mercator was able to resolve nearly 20,000 interfaces. However, many other interfaces were unreachable from the Mercator host. Mercator could not therefore determine whether such interfaces were aliases for already discovered routers. Three causes may be attributed to this:

Some of these interfaces are numbered from private address space [11], and (with minor exceptions), this space

[2] From a public route server at `route-views.oregon-ix.net`.

is not globally routable. To these interfaces, then, the alias resolution procedure cannot be applied.

Some others are assigned non-private addresses, but the corresponding ISP does not propagate routes to these interfaces beyond its border. To these interfaces, again, the alias resolution procedure cannot be applied.

Finally, some of these interfaces are assigned non-private addresses, and some, but not all parts of the Internet core (and, in particular, the backbone ISP to which the Mercator host defaults) carry routes to these interfaces. To such addresses, we attempted source-routed alias resolution (Section III-E). This heuristic works, but is very slow, because it involves a random search among all source-route capable routers. As of this writing, we were able to resolve only about three thousand interfaces using this technique.

To date, Mercator has discovered about 3,000 distinct[3] private addresses. A total of 15,000 interfaces are not directly reachable from Mercator, and its unclear how many of these can be reached using source-routing.

Given the widely-held belief that "source-routing does not work", our experiences with source-routing path probing (Section III-D) are probably of most interest. We summarize our experiences below:

About 8% (nearly 10,000) routers on the Internet *are* source-route capable. Of these routers, several belong to large US backbones, implying that they are actively used (presumably to debug ISP infrastructures using source-routed traceroutes).

Source-routed path probing works reasonably well (see the next bullet), although, for a given traceroute, not all intermediate hops respond always respond. However, this is sufficient for our purposes, since our goal is to infer router adjacencies, rather than study paths. Nearly 1 in every 6 links in our map was discovered using source-routed probing alone (*i.e.,* direct path probing did not subsequently re-discover these links).

We did encounter two bugs in source routing. First, there is at least one router in the Internet that responds to source routed packets as though the packet was sent directly to it. That is, this router completely ignores the IP loose source route option. Second, some earlier versions of a router vendor's operating system non-deterministically fail to decrement the TTL on UDP packets with the loose source route packet. We do not know the extent of these bugs. Of these bugs, the latter has the potential for corrupting our map. We appropriately account for this potential in our analyses of the map characteristics (Section IV-D).

Private addresses, by definition, are not required to be globally unique, so more than one physical interface can be assigned the same physical address.

Finally, our experiences with the "social" consequences of sending traceroutes were similar to those reported elsewhere [4]. Many tens of system administrators (largely from corporate sites containing firewalls which log UDP packets to non-existent ports) noticed our activity and registered abuse complaints with our institution.

C. *Validating the Map*

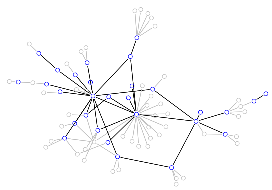
How did we validate the resulting Internet map? The previous section discussed simple validation tests for our map. Could we have used the data from [4] to carefully compare, on a link-by-link basis, our two maps? Although theoretically feasible, such a comparison would have required significant additional programming effort. That data set was gathered from a different location in the Internet. Consequently, it discovered different interface addresses than Mercator; to compare the two data sets, we would have had to resolve aliases (Section III-E) between the two sets of interface addresses. We have deferred such validation for future work.

Instead, we are currently validating *subgraphs* of our Internet map. Our strategy is to compare published ISP maps against ISP subgraphs extracted from our map. To extract ISP subgraphs, we infer routers belonging to an ISP from DNS names assigned to router interfaces (not all router addresses have corresponding names assigned in the DNS; we failed to complete the inverse mapping on nearly 30% of the discovered interfaces). Using this technique, and the *nam* network animator [8], we were able to visualize ISP structures (Figure 4). In some cases, ISPs use naming conventions that allow us to infer inter-city links, "core" routers, and customer connections. Using these, we can sometimes isolate the core portions of ISP networks; studying the structure of ISP networks is left for future work.

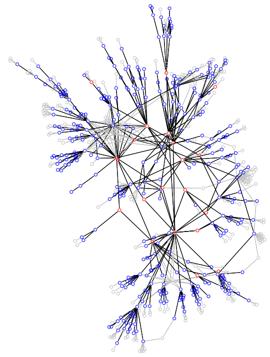
The extraction of ISPs is tedious enough, but the validation of the map is made much harder by the fact that most commercial ISPs do not publish router-level connectivity (this is regarded as proprietary information). To date, we have compared our discovered topology against the backbone structures of a local ISP and of part of a statewide research and educational network[4]. Mercator discovered all routers and all but 1 link in each respective networks. That these networks are topology close to the host which ran Mercator is probably irrelevant; our probing is not informed by topological closeness, so we expect that Mercator would show similar fidelity for distant, but similarly sized ISPs.

These validation results are somewhat encouraging. The

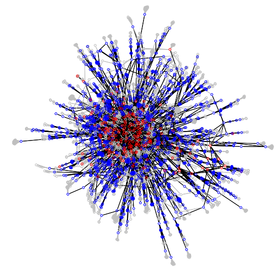
Los Nettos and Calren2 respectively.



(a) A small ISP

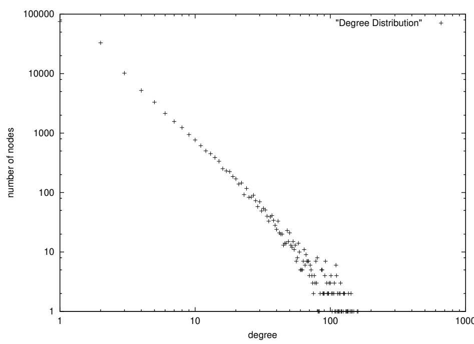


(b) A medium-sized ISP

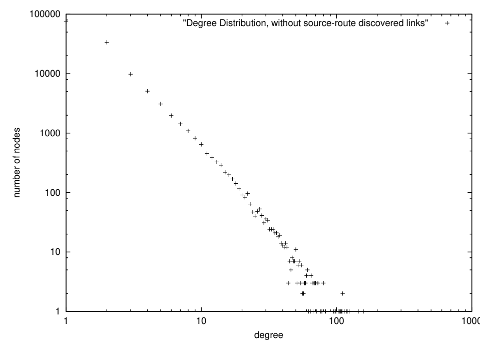


(c) A large national backbone

Fig. 4. Mercator can be used to visualize, and analyze, ISP structures.



(a) The degree distribution



(b) ... without source-route discovered links

Fig. 5. The degree distribution of the Mercator map.

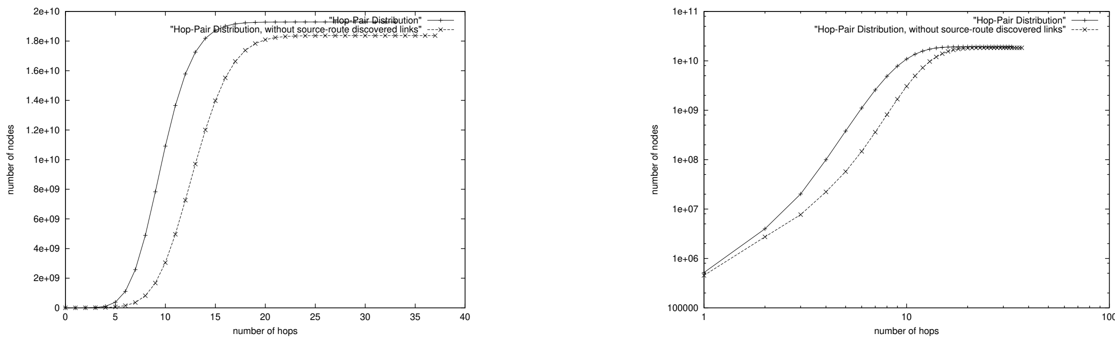
technique appears to give a fairly complete picture of ISPs which might be considered to be at the edge of the Internet's transit portion. It gives us greater confidence in the hypothesis that Mercator can map the transit portion with reasonable fidelity. There is one caveat, however. The connectivity structure of both networks are relatively simple; they both consist of simple rings, with a small number of "spur" links. It is easy to see that Mercator can map such sparse ISPs quite well. It is less clear how successful Mercator will be in mapping more meshed ISP structures (*e.g.,* those that consist of an ATM core with private virtual circuits established between routers). We are currently validating such structures in our map.

D. Discussion and Results

The previous subsections have indicated several reasons why Mercator's Internet map may be incomplete, and in parts maybe even incorrect. As future work, we hope to run Mercator from different locations around the Internet to see if we can improve the fidelity of the map. We believe, however, that the map we have generated still has some uses. It provides representative realistic topologies for protocol simulations [1], it can be used in heuristics for service proximity [10], and can provide visual context for isolating faults in networks [18].

Can this graph be used for understanding the large-scale structure of the Internet? On the one hand, we do not have mathematical bounds for the likely degree of inaccuracy of Mercator's map. Furthermore, some graph-theoretic properties, such as estimates of neighborhood sizes, can be very sensitive to missing links. This would argue against being able to draw reliable conclusions about network structure. On the other hand, we believe that, unless the Internet infrastructure improves greatly—in the sense that source-route capability, and other ICMP services, become more widely available, and routes to router interfaces are widely propagated—it is logistically difficult to obtain a map which has a significantly higher accuracy than our own. This would argue that we might never be able to confidently analyze the statistical properties of the Internet topology.

Despite this, and particularly because there is much re-



(a) The hop-pair distribution

(b) The hop-pair distribution on a log-log scale

Fig. 6. The hop-pair distribution of the Mercator map.

cent interest in trying to understand network structure [9], [15], we analyze below the degree distribution, and the hop-pair distribution [9] of the Mercator map. To understand whether the source-routing bug described in Section IV-B, we use two maps in computing these distributions: the complete Mercator map, as well as the map without any source-route discovered links.

Figure 5(a) plots the degree distribution on a log-log scale. The plot is interesting. For node degrees less than 30, the plot is linear, lending some support to the conjecture that a power law governs the degree distribution of real networks [9]. However, starting from about a degree of 30, the distribution is significantly more diffuse. This might indicate that a different law governs the distribution of high degree nodes—such nodes are usually found closer to the core of the network.

Figure 6(a) plots the number of pairs of nodes within at most hops of each other as a function of . This hop-pair distribution also has an intriguing shape. For small (less than about 5), the number of pairs of nodes within hops of each other increases relatively slowly. This could be an artifact of our methodology. More likely, we believe, is the explanation that beyond small , the number of nodes reachable within hops increases dramatically because nodes within a stub domain can reach other stub domains. Another interesting feature of this curve is that 97% of the hop-pairs are at a of 15 hops or less. Finally, Figure 6(b) plots the hop-pair distribution on a log-log scale. Can this distribution be described by a power law, especially for less than 15? Quite possibly, although the slight non-linearity at lower leads us to believe that that question still remains open, even ignoring methodological inaccuracies.

Figure 5 and Figure 6 also show that source-route discovered links do not skew the *qualitative* conclusions we might draw from these distributions. That is, our conclusions about the two distributions hold, but with possibly different numerical values, for the map without the source-route discovered links.

V. Conclusions and Future Work

This paper is, to our knowledge, the most extensive attempt to date to map the Internet. It documents several techniques to infer the Internet map, and reports on our experiences in designing and experimenting with different heuristics for increasing the fidelity of the map. Clearly, our results have been mixed. We have been able to explore more of the Internet than previous efforts, but have, in the process, revealed several reasons why it is exceedingly difficult to obtain a highly accurate map in the existing Internet infrastructure.

We intend to explore several directions in the future; running Mercator from more than one location, evaluating heuristics for inferring topological elements such as shared media, and validating sections of the map with the help of ISPs in order to try to bound the degree of inaccuracy of our map.

Acknowledgements

Deborah Estrin and Scott Shenker provided the motivation for this work, and gave valuable advice. Cengiz Alaettinoglu suggested significant improvement to earlier versions of this draft.

References

[1] Sandeep Bajaj, Lee Breslau, Deborah Estrin, Kevin Fall, Sally Floyd, Padma Haldar, Mark Handley, Ahmed Helmy, John Heidemann, Polly Huang, Satish Kumar, Steven McCanne, Reza Rejaie, Puneet Sharma, Kannan Varadhan, Ya Xu, Haobo Yu, and Daniel Zappala. Improving simulation for network research. Technical Report 99-702, University of Southern California, March 1999.

[2] R. Braden. Requirements for Internet Hosts — Communication

Layers. Request for Comments 1122, Internic Directory Services, October 1989.

[3] H.-W. Braun and K. C. Claffy. Global ISP Interconnectivity by AS number. http://moat.nlanr.net/AS/.

[4] Hal Burch and Bill Cheswick. Mapping the Internet. *IEEE Computer*, 32(4):97–98, April 1999.

[5] J. D. Case, M. Fedor, M. Schoffstall, and C. Davin. Simple Network Management Protocol (SNMP). Request for Comments 1157, Internic Directory Services, May 1990.

[6] K. C. Claffy and D. McRobb. Measurement and Visualization of Internet Connectivity and Performance. http://www.caida.org/Tools/Skitter/.

[7] Dartmouth University. Intermapper: An intranet mapping and snmp monitoring program for the macintosh. http://www.dartmouth.edu/netsoftware/intermapper.

[8] Deborah Estrin, Mark Handley, John Heidemann, Steven McCanne, Ya Xu, and Haobo Yu. Network visualization with the VINT network animator nam. Technical Report 99-703, University of Southern California, March 1999.

[9] C. Faloutsos, M. Faloutsos, and P. Faloutsos. What does Internet look like? Empirical Laws of the Internet Topology. To appear, ACM SIGCOMM 1999.

[10] P. Francis, S. Jamin, V. Paxson, L. Zhang, D. Gryniewicz, and Y. Jin. An Architecture for a Global Internet Host Distance Estimation Service. In *Proceedings of IEEE Infocom*, March 1999.

[11] V. Fuller, T. Li, J. Yu, and K. Varadhan. Classless Inter-Domain Routing (CIDR): An Address Assignment and Aggregation Strategy. Request for Comments 1519, Internic Directory Services, September 1993.

[12] R. Govindan and A. Reddy. An Analysis of Internet Inter-Domain Topology and Route Stability. In *Proc. IEEE INFOCOM '97*, Kobe, Japan, Apr 1997.

[13] J.-J. Pansiot and D. Grad. On routes and multicast trees in the Internet. *ACM SIGCOMM Computer Communication Review*, 28(1):41–50, January 1998.

[14] V. Paxson. End-to-end Routing Behavior in the Internet. In *Proceedings of the ACM SIGCOMM Symposium on Communication Architectures and Protocols*, San Francisco, CA, September 1996.

[15] G. Phillips, H. Tangmunarunkit, and S. Shenker. Scaling of Multicast Trees: Comments on the Chuang-Sirbu scaling law. To appear, ACM SIGCOMM 1999.

[16] J. Postel. Internet Protocol. Request for Comments 791, Internic Directory Services, September 1981.

[17] R. Siamwalla and R. Sharma and S. Keshav. Discovering internet topology. Unpublished manuscript.

[18] A. Reddy, D. Estrin, and R. Govindan. Large-Scale Fault Isolation. Technical Report 99-706, Computer Science Department, University of Southern California, March 1999. Submitted for publication.

[19] Y. Rekhter and T. Li. A Border Gateway Protocol 4 (BGP-4). Request for Comments 1771, Internic Directory Services, March 1995.

[20] C. A. Waldspurger and W. E. Weihl. Lottery Scheduling: Flexible Proportional-Share Resource Management. In *First Symposium on Operating Systems Design and Implementation (OSDI)*, pages 1–11. USENIX Association, 1995.