Network Maps beyond Connectivity

Cheng Jin Zhiheng Wang Sugih Jamin
{*chengjin, zhihengw, jamin*}*@eecs.umich.edu*

December 1, 2004

Abstract

Knowing network topology is becoming increasingly important for a number of applications such as server placement [1] and traceback of DDoS attacks [2]. Recent works in modeling the Internet topology and constructing network maps have focused on the connectivity aspect. This paper describes the first study in incorporating connectivity, latency, and routing information all into a network map based on a large set of `traceroute` data. We highlight the common challenges in constructing such network maps and discuss our solutions. We evaluate our network map based on various Internet routing models proposed in the literature. The evaluation shows that, for those `traceroute` data that we are able to evaluate, at least 85% of computed hop-counts and latencies are within a factor of two of the actual values. Furthermore, we show that a flat routing model based on hop-count performs as well as more complicated policy routing models.

1 Introduction

There is a frequent need in network research to learn the topology of the Internet or other networks of interest. The knowledge of network topologies proves useful in two ways. First, studies to uncover properties of network topologies can be made based on real network topologies. There have been a number of studies [3, 4, 5, 6, 7] that try to understand and model the Internet topology based on actual data. Second, the actual network topological information can be used by a number of applications and services, such as CDN placement [1, 8], and prevention of DDoS attacks [2, 9]. AS-level connectivity can usually be obtained from BGP routing table dumps [10, 6]. It is generally more difficult to obtain host-level connectivity since host-level routing tables are not readily available. There have been a number of studies [11, 12, 13] that focused on building host-level connectivity maps. Authors in [11] build network maps for visualization of network connectivity. Authors in [12] capture the transit portion of the Internet to study path inflation by policy routing, and authors in [13] focus on building complete connectivity maps of ISPs. All three mapping efforts produce only connectivity maps.

Connectivity information is sufficient to describe AS-level topology since no other metrics such as latency or bandwidth can be meaningfully assigned between many large ASs. Such is not the case with host-level network maps. Host-level network topologies also contain information such as latency and bandwidth. Latency and bandwidth information are often highly useful for performance studies. For example, for studies on TCP's throughput [14], in either simulations or real experiments, end-to-end latency determines how quickly a sender detects network congestion, and determines the throughput along with the bottleneck bandwidth. In simulations, end-to-end latencies and bottleneck bandwidths generally computed based on some routing model that computes paths among nodes inside a network. Presently, there are no studies on building a network map with more than connectivity information or on the best routing model to use for path computation. In this paper, we combine connectivity, latency, and routing information all into an interface-level network map based on Internet `traceroute` data. The interface-level network map differs

from the host-level network map in that only IP addresses, not actual hosts, are represented in the map. We will not attempt to resolve interface aliases [12, 6], i.e. a single router having multiple interfaces, since the network metrics we use for evaluations, hop-counts and latencies, are valid only on the IP interface level.

We emphasize that we are not proposing a network topology generator in this paper; instead, we are interested in utilizing a given set of Internet measurements to generate the most comprehensive network map covered by the measurements. We hope that an accurate network map based on real measurements will be useful to applications and services that require topological information, and will provide the basis for a more complete model of Internet topology. Our contributions in this paper are as follows:

- A detailed study on how to incorporate link latencies into topology construction.
- A survey of routing models that can be used for path computations.
- An evaluation of the network map showing that we can indeed compute hop-count and latency information with reasonable accuracy on this network map.

In the rest of the paper, we first introduce a framework for building a full-featured network map and give a survey of current approaches to building network maps in Section 2. We then explain the common challenges in incorporating latency information into network maps and discuss some of our solutions in Section 3. Section 4 surveys a number of routing models used to compute end-to-end latencies on our network map. In Section 5, we evaluate the network map by checking the computed hop-counts and latencies in the map against the actual data. We conclude our discussion in Section 6.

2 Building A Complete Network Map

The focus of this work is to define procedures and algorithms to build a full-featured network map that generates accurate end-to-end network metrics. Figure 1 illustrates our approach to constructing a full-featured network map. We separate a network map into three layers of information. The bottom layer is the connectivity information. The middle layer has static link information such as minimum link latency and nominal bandwidth. In this study, a minimum link latency is the sum of propagation delay on the physical link and the router processing delay under no load. We are not aware of any effort that tries to model either minimum link latencies or nominal bandwidths. The top layer captures dynamic events such as routing instability and packet queueing. There are a number of studies [15, 16, 17, 18] that address the dynamic behavior of the Internet, which can be incorporated into a static network map. Having the three layers of information alone is not enough to give us end-to-end network metrics. We also need to define network routing in order to compute end-to-end network metrics. We examine a number of routing models proposed in the existing literature to compute latencies in Section 4.

As the first step toward building a full-featured network map, we construct a network map that provides accurate end-to-end latencies. An end-to-end latency has both a static and a dynamic component. The static component of an end-to-end latency is the sum of propagation delay and fixed processing delays at all intermediate routers, i.e., the minimum latency. The dynamic component is the sum of queueing delays at intermediate routers. The two components are governed by different physical processes. The static latencies are governed by the propagation speed of light inside various mediums and semiconductor physics. The dynamic queueing delays are governed by packet arrival processes and users' network usages at different times of the day. Therefore, it is appropriate to model static latencies separately from dynamic latencies. In this paper, we build a network map that can accurately represent the static end-to-end latencies.

We are aware of only one current approach that can provide both connectivity and latency information on the interface-level. Authors in [19] propose a method that computes hop-by-hop link latencies by modeling path latencies as linear combinations of individual link latencies and then solving the system of linear

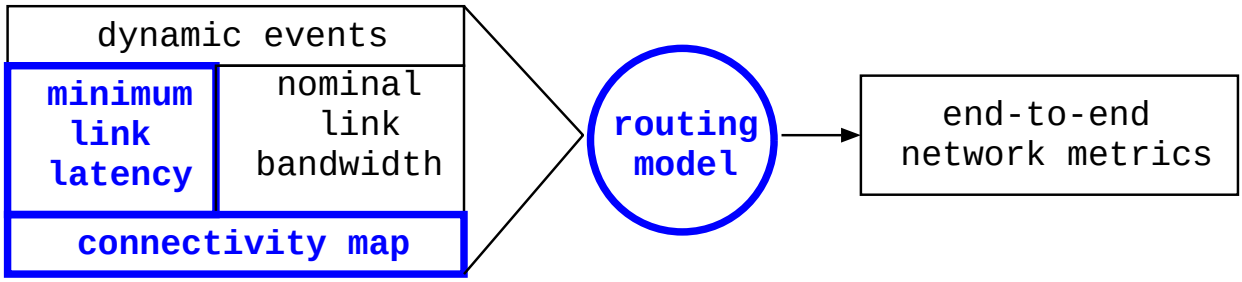

Figure 1: Components of a full-featured network map.

equations. The authors use `traceroute` to collect path and Round-Trip Time (RTT) data to solve for a number of previously unmeasurable link latencies. For example, if we have H measurement hosts and D destinations where $H \ll D$ typically, the number of RTT measurements we need to conduct is $O(D \cdot H)$. If we were to obtain the connectivity information among these hosts, we could have $O((D+H)^2) \approx O(D^2)$ latencies by doing distance computations among the $(D+H)$ hosts. The drawback with their approach is that the system of linear equations are typically over-defined, and measurement noises make the over-defined system unsolvable. To deal with noisy measurements, the authors use least-square approximation to solve the system of linear equations. We adopt the basic idea in this approach to building an interface-level network map without explicitly solving the system of linear equations due to the large number of links that need to be computed. We compute individual link latencies on every path independently. We then use a filtering technique discussed in the next section when multiple values exist for a single link latency.

The current approach to generating host-level connectivity maps is for one or more measurement hosts on the Internet to actively probe and discover routes to other hosts. After probing thousands of hosts on the Internet, one ends up with a collection of paths that potentially has a large number of shared routers, or more precisely router interfaces, among the many paths. The paths can then be joined together at the shared router interfaces to form a connected topology. There are also approaches that yield latency information directly without connectivity. The common practice to collecting end-to-end latencies is to use `ping` or time the three-way handshake of TCP connection initiation [20]. The advantage of this approach is its low cost for a single measurement, but the disadvantage is that it is a point-to-point measurement. We would not obtain any connectivity information from these point-to-point measurements so we could not easily compute latencies between hosts not conducting measurements on a network map. There is a triangulation technique [21] that can approximate latencies among hosts lacking connectivity information. In [19], we used a set of `traceroute` gateways [22] to probe a set of destination hosts. To estimate the end-to-end latency between two destination hosts, we chose the `traceroute` gateway through which the triangulated latency was the minimum. This minimum triangulated latency became the estimated latency for the given pair of hosts. However, studies [23, 24] have shown that Internet paths often are not the shortest in latency. Since it is difficult to evaluate the quality of the triangulated latencies without conducting measurements at each client host, we do not use the triangulation technique.

3 Incorporating Link Latency Information

In this section, we will first give an overview of the basic operations involved in computing link latencies. We present a detailed description of how we construct the network map while incorporating latency information. There are a number of challenges to accurately computing link latencies. We will discuss these challenges and some of our solutions.

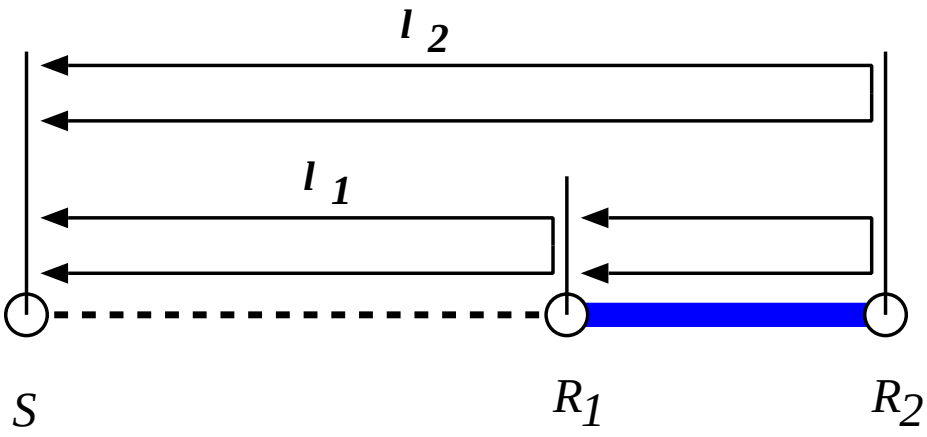

Figure 2: Compute link latency based on `traceroute` data.

3.1 Computing Individual Link Latency

We used a previously collected set of `traceroute` data, from a number of `traceroute` gateways to clients extracted from log files of five web sites. The detailed description of the data collection procedure can be found in [1]. There are a total of 2,306,560 individual `traceroute` runs in our data collection. Approximately 2% of them are unusable due to outages in the `traceroute` gateways.

Based on latencies between `traceroute` gateways and intermediate router interfaces, we compute link latencies and build the interface-level network map. For each link, we first obtain a filtered RTT sample (discussed in Section 3.3.2) between the source `traceroute` gateway and each of the two router interfaces connected by the link. We then subtract the two RTT samples to obtain the link latency. The link latency here is actually a round-trip latency since we subtract two RTT samples. For simplicity, we will refer to it simply as link latency. Figure 2 illustrates this procedure with two adjacent router interfaces, R_1 and R_2, on a path from S. We can compute the link latency between R_1 and R_2 by subtracting latency l_1 from l_2. We call this the hop-by-hop subtraction. Based on hop-by-hop subtraction, we can compute latencies for all links that appear in the `traceroute` data.

3.2 Constructing A Network Map with Latencies

The most important step in our construction is getting accurate latency information between source `traceroute` gateway and intermediate router interfaces. In this section, we first describe how we obtain these latencies and then give a detailed step-by-step procedure to building an interface-level network map.

Our initial attempt to perform hop-by-hop subtraction is to subtract RTT samples of adjacent router interfaces in each `traceroute` run and filter link latencies if there are multiple samples for each link latency. The main drawback is that RTT samples from a single `traceroute` (at most three) are generally very noisy. Subtracting these noisy samples will likely produce large margins of error in latency computation. A better approach is to collect all RTT samples for each source-destination pair and filter the samples to obtain a single value. We filter RTT samples for all source-destination pairs and then perform hop-by-hop subtraction of these filtered values. We evaluated the accuracy of latency computation for both approaches with a small set of data, and the results indicated that the latter approach worked better. Based on our initial evaluation, we design the map-construction procedure as follows:

1. We organize `traceroute` outputs according to the individual source `traceroute` gateway. For each source `traceroute` gateway, we generate two sets of data. The first set contains RTT samples

between the source `traceroute` gateway and all intermediate router interfaces and all destinations. Step 2 describes the generation of this data set. The second set contains intermediate router interfaces on the path from the `traceroute` gateway to each destination end-host. Step 3 describes the generation of the second data set.

2. There are usually a number of RTT sample between the `traceroute` gateway and most of the router interfaces. We apply our filtering procedure to each set of RTT samples to obtain a single value. We discover that a router can sometimes be reached by the source `traceroute` gateway via a number of different paths. Therefore, RTT samples between the same pair of `traceroute` gateway and destination must be further identified by the exact path to which they belong. Ideally, the complete path discovered by `traceroute` can be used to identify the set of RTT samples from a source to a destination; however, the storage and computation cost would be overwhelming if we were to do it for every source-destination pair. We observe that most paths branch out around the fourth hop, likely because it takes approximately four hops to reach transit backbone networks. We use the fourth-hop router interface on every path to identify RTT samples on that path. We discard a small number (fewer than .1%) of `traceroute` runs that are missing the fourth-hop router interfaces.

3. In order to perform hop-by-hop subtraction, we also need the exact path traversed from each source to each destination in the `traceroute` data. When a `traceroute` run has responses from all intermediate routers, we simply record the IP address of each router interface encountered. However, many `traceroute` runs are missing one or more router interfaces. Our first solution is to skip the missing interfaces by establishing a virtual link between the last known interface and the next available interface. However, this would create a semantic inconsistency in what we mean by link latency since some link latencies would be virtual link latencies. To have a consistent notion of link latency, we decide to only allow real links in the network map. The cost of not establishing a virtual link is that parts of the network may be disconnected if there is only one outgoing path, and an intermediate router interface on that path did not respond to `traceroute`. Another problem that prevents us from recording IP addresses directly is that some router interfaces discovered have private IP addresses [25]. These private IP addresses can not simply be added to the path list since many ISPs allocate addresses from the same ranges of private IP address, e.g., the `10.255.255.255` range. We omit private IP addresses and connect together two router interfaces that come immediately before and after the private IP addresses. We treat the case of private IP address differently from missing router interfaces because private IP addresses should not be "leaked" to the real Internet, i.e., real Internet hosts should not communicate with non-local private IP addresses. To the rest of the Internet, two router interfaces that come before and after a router interface using private IP address are directly connected.

4. The two sets of data give both latency and path information from each `traceroute` gateway to all IP addresses reached by the `traceroute` gateway. We compute link latencies by doing hop-by-hop subtraction of latencies of paths originated at each `traceroute` gateway. From the collection of link latencies for each `traceroute` gateway, we extract the connected components with at least 100 IP addresses to form the individual `traceroute`-gateway map. We then combine the individual `traceroute`-gateway maps into an aggregate network map. There may be multiple samples for the same link latency from different `traceroute` gateways. We apply the same filtering technique again to obtain a single link latency. Finally, we extract the largest connected component from this aggregate map as the interface-level network map.

There are many imperfections in our construction of the network map, such as the filtering technique, the identification of RTT samples using the fourth-hop router, etc. In the next section, we discuss some of the challenges we face with our approach.

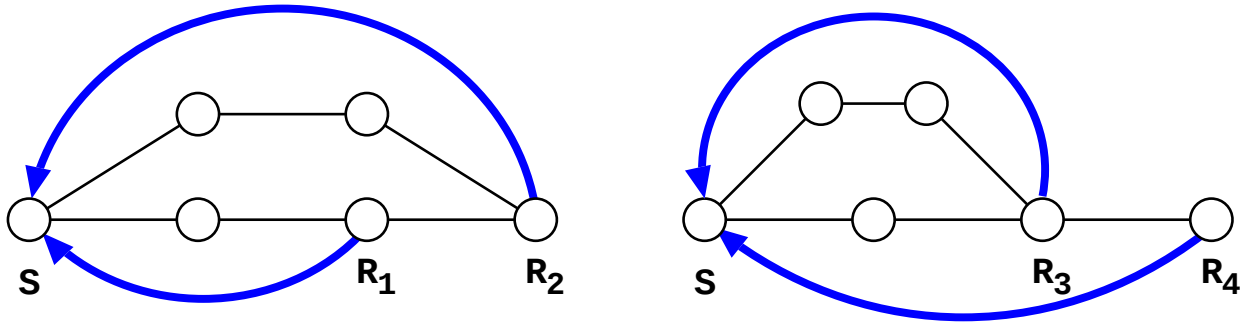

Figure 3: Effect of path asymmetry on link-latency computation.

3.3 Challenges in Building A Latency Map

There are three main challenges to building an accurate network map with latency information: asymmetry of Internet paths in the forward and reverse directions, RTT measurement noise, and instability of Internet routes and network topologies. The first two challenges are relevant to network maps with latency information. The last one is also relevant to connectivity maps. Here we discuss each challenge in detail and our solution to lessen its impact on the accuracy of network maps.

3.3.1 Path Asymmetry

As Figure 2 illustrates, the hop-by-hop subtraction implicitly assumes that paths from the source to the two router interfaces share all but the link connecting the two interfaces for both the forward and reverse directions. This assumption would be correct if Internet routing is completely symmetric. However, the author in [15] found that from 25% to 80% of Internet paths in their data sets, collected between 1994 and 1995, were asymmetric depending on where these paths originated in the Internet. Furthermore, their data sets showed that there was a trend of increasing path asymmetry. Therefore, it is important to understand how routing asymmetry affects the accuracy of constructed network maps.

Under stable routing, forward paths from a source to two consecutive router interfaces (en-route to a destination) differ only at the latter router interface. In Figure 2, the path S–R_1 is part of the path S–R_2 since both are on the longer path from source S to the same destination. Even if forward and reverse paths are asymmetric, as long as the two reverse paths are identical except for the latter router interface R_2, our approach will compute the correct link latency. When the two reverse paths are different, our link-latency computation will be incorrect. Figure 3 shows a scenario of routing asymmetry, where the two reverse paths are different. For clarity, we only indicate reverse paths in the figure. In Figure 3, the reverse path R_2–S has a larger latency (represented by the longer arch) than that of R_1–S. Since the latency of R_2–S would also be larger than that of R_1–S even when paths are symmetric, we can not determine whether path asymmetry exists. However, when the reverse path R_3–S has a larger latency than R_4–S in Figure 3, we will be able to recognize the existence of path asymmetry. When path asymmetry is detected, we will not attempt to calculate the link latency R_3–R_4 and remove the RTT samples between S and R_3.

In general, path asymmetry can cause both under-estimation and over-estimation depending on the difference in latency between two reverse paths. It is also very difficult to assess the impact of path asymmetry on link-latency computation without knowledge of paths in both directions. In this study, we assume path symmetry as long as we observe consistent latencies along a path, i.e., when interfaces with higher hop-counts always return longer RTT sample values than any of the interfaces with lower hop-counts on the same path. We will only calculate link latencies for those interfaces that return consistent RTT samples.

3.3.2 RTT Measurement Noise

The second challenge comes from the high variability of RTT measurements. An RTT measurement has three components: propagation delay, router processing delay, and queueing delay. The propagation delay component is constant as long as there is no path change, and the router processing delay component is the sum of per packet fixed processing delay at each router. The queueing delay component may be large and fluctuating rapidly when congestion occurs, which introduces the high variability. We want to capture the minimum latency, i.e., the sum of propagation and processing delays, and use that as the latency between two router interfaces.

From a set of RTT samples between a `traceroute` gateway and a router interface, we could use the minimum RTT as the minimum latency. However, the presence of spurious data in `traceroute` makes the use of minimum RTT error prone. We often find multiple responses mingled on a single line, which could be due to route changes in the middle of a `traceroute` run. We also notice that `traceroute` gateways often combine streams from both `stdin` and `stderr` such that numeric values in error messages in `stderr` could be interpreted as RTT values during parsing. Creating special cases to handle all such spurious instances during parsing would be too time consuming to be practical. Instead, we develop a filtering technique to avoid using outliers in RTT samples as latencies since outliers are likely to be spurious data. We filter each set of RTT samples by examining its distribution. If all RTT samples differ from the minimum RTT by a small margin (< 10 ms), then we take the minimum as the latency. Otherwise, starting with the minimum RTT sample, we find the first neighborhood of RTT samples of sufficiently large size, one-third of total samples, but at least 8 samples. We use the minimum in this neighborhood as the latency. If we are unable to do either, we find the minimum mode and use the minimum value in the neighborhood around the mode. When locating the mode, we ensure that the neighborhood around the mode contains at least 5 samples. If we are not able find a mode, we will use the median of the RTT distribution. The technique and parameters used in filtering are based on our own observations and repeated refinements after checking computed latencies in the network map against the actual latencies in the `traceroute` data.

However, our filtering technique can not always remove variable delays in RTT samples because of router configurations or traffic patterns in the Internet. One component of variable delay is the delayed processing of ICMP packets at routers themselves [26, 27]. This extra delay does not show up in regular packet forwarding so it is not part of any minimum latency that we are interested. Another component of variable delay is due to persistent congestion at a router. If congestion is present at all times, we will not be able to capture the minimum latencies of those router interfaces affected by congestion. The computed latencies from RTT samples under congestion will over-estimate minimum latencies. The over-estimation of latencies can cause both under-estimation and over-estimation of link latencies depending on where the variable delay is. We believe the errors on link latencies introduced by RTT measurement noise is not significant for backbone links because there are usually many RTT samples. For links in the stub networks, there are usually fewer samples except for those close to the `traceroute` gateways. Thus, filtered latencies are not as reliable, which translates into less accurate link latencies for stub networks.

3.3.3 Topological Instability

Routing instability has been a frequent event since the earlier days of the Internet. The analysis in [15] showed that only 68% of the routes were stable on the order of days based on data collected from 1994 to 1995. The analysis also showed that most of the shorter-lived routes were in backbone networks rather than stub networks. A more recent study [18] confirms the continuing existence of route oscillations in backbone networks. Considering that the duration of our data collection is 30 days, our `traceroute` data almost certainly contains many instances of routing instability. Furthermore, because our link-computation is highly path sensitive, routing instability can severely impact the accuracy of link latencies in our map.

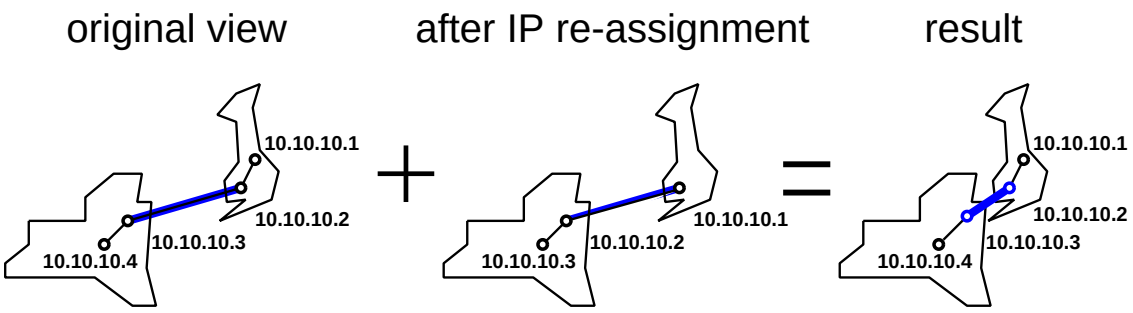

Figure 4: Effect of IP address re-assignment on link-latencies.

There are two consequences of routing instability on our network map. One consequence of routing instability is that our `traceroute` data contains links and paths that are never simultaneously present in the Internet, e.g., primary links and backup links are generally not used simultaneously. The addition of the backup links to the network map may create back-door paths among some hosts, thus forming paths much shorter in latency than the actual paths. Due to these backup links, a network map constructed using our approach will probably have richer connectivity and likely shorter end-to-end latencies than in the real Internet. Another consequence of routing instability is the introduction of different RTT values between a source and an intermediate router interface. Using a backup route that is typically longer than the primary route can result in a longer path, both in latency and in hop-count, to router interfaces that have been reached via the primary route before routing failure. These revisited interfaces would appear twice with different hop-counts in the `traceroute` output. We remove all cases involving router failures along with routing loops by implementing a loop/failure detection heuristic. The heuristic examines all interfaces traversed between a source and a destination and removes all instances where the same router interface appears at multiple non-consecutive hops. However, we allow the same router interfaces to appear in multiple consecutive hops. Some gateway routers forward all incoming packets from another domain to itself at a different interface for local delivery so multiple hops may use the same IP address [6] to reply to the source of `traceroute`.

One source of topological instability comes from networks that reconfigure their IP addresses or switch to a different service provider. As a result, different physical links may appear to be a single link. In Figure 4, we show a hypothetical example of how network reconfiguration causes bad latency estimates. In the original view of the network, we have four interfaces, `10.10.10.1` to `10.10.10.4`, located in England and the state of New York. Soon after the original snapshot, the ISP decides to reconfigure the network and moves one router IP address that was in England across the Atlantic Ocean to the state of New York. The latency of the trans-atlantic link remains the same, but the link changes from `10.10.10.2`–`10.10.10.3` to `10.10.10.1`–`10.10.10.2`. Taking the minimum of the RTT samples for `10.10.10.1`–`10.10.10.2` causes the latency estimate for the present trans-atlantic link to have the latency of the much shorter link with the same IP addresses in the original view. Worse yet, the original trans-atlantic link `10.10.10.2`–`10.10.10.3` will have the latency of the new link, which is also much shorter. The net result is that three very short links connect IP addresses in two different continents. We observe instances similar to the above scenario where an entire block of addresses were moved to a different geographic location. We have yet to devise a solution to cope with relocation of IP addresses and will address this in the future refinement of the construction procedure.

4 Routing Models

The interface-level network map generated has both connectivity information and link latency information. In network simulations, there is often a need to know the end-to-end latencies between host pairs in a network. To compute such end-to-end latencies, we need a model of network routing. We will examine simple models based on Dijkstra's shortest-path algorithm as well as a number of policy-routing models proposed by previous works [28, 7, 29] to compute end-to-end latencies. Authors in [7] used shortest-path routing in AS hop-count as an approximation to policy routing on the Internet. In a later paper [29], the (same) authors established a more accurate model of policy routing that took into account the observations on peering relationship and AS degrees reported in [28]. Below, we describe these routing models in detail.

4.1 One-Level Routing Models

Under one-level model, we assume a flat routing hierarchy, which is not used in today's Internet. We study the flat model to see whether this simple model can generate accurate end-to-end network metrics in our network map. We assume a one-level routing protocol that finds the shortest path between two hosts in the network. We use Dijkstra's shortest path algorithm [30] to compute three types of shortest paths: shortest paths in latency, router-level hop-count, and AS-level hop-count. Dijkstra's algorithm based on hop-count may find multiple paths with the same number of hop-counts between two IP addresses. In this case, we select the path with the minimum latency. As the final tie-breaker, we select the path with the smaller IP address for the first router interface that differs between two paths. We use router-level hop-counts as tier-breakers for Dijkstra's shortest-path algorithm in latency.

4.2 Two-Level Routing Models

A two-level routing model is more realistic in the sense that it separates routing on the AS level from routing inside a single AS. The top-level routing is done on an AS topology, and then Dijkstra's shortest-path algorithm in hop-count is run for the bottom-level routing inside each AS. Since our network map is an interface-level map, we need to construct an AS overlay on top of our map in order to run various two-level routing models. We adopt the approach used in [7, 29] to compute an AS overlay. We obtain an AS connectivity map on December 1, 2001, which is during the data collection time, from NLANR (`http://moat.nlanr.net/AS/`), which contains 8,259 unique ASs. To establish the connection between the AS overlay and our network map, we need to project each IP address in the network map onto the AS connectivity map. We map each IP address into the AS that advertises the network address that has the longest common prefix as the IP address. We map all IP addresses into 2,469 of the 8,259 unique ASs. We will give details about the AS overlay in Section 5.1.

We also incorporate the modeling of AS peering relationship studied in [28]. The policy-routing model in [28] infers relationships such as customer-provider or peer-to-peer based on AS degrees. This model only accepts paths that do not violate peering relationships. For example, it will not allow an AS path connecting two providers through one of their shared customers. Likewise, we also classify AS relationships into two categories: customer-provider and peer-to-peer. We make the same assumption as in [28] with regard to how AS degree affects the relationship. We classify the AS relationship between two ASs to be customer-provider when there is a "significant" difference between the two AS degrees. The AS having the larger degree is the provider. For pairs of ASs with similar degrees, we classify the relationship as peer-to-peer. We use two parameters $\{m, T\}$ to capture AS relationships. We assume a provider AS will always have a degree that is at least m ($multiplier$) times that of its customer, and we further assume that a provider must have a degree of at least T ($Threshold$). We will omit T when we do not place the requirement on AS degree in our evaluation.

To model the AS-level policy routing, we adopt the procedure used in [29]. We assume that the path must follow a sequence of $[\{c-p\}|\{p-p\}]*[\{p-c\}|\{p-p\}]*$. We abuse the standard regular expression in order to write the model for policy routing in a compact form. The notations $\{c-p\}$, $\{p-c\}$, and $\{p-p\}$ represent customer-provider, provider-customer, and peer-to-peer relationship, respectively. This sequence states that a valid path is one that starts with zero or more customer-provider or peer-to-peer links, followed by zero or more provider-customer or peer-to-peer links. Under this model, an invalid AS path is one where a provider-customer link is followed by zero or more peer-to-peer links, and then followed by a customer-provider link. In our path computation, we don't explicitly forbid such a path since doing so often eliminates all paths connecting two interfaces due to limitations in our policy routing model; instead, we assign a large AS hop-count to the link that violates the AS path hierarchy, i.e. the customer-provider up link that appears after a provider-customer down link. Thus, when our policy routing model fails, there would still be a path, which has the fewest violations, connecting two interfaces. We examine the percentage of path violations, for $m = 2$, the percentage is between 10% and 15% for most `traceroute` gateways. The percentage drops to around 5% for $m = 10$.

We use Dijkstra's shortest-path algorithm as the basic algorithm for all two-level routing models. To compute the path between two IP addresses in our map, we first compute an AS path that traverses the fewest AS hops. Inside each AS, we always select the path with the fewest interface-level hop-counts. If more than one path with the fewest AS hops exists, we then select the path with the fewest interface-level hops, and then those with smallest latencies if two or more paths are again identical. Finally, we will select the path with the smallest AS numbers.

5 Evaluation of the Network Map

We are able to construct an interface-level network map consisting of 85,007 IP addresses and 162,821 links with latency information. In this section, we evaluate our interface-level map and determine the accuracies of end-to-end hop-counts and latencies. There are two implications with evaluating the map. First, we want to validate our methodology of constructing network maps. Second, if we are able to find a routing model that reasonably duplicates end-to-end network metrics, we then essentially find a way to substitute computation for storage. Instead of storing all pairs of latencies or hop-counts among hosts in a network, we could simply store link latency and connectivity information and use the routing algorithm to compute reasonably accurate network metrics.

Below, we will first study the coverage of the Internet by our network map. We want to understand the limitation of our network map and therefore our evaluations. We then present evaluation results on computed network metrics. We will present the distributions of computed network metrics, the distributions of errors in computed network metrics, and the distributions of the percentages of correct pair-wise ordering of network metrics.

5.1 Map Coverage

Even though our study is not focused on obtaining the most complete network map of the Internet, we are still interested in learning how much of the Internet our network map actually covers. In terms of AS coverage, even though only 30% of ASs are mapped into our interface-level network map, all ASs (26) with degrees larger than 100 are covered by our network map; furthermore, only 64 of 366 ASs with degrees larger than 10 are not covered by our network map. We feel that the the majority of the more important ASs, i.e., those with large degrees are covered. However, AS coverage does not give a complete picture since many ASs contain tens of thousands of IP addresses. We also compare our network map against previously published network maps. The mapping effort reported in the SCAN project [12] claims that its network

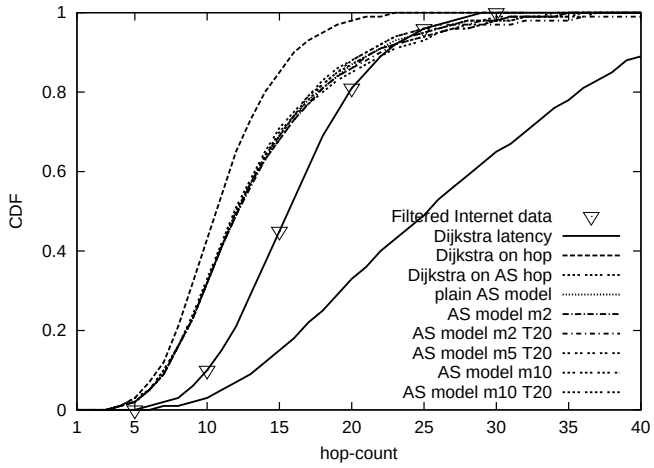

Figure 5: Hop-count distribution.

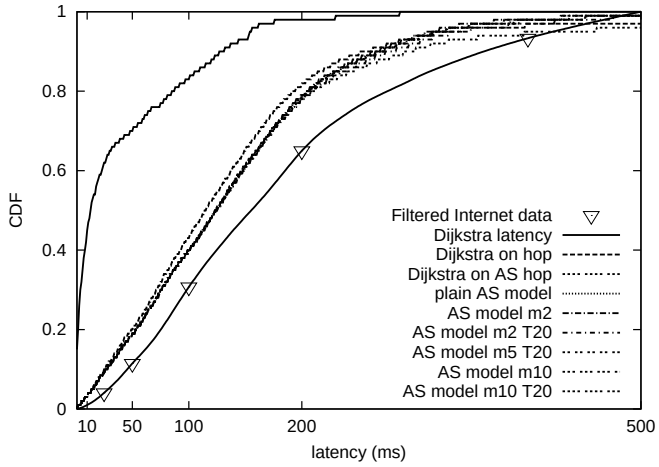

Figure 6: Latency distribution.

map captures the transit portion of the interface-level Internet topology. There are a total of 228,298 IP addresses in the SCAN map. Since our network map contains only 85,007 IP addresses, we expect our map would be a subset of the SCAN map. However, only 13,662 IP addresses are common to both maps, which means that only 16.1% of the IP addresses in our map are in the SCAN map. Given the large number of non-overlapping IP addresses, we feel that such discrepancy is not a result of different collection times. It is likely that both the SCAN map and our network map are incomplete at least in terms of end-host coverage.

In [31], the authors study the marginal utility of `traceroute` and conclude that, given a set of destinations, six or seven measurement hosts are enough to discover most of the IP addresses that would be discovered with a larger number of measurement hosts. Considering that we use 50 `traceroute` gateways as measurement hosts in our data collection, as described [1], we may have discovered all possible IP addresses associated with the destination set. We attribute the existence of the large number of IP addresses that have not been discovered to the limitation of our destination set, in terms of either the number or the distribution of IP addresses. The Rocketfuel project [13] has recently developed a number of heuristics that make intelligent selections of destinations to obtain complete maps of ISP networks. These destination selection heuristics will be incorporated into our future mapping efforts.

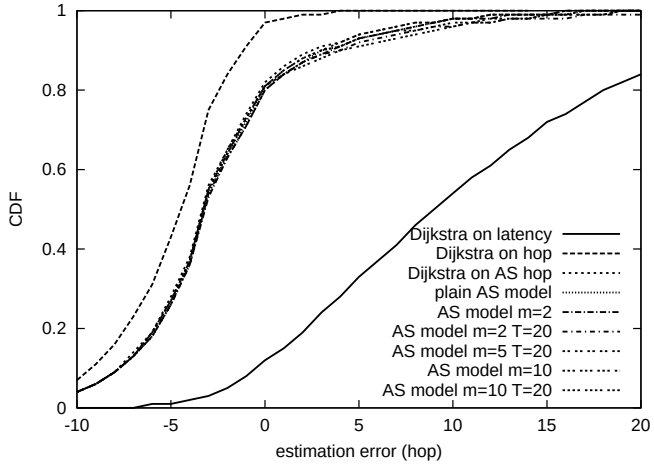

Figure 7: Hop-count error distribution.

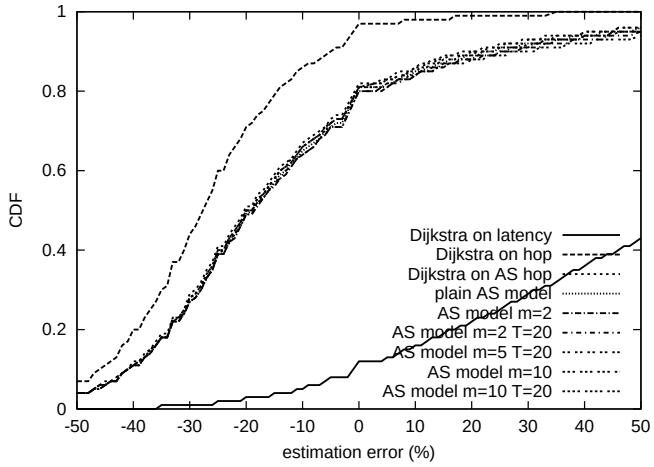

Figure 8: Hop-count percent error distribution.

5.2 Accuracy of Computed Network Metrics

We validate the interface-level network map by comparing the end-to-end metrics in the `traceroute` runs and those computed on the network map. The end-to-end metrics that we are able to compare are the latencies and hop-counts. Since we only have real hop-count and latency information for those interface pairs appearing in the `traceroute` output, we will only be able to compare the end-to-end metrics for these interface pairs. Even though we use the term "end-to-end," we will also compare the metrics between end-hosts and intermediate router interfaces since we want to evaluate as much of the map as possible, not just those involving end-hosts.

To evaluate latencies in the network map, we find all source-destination pairs in the network map that have corresponding matching pairs in the `traceroute` data. We declare two pairs a match if the corresponding interfaces in the two pairs have identical IP addresses. We are able to find a total of 3,029,584 matching pairs out of 3,626,233 in the parsed `traceroute` data. We are not able to match all pairs because some interfaces are disconnected from the network map due to missing intermediate router responses in the `traceroute` data.

Figure 5 and 6 show the distributions of hop-counts and latencies, respectively, under various routing models along with the actual distributions. The x-axis is either hop-count or latency in ms, and the y-axis

is the CDF of network metric. The curve labeled "plain AS model" is one where the customer-provider relationship is strictly enforced–the AS with larger degree is always the provider. All other curves are as indicated by their labels. In Figure 5, hop-counts computed by Dijkstra's shortest-path algorithm in latency are significantly larger than the actual hop-counts. In searching for the shortest latency paths, the Dijkstra's algorithm has to find "short-cuts" that are are the shortest in latency, but longer in hop-counts, than the actual paths. Hop-counts computed by Dijkstra's shortest-path algorithm in hop-count are shorter than hop-counts of the actual paths. However, the two distributions are qualitatively similar, the computed distribution is essentially the actual distribution shifted to the left by five hops. The distributions computed by the two-level models are almost identical and are in between the actual distribution and the distribution computed by the Dijkstra's shortest-hop-count algorithm. In Figure 6, as expected, the latencies computed by Dijkstra's shortest-path algorithm in latency are significantly smaller than the actual latencies. In addition, the computed distribution shows qualitative difference since a large percentage (70%) of the latencies are smaller than 50 ms compared to only 15% in case of the actual distribution. In Figure 6, all other routing models under-estimate latencies in the network map. The distributions under these routing models are similar to the actual latency distribution in the sense that there are significant number (20%) of large latencies (greater than 200 ms), and the number of latencies less than 50 ms is smaller than 20%. We observe that even though the two-level routing models use very different parameters, they all have very similar performance, which is consistent with the finding in [29] that there is little performance difference between a simple model of policy routing and more realistic models.

We also examine the errors between computed network metrics and actual network metrics. We will first discuss the errors in terms of hop count. We compare the computed hop-counts under each model with the actual hop-counts for each pair of matched IP addresses. We show the distributions of the actual errors in hop-count in Figure 7, and the distributions of percentage errors, computed by dividing errors in hop-count by actual hop-counts, in Figure 8. Dijkstra's shortest-path algorithm on latency has larger margins of errors both in terms of absolute error in hop-count and in terms of percentage error compared to the two-level routing models. We note that such errors are generally exclusively under-estimation errors. Even though our data was collected nine months later, our results are in good agreement with the findings in [7] where the inflation in router hop-counts by Internet policy routing is generally not more than ten hops compared to the shortest-hop-count paths as indicated in Figure 7. Furthermore, authors in [7] report that the inflation ratios of actual paths were generally less than two. In Figure 8, we find good agreement with prior result where only 5% of the paths computed using Dijkstra's shortest-path algorithm in hop-count are inflated by more than a factor of two.[1]

We examine latencies smaller than 200 ms separately from those that are smaller than five seconds. We make this separation to ensure that errors are not exceedingly large for smaller latency values. For each case, we examine in detail how the individual hop-counts and latencies match with the actual data. As we have indicated earlier, Dijkstra's shortest-path algorithm in latency produces many underestimated latencies. We note that almost 80% of computed paths have latencies smaller by at least 50 ms in Figure 9, and 90% of the computed latencies are smaller by more than 50% in Figure 11. Clearly, Dijkstra's shortest-path algorithm in latency is not a realistic routing model for network maps. The rest of the routing models produce very similar results with Dijkstra's shortest-path algorithm in hop-count slightly outperforming the rest. We note that, in the case of actual latencies smaller than 200 ms, nearly 40% of the computed latencies are within 20 ms of the actual latencies. There are a relative small percentage (less than 15%) of latencies that are off by more than 50 ms. We also represent the errors in terms of percentage in Figures 11 and 12. Again, we observe that the vast majority of latencies are within a factor of two of the actual latencies. Approximately 15% of the latencies are outside of this range.

[1] By a factor of two, we mean that the computed values are within -50% to 100% of the actual values.

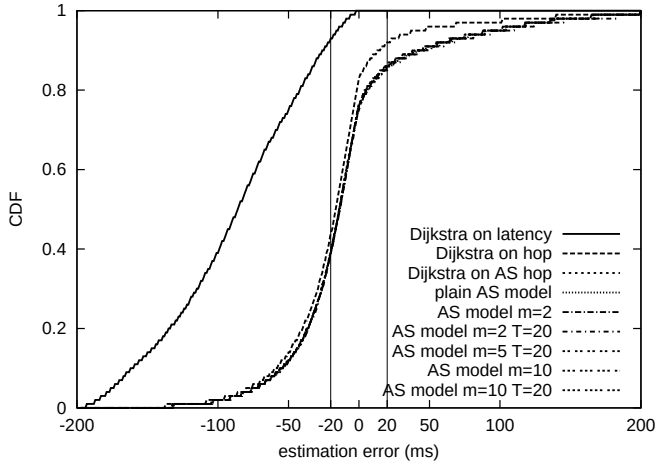

Figure 9: Latency error for latencies $\leq$ 200 ms.

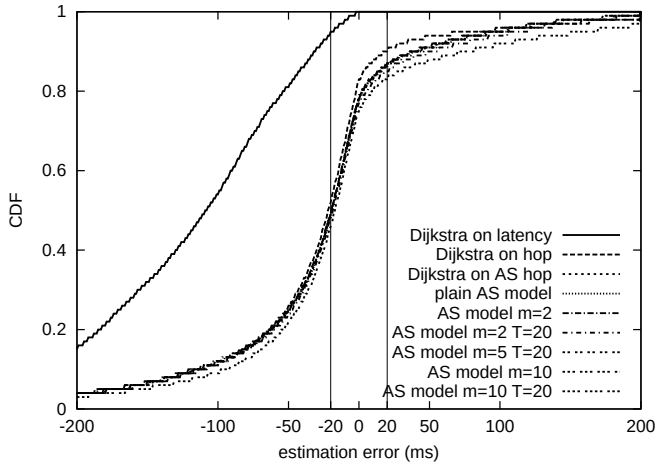

Figure 10: Latency error for latencies $\leq$ 5 s.

As a final evaluation of the network map, we want to test whether orderings of the network metrics are preserved through computation under various routing models. By this we mean, whether two network metrics have the same ordering in the actual data set as when they are computed under a particular routing model. For certain applications, such as the closest-server selection, knowing the correct ordering of latencies between a client and a set of servers will allow the client to incur the least access latency. We examine the ordering of hop-counts and latencies of paths rooted at each of the 49 `traceroute` gateways. For each `traceroute` gateway and every pairs of destination IP addresses reachable by the `traceroute` gateway, we examine the ordering of hop-counts and latencies from the `traceroute` gateway to the two IP addresses. The ordering under a particular routing model is correct if it matches the ordering of actual network metrics. We compute the percentage of correct orderings for each `traceroute` gateway and then plot the CDF of the 49 percentages. Figure 13 shows the CDF of correct ordering percentage for the 49 `traceroute` gateway. Dijkstra's shortest-path algorithm in latency again has the lowest percentage of correct orderings, while Dijkstra's shortest-path algorithm in hop-count has the highest percentage of correct orderings. All other routing models have nearly identical performances. We see under most routing models, the mean ordering accuracy is approximately 68%. This is a only small improvement from random ordering which will give a correct ordering half the time. Figure 14 shows the CDF for latency orderings.

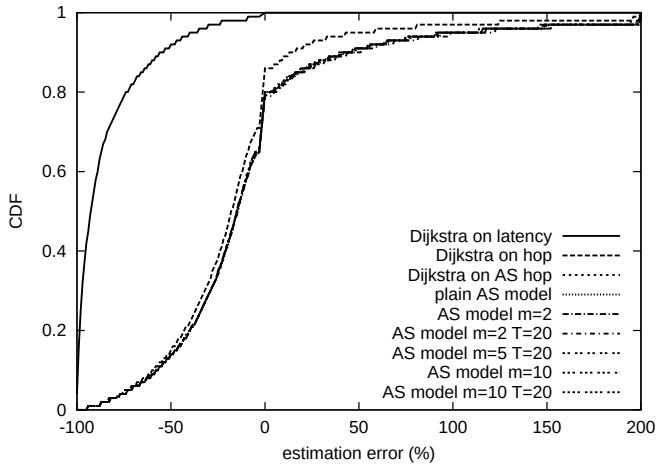

Figure 11: Percent error for latencies $\leq$ 200 ms.

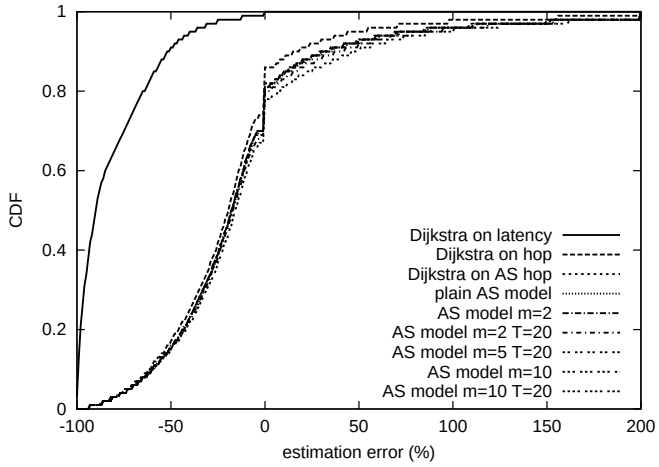

Figure 12: Percent error for latencies $\leq$ 5 s.

Again, we observe similar qualitative results with Dijkstra's shortest-path algorithm in latency having the worst performance. All other routing models are essentially equivalent with the one-level model in hop-count slightly outperforming two-level routing models. The mean ordering accuracy is around 78%, which is a larger improvement over random ordering. It is clear that having such a network map will improve the ordering of latencies and hop-counts over random ordering; however, further studies are needed to improve the ordering accuracy based on network maps.

From the evaluation results, we conclude that Dijkstra's shortest-path algorithm in latency is not an appropriate routing model for computing network metrics on interface-level network maps. All other routing models are essentially equivalent to each other. The simple routing model based on Dijkstra's shortest-path algorithm in hop-count generally provides as accurate network metrics as the more sophisticated two-level routing models. Despite the varying parameters, all of the two-level routing models have nearly identical performance in our evaluations. We again like to point out that this result is consistent with the finding in [29] where the authors conclude that a simple policy routing model produces results very similar to those by a more accurate model.

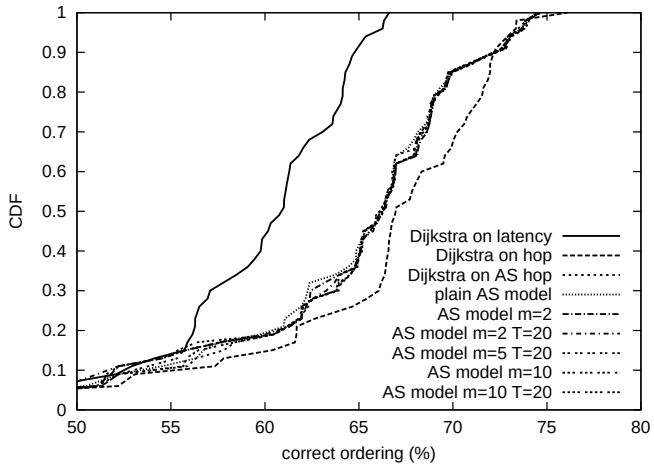

Figure 13: Ordering of hop-counts for paths rooted at each `traceroute` gateway.

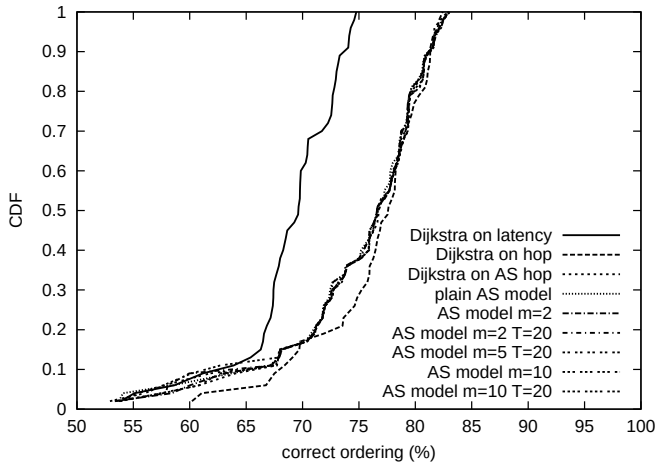

Figure 14: Ordering of latencies for paths rooted at each `traceroute` gateway.

6 Conclusion

Many simulation studies in network applications and services can benefit from accurate network maps that contain more than just connectivity information. In this paper, we describe a model for building a more comprehensive network map than one with just connectivity information. We demonstrate this concept by combining connectivity, latency, and routing information into an interface-level network map based on a large collection of `traceroute` data. We discuss the challenges of this approach and some solutions. We evaluate our interface-level map by comparing computed hop-counts and latencies against the `traceroute` data under a number of routing models. For the network maps we constructed, a one-level routing model based on shortest hop-count generates the most accurate network metrics of all the routing models studied.

We believe that it is possible to construct a reasonably accurate network map with both connectivity and latency information with `traceroute` data. We note that this is only our first step at building a more comprehensive network map. Future work includes improved routing models, mechanisms that identify routing instability and minimize its effects , and the modeling of dynamic events such as network congestion.

References

[1] E. Cronin, S. Jamin, C. Jin, T. Kurc, D. Raz, and Y. Shavitt, "Constrained mirror placement on the internet," *IEEE Journal on Selected Areas in Communications*, 2002.

[2] D. Song and A. Perrig, "Advanced and authenticated marking schemes for ip traceback," in *IEEE Infocom'01*, Apr. 2001.

[3] C. Faloutsos, P. Faloutsos, and M. Faloutsos, "On power-law relationships of the internet topology," *in Proc. of ACM SIGCOMM*, Sep. 1999.

[4] A. Medina and I. Matta, "Brite: A flexible generator of internet topologies," Tech. Rep. BU-CS-TR-2000-005, Boston University, Boston, MA, 2000.

[5] C. Jin, Q. Chen, and S. Jamin, "Inet: Internet topology generator," Tech. Rep. CSE-TR-433-00, University of Michigan, 2000.

[6] H. Chang, S. Jamin, and W. Willinger, "Inferring as-level internet topology from router-level path traces," Aug. 2001.

[7] H. Tangmunarunkit, R. Govindan, S. Shenker, and D. Estrin, "The impact of routing policy on internet paths," *in Proc. of IEEE INFOCOM 2001*, Apr. 2001.

[8] L. Qiu, V. Padmanabhan, and G. Voelker, "On the placement of web server replics," *in Proc. of IEEE INFOCOM*, Apr. 2001.

[9] K. Park and H. Lee, "On the effectiveness of router-based packet filtering for distributed dos attack prevention in power-law internets," *in Proc. of ACM SIGCOMM*, Sep. 2001.

[10] "NLANR Network and Measurement Analysis Group", ," University of California, San Diego. [Online]. Available: http://moat.nlanr.net/, 1997-1999.

[11] H. Burch and B. Cheswick, "Mapping the internet," *IEEE Computer*, vol. 32, no. 4, pp. 97–98, Apr. 1999.

[12] R. Govindan and H. Tangmunarunkit, ""Heuristics for Internet Map Discovery"," *in Proc. of IEEE INFOCOM 2000*, pp. 1371–1380, Mar. 2000.

[13] "Rocketfuel," University of Washington. [Online]. Available: http://www.cs.washington.edu/research/networking/topsim/.

[14] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP throughput: A simple model and its empiracl validation," *in Proc. of ACM SIGCOMM*, Sep. 1998.

[15] V. Paxson, "End-to-end routing behavior in the internet," *in Proc. of ACM SIGCOMM*, pp. 25 – 38, 1995.

[16] V. Paxson, "End-to-End Routing Behavior in the Internet," *Proc. of ACM SIGCOMM '96*, pp. 25–38, Aug. 1996.

[17] C. Labovitz, R. Malan, and F. Jahanian, "Origins of internet routing instability," *in Proc. of IEEE INFOCOM*, 1999.

[18] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian, "Delayed internet routing convergence," *in Proc. of ACM SIGCOMM*, 2000.

[19] Y. Shavitt, X. Sun, A. Wool, and B. Yener, "Computing the unmeasured: An algebraic approach to internet mapping," *in Proc. of IEEE INFOCOM*, Apr. 2001.

[20] W. R. Stevens, *TCP/IP Illustrated, Volume I*, Addison-Wesley, 1994.

[21] S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang, "On the placement of internet instrumentation," *Proc. of IEEE INFOCOM*, Mar. 2000.

[22] TRaceRT, "Multiple traceroute v0.96," [Online]. Available: http://www.tracert.com/cgi-bin/trace.pl, 2000.

[23] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang, "Idmaps: A global internet host distance estimation service," *ACM/IEEE Transactions on Networking*, Oct. 2001.

[24] S. Savage et al., "The End-to-End Effects of Internet Path Selection," *Proc. of ACM SIGCOMM '99*, Sep. 1999.

[25] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear, "Address allocation for private internets," RFC 1918, Internet Engineering Task Force, Feb. 1996.

[26] J. Nagle, "Congestion control in ip/tcp internetworks," RFC 896, IETF, Jan. 1984.

[27] F. Baker, "Requirements for ip version 4 routers," RFC 1812, IETF, Jun. 1995.

[28] L. Gao and J. Rexford, "Stable internet routing without global coordination," *Proc. ACM SIGMETRICS 2000*, Jun. 2000.

[29] H. Tangmunarunkit, R. Govindan, and S. Shenker, "Internet path inflation due to policy routing," *Proceeding of SPIE ITCom*, pp. 188–195, Aug. 2001.

[30] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms*, Cambridge, MA: MIT Press, 1990.

[31] P. Barford, A. Bestavros, J. Byers, and M. Crovella, "the marginal utility of network topology measurements," *SIGCOMM Internet Measurement Workshop'01*, 2001.