Path Stitching: Internet-Wide Path and Delay Estimation from Existing Measurements

DK Lee, Keon Jang, Changhyun Lee, Gianluca Iannaccone†, and Sue Moon
KAIST, †Intel Research Berkeley
{dklee, keonjang, chlee, sbmoon}@an.kaist.ac.kr, †gianluca.iannaccone@intel.com

***Abstract*—Many measurement systems have been proposed in recent years to shed light on the internal performance of the Internet. Their common goal is to allow distributed applications to improve end-user experience. A common hurdle they face is the need to deploy yet another measurement infrastructure. In this work, we demonstrate that *without any new* measurement infrastructure or active probing we obtain composite performance estimates from AS-by-AS segments and the estimates are as good as (or even better than) those from existing estimation methodologies that use on-demand, customized active probing.**

The main contribution of this paper is an estimation algorithm that breaks down measurement data into segments, identifies relevant segments efficiently, and, by carefully stitching segments together, produces delay and path estimates between any two end points. Fittingly, we call our algorithm *path stitching*. Our results show remarkably good accuracy: error in delay is below 20 ms in 80% of end-to-end paths.

I. Introduction

Internet-wide services and applications depend on accurate information about the internal network state to deliver good performance to end-users. For example, content distribution networks use delay information to direct clients to replicas that would provide best performance. Peer-to-peer VoIP systems (e.g., Skype) have been shown to deliver better voice quality when AS path information is taken into account for selecting peers [16]. However, today's Internet does not provide such information explicitly and developers resort to ad-hoc measurement tools to obtain the necessary data. This poses an additional tax on the development cost of new services and applications. For this reason, a number of systems have been recently proposed and implemented to provide a shared measurement infrastructure for distributed applications [3], [5], [10], [19]. They follow a common plan of action: (1) define estimation methodologies for delay, path, loss rates, etc.; (2) carefully construct an active probing strategy and instrument end-systems to collect measurements accordingly.

In this work we diverge from this tradition of active measurement. We are interested in the potential of estimation methodologies in isolation from data collection. Our goal is to demonstrate that *without any new* measurement infrastructure or active probing we obtain composite performance estimates from AS-by-AS segments and the estimates are as good as (or even better than) those from existing estimation methodologies that use on-demand, customized active probing. The key idea behind scaling measurements to the size of the Internet is to take advantage of the known underlying structure of the network. Existing approaches in line with ours are iPlane [10] and Akamai's core points [9]. They derive estimates by composing performance measures of network segments along the end-to-end path. Our approach differs from these two in that we construct end-to-end information from performance measures segmented *by the AS*. Let us illustrate our approach in the following simple example. Consider a query for some performance metric between two points x and y in the Internet. Assuming that we have access to segmented performance measures, we infer the AS path between the two points and construct the end-to-end metric corresponding to those ASes on the inferred path.

The main contribution of this paper is an estimation algorithm that breaks down measurement data into segments, identifies relevant segments efficiently, and by carefully stitching segments together, produces delay and path estimates between any two end points. Fittingly, we call our algorithm *path stitching*. In this work we use *path* and *round-trip delay* as measures of interest for validating our structural approach.

Our approach is based on the following assumptions: (1) AS-level path inference is accurate; (2) measurement data segmented by the AS is readily available; and (3) characteristics of end-to-end path retain temporal stability. We argue that these assumptions are reasonable and present surmountable challenges. AS path inference has been an active area of research [13]–[15] and published methodologies now report 90% accuracy in AS path inference. Numerous end-to-end measurement data sets are publicly available today [6], [10], [17] that we can utilize. Finally, with regard to the third assumption, routes from a single end-host to many destinations are known to be fairly stable despite Internet's inherent dynamic nature [12].

Our results show remarkably good accuracy: error in delay is below 20 ms in 80% of end-to-end paths. We also present a comparison with iPlane [10], where measurements are carefully designed from hundreds of vantage points to maximize accuracy. Path stitching show an accuracy similar or slightly better than iPlane without having to instrument any new measurement node.

The accuracy of our path stitching algorithm defines what is *already* achievable without instrumenting any additional measurement node. Our work is a step towards bringing a diverse set of measurements together to improve accuracy without additional active measurements. Path stitching represents a reference baseline valuable when it comes to understand the benefits (and the costs) of deploying a new measurement

infrastructure.

The rest of the paper is organized as follows. Section II describes data sets that we use throughout the paper, and Section III presents our path stitching algorithm. In the following Sections IV and V we describe how to deal with two types of complications: when there is no exact path and when there are too many valid paths. We evaluate path stitching in Section VI, and conclude the work in Section VII.

II. Data

We use Ark [6] traceroute outputs, and Routeviews [2] and RIPE [1] BGP routing tables. These datasets are among the largest data archives publicly available and hold constantly updated information about IP and AS-level topologies. Thus they provide a good starting point for our investigation into the feasibility of path stitching.

Ark project collects traceroutes from 18 monitors to every /24 routable prefix. From Ark, we use one round (cycle-20080407) of traceroute outputs taken from April 7th to 9th, 2008 (a total of approximately 14 million traceroute outputs). RouteViews and RIPE are two most widely used repositories of BGP routing table snapshots. We use BGP routing table snapshots from the same three-day-period as our Ark data.

To evaluate the accuracy of our estimation methodology, we take direct path and delay measurements between a set of hosts and compare them against the path stitching estimates. We ran traceroute 50 times a day between 184 PlanetLab (PL) nodes during the same period as the Ark data. We then split this data set in two smaller sets: `pl-easy` and `pl-hard`. The distinction between the two sets lies in the co-location of Ark monitors at the source AS. 462 pairs in the `pl-easy` set have the source node located in the same AS as an Ark monitor. The latter set `pl-hard` contains the remaining $10,077$ pairs whose source PL nodes are not in the same ASes as any Ark monitor. This set is useful to evaluate our methodology for uncharted (or partially measured) network regions[1].

III. Path Stitching

Our goal is to estimate end-to-end path and round-trip delay between any two hosts in the Internet without resorting to active probes but re-using existing network measurement data. In the design of the path stitching algorithm we are guided by two main objectives: (i) coverage: the algorithm must be able to answer even those queries about end systems that are not present in the existing measurement data sets; and (ii) accuracy: the estimate should be as close to the actual measurement as possible.

Given two IP addresses as input, the path stitching algorithm operates as follows:

Step 1: Map IP addresses to AS numbers.
We use the BGP routing tables to map an IP address to an AS number. The longest prefix match on the IP address returns the prefix and corresponding AS path. The last AS number is then taken as the origin AS for the host.

Step 2: Infer AS-level paths between ASes.
We follow KnownPath's methodology [15] to infer AS paths between two ASes. KnownPath exploits the AS paths already present in BGP tables and infers AS paths by extending these known paths.

Step 3: Stitch path segments along the inferred AS path.
Taking as input the inferred AS path from Step 2, we extract router-level path segments from the traceroute database and stitch them up along the inferred AS path. This step may result in no candidate paths or may lead to too many candidate paths. We discuss approximation methodologies for the former case in Section IV and preference rules for the latter in Section V.

Step 4: Return the best candidate paths and delays.
When Step 3 outputs stitched paths, the final step is to calculate round-trip delays along the paths and return them as query result. The results contains both the most recently measured round-trip delays as well as a distributions of all the measured delays along the path.

Each step above makes use of the Ark and BGP data. In order to handle potentially large data sets efficiently, we preprocess and convert them to a more easily manageable format, "path segments". We split traceroute outputs into intra- and inter-domain segments. The set of intra-domain segments of a AS A (indicated by $:A:$) cover all known paths between any ingress and egress points of the AS A (together with router-level latency information). The set of inter-domain segments between AS A and AS B (indicated by $A::B$) describe all inter-AS connections that appear in the Ark data set. These path segments represent the basic components that are later stitched by our algorithm.

For example, when we resolve a simple query from a host a to b. First, we map a to AS A and b to AS B (**Step 1**). We derive the AS-level path from a to b (**Step 2**), AB in this example. Then, among the intra-domain segments $:A:$, we search for those that start with a. Then we resolve the portion $A::B$, and find a segment that rendezvouses with the egress point of the $:A:$ segment. We continue to stitch up in a similar manner (**Step 3**). Actual queries are more complicated to answer than the above sample case. The following sections address the challenges in detail.

IV. Approximation in path stitching

Path stitching does not always return a stitched path. It fails when the path segment repository is missing data for the following three reasons: (i) the source or destination IP address maps to an AS that is not present in the repository; (ii) the inter-domain segment is not present in the repository; or (iii) the end of a path segment does not match any of the beginnings of the next AS path segments.

Data Type	Total AS	Transit AS	Stub AS
Ark	14378	4418	9960
BGP	28244	4847	23397

TABLE I
Number of ASes in Ark and BGP data

[1]The inter-domain connectivity of Planetlab nodes relies on research networks and may not be representative of the Internet [4]. However, the goal of this data set is to evaluate the accuracy of our path stitching. We leave the problem of representativeness for future work.

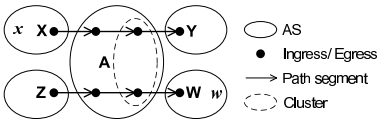


Fig. 1. Example of clustering

The first case of missing ASes has no solution other than collecting more measurements. Ark monitors probe every routable /24 prefix. Yet the Ark dataset is missing 50% of ASes present in the BGP tables (see Table I). A careful look at the data reveals that the Ark dataset covers 93% of the transit ASes and just 42% of the stub ASes[2]. The $13,866$ ASes not covered in the Ark data set correspond to approximately 110 million, or 5.8%, of the IP addresses that originate from all ASes in the BGP data: an order of magnitude smaller percentage than in the number of ASes. For those IP addresses, our path stitching cannot generate any estimate. For future work, we consider incorporating a second traceroute-based dataset (DIMES [17]) to increase the IP address coverage above the current 94.2% and AS coverage from 50.9%.

For the second case of missing inter-domain segments, we search for a reverse segment. That is, if we cannot find an inter-domain segment $A::B$, we consider the reverse path segment, $B::A$, instead, as a potential candidate. This is reasonable given that inter-domain segments are typically over point-to-point links and all links are bi-directional.

We address the third case by clustering IP addresses. Figure 1 illustrates our solution. In the example, the Ark data set contains path segments for AS paths, XAY and ZAW. When a query for IP addresses x and w arrives, the algorithm infers correctly an AS path XAW, but it is not able to find in the repository segments that can be stitched together. We then resort to approximation by clustering: the dotted circle in Figure 1 collapses two egress points into one. We employ multiple levels of clustering, first by the router (called IP aliasing), the Point-of-Presence (PoP), and the prefix length. Clustering at the router level has been addressed in previous work [18], and recent work by Madhyastha *et al.* offers router aliases as well as PoP clusters [10]. In our evaluation we use their router aliases and PoP clusters from March 2008. We extend approximation one step further than in [10] and allow clustering by the prefix when router or PoP clustering fails.

V. Preference rules

Now we turn our attention to the cases with *too many* stitched paths for a given query. Our goal in this section is to define the rules of preference and apply them to trim the list of candidate paths. A key insight in designing good preference rules is to reflect the actual mechanism that route packets through the network. We choose the following three preference rules, but omit supporting material due to space limitation. We refer to our technical report for detail [8].

A. Preference Rule #1: Proximity

IP addresses in the Internet far outnumbers ASes and there is no public data set that contains all the IP addresses. Thus end points of a query are likely to be not found in the path segment repository. Our first rule of preference addresses this problem by proximity. The proximity rule dictates that the path segments closest to the queried IP addresses are chosen for path stitching. The proximity is measured by the common prefix length[3]. The proximity rule improves accuracy in delay estimation, for delay within an AS varies widely.

B. Preference Rule #2: Destination-Bound Path Segments

Routing decisions in the Internet are made at every hop based on the destination. When we segment traceroute outputs by the AS and create the path segment repository, we keep this destination information with the segments and use it later in path stitching. Using only those segments with the same common destination prefix ("destination-bound segments") makes sense as it is consistent with how packets are routed in the Internet.

C. Preference Rule #3: Most Recent Path Segment

Even after applying the two preference rules described above, we may be left with more than one candidate path. We need a final preference rule. We decide to rank the candidate paths according to the time of the actual measurement (from most to least recent). Indeed, end-to-end routes can change at any moment in the network, thus the most recent segment is likely to represent the end-to-end route most accurately.

The path stitching algorithm first applies the proximity rule to reduce the initial path segments in the source AS. Then, it selects the most recent inter- and intra-domain path segments towards the same destination prefix as the query. Finally, it applies the proximity rule at the destination AS.

VI. Evaluation

In this section we demonstrate step-by-step how estimates from our path stitching fare in comparison to on-the-spot actual measurements. First, we evaluate the overall quality of AS path inference in **Step 2** of path stitching. Then we show how much value the approximation methods in Section IV and preference rules in Section V bring to **Step 3**. As a final part in evaluation, we compare our results against iPlane [10].

A. AS Path Accuracy

For every pair of hosts in `pl-easy` and `pl-hard`, we execute **Steps 1** and **2** of our algorithm and obtain *inferred AS paths*. For all pairs in `pl-easy` and `pl-hard` we also have the actual *measured AS paths*. Due to space constraints, we only summarize the result in this section; in `pl-easy` pairs, we see that 72% of inferred AS paths are exact matches of the measured AS paths, while in `pl-hard` pairs, only 26% of inferred AS paths are exact matches. For more detailed results, refer to our technical report [8].

B. Evaluation of Approximation Methods

The goal of the approximation methods in Section IV is to produce approximate stitched paths in the face of no stitched path. For both sets of `pl-easy` and `pl-hard` we stitch

[2]We call an AS transit if it appears in any AS path (not the first nor the last AS) in a BGP table; otherwise, stub.

[3]The size of the network prefix (e.g., /22) as a distance metric is an effective means of discriminating paths, as it allows strict ranking.

path segments along the measured AS paths and inferred AS paths. We compute the fraction of pairs that find stitched paths without any approximation. Then we apply approximation methods one by one. As predicted, we show incremental improvement in the fraction of pairs with stitched paths.

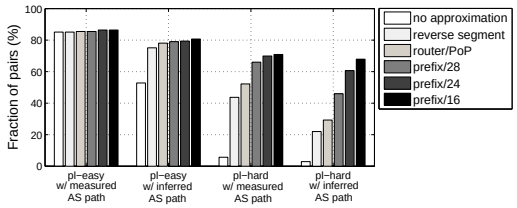


Fig. 2. Fraction of pairs with stitched paths

The fraction of pairs with stitched paths does not differ much between the measured and inferred AS path cases of `pl-easy`, because the Ark monitors are co-located with the source hosts of `pl-easy`. Yet for 13.6% or 63 pairs, there exists no stitched path. We have two explanations. For 59 or 12.8% of pairs, their AS paths include ASes or inter-AS links that are not present in the Ark data set. For the remaining 4 or 0.9% of pairs, the path segments cannot be stitched no matter what clustering we use.

The case of `pl-hard` is more complicated. When no approximation is used, only 6% of pairs find stitched paths with measured AS paths and even less with inferred AS paths. Considering the fact that no Ark monitor resides in the same AS as the source hosts in `pl-hard`, even those small numbers are surprising. Those originating ASes have appeared on some routes in the Ark data set. The largest increment in the fraction of pairs comes when we use reverse segments. Further relaxation on clustering constraints show definite incremental improvement. The Ark data set covers about half of the ASes observed in BGP. In order to bring the fraction of pairs with stitched paths from 70% with measured AS paths and 68% with inferred AS paths to the level of `pl-easy`, we conclude data sets with wider coverage of ASes are needed.

Additionally, we repeat the evaluation without router and PoP-level approximations. Resolving router aliases and clustering at the PoP-level require a large number of additional probes. Our interest here is to evaluate our algorithm when those datasets are not available. Indeed, the additional dataset bring limited benefit: clustering with /28 and /24 prefix and without router and PoP, we miss only 5 (0.04%) pairs for the `pl-hard` with measured AS paths case, and 165 (1.6%) pairs for the `pl-hard` with inferred AS paths case. For `pl-easy` cases, there are no missing pairs.

C. Evaluation of Preference Rules

In this section we demonstrate how each preference rule reduces the number of candidate path segments, as well as deviation in delay estimates from real measurements. If an inferred AS path includes a transit AS with a large number of intra-domain segments, it is not practical to consider all possible combinations of path segments[4]. To isolate the effect of preference rules from other factors in our evaluation, we only consider those host pairs in `pl-easy` and `pl-hard` that find stitched paths without any approximation method. We are left with 393 `pl-easy` pairs and 572 `pl-hard` pairs.

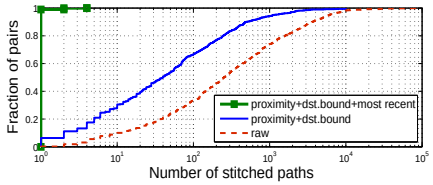
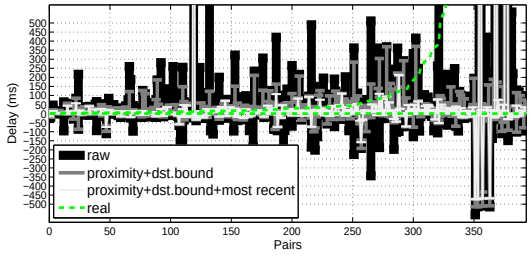
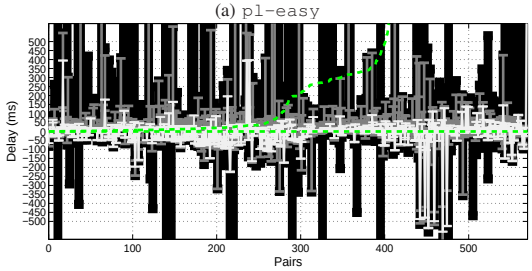


Fig. 3. CDF of no. of stitched paths of `planetlab`

In Figure 3 we draw the CDF of the number of stitched paths per host pair. The dotted red line marked 'raw' represents the total number of stitched paths before we apply any preference rule. We see that almost 60% of host pairs have 500 or more stitched paths. Now we apply the preference rules of proximity and destination-bound path segments and see the number of stitched paths decrease greatly. Still, about 30% of host pairs have about 100 or more stitched paths. Only when we use the segments of most recent measurement, we see the number of stitched paths drop to 1 for almost all pairs.



(a) `pl-easy`



(b) `pl-hard`

Fig. 4. Improvement in delay estimation

In Figure 4 we plot the minimum and maximum delay estimates of all stitched paths per query against actual measurements. For the ease of illustration, we peg the minimum of measured delays to 0 ms in Figure 4. We draw the difference between the maximum and minimum measured delay as a dashed line. It represents variability in actual measurements. Values that fall between the dashed line and the horizontal line of 0 ms delay are basically indistinguishable from real measurements.

We evaluate the following two combinations of preference rules: *(i)* only with the proximity and destination-bound prefixes and *(ii)* with the first two plus the most recently updated segments. The colors in the bar graph lightens as more preference rules apply. The lightening trend in colors indicates that our preference rules not only reduce the number of stitched paths, but also bring the estimates close to the

[4]In our path segment repository, the median number of segments of transit ASes is 25, but the maximum number reaches 124, 317.

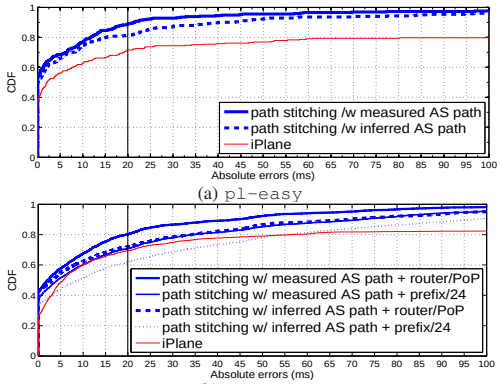
actual measurements.

D. Comparison with iPlane

Most latency prediction systems based on network coordinates require full-mesh type of measurements between participating nodes and are not amenable to take the Ark data set as input [3], [5]. In this section, we choose to compare our algorithm to iPlane [10] that uses a similar structural approach to latency prediction and is shown to provide more accurate results than other solutions [5], [12].

We obtained the binary code of the iPlane and router- and PoP-level clustering and IP-to-AS mapping collected during the same period as the Ark data set. We then fed our Ark data to the iPlane along with the additional information. By doing so, we effectively turn the Ark monitors to iPlane vantage points. We use `pl-easy` and `pl-hard` sets for the evaluation.

First, we examine the number of successful answers. Our numbers are 367 with /24 clustering and inferred AS paths in `pl-easy` and 6,103 with the same combination in `pl-hard`. With measured AS paths, we got 399 and 7,048, respectively (see Figure 2), while iPlane returns 325 in `pl-easy` and 5,109 in `pl-hard`. We find the numbers comparable and for the rest of the evaluation, we use only those pairs that both iPlane and our path stitching return answers.



(a) `pl-easy`

(b) `pl-hard`

Fig. 5. Absolute errors

In Figure 5 we plot the CDF of absolute errors. In the case of `pl-easy` (top graph), we only show results with /24 clustering because other approximation methods return almost the same results. Path stitching reports consistently smaller absolute errors. We note that the iPlane performance observed in Figure 5 is comparable to the best case reported in Figure 4 of [11]. As we have seen in Figure 2, a relatively large fraction of pairs are stitched with approximation methods in the case of `pl-hard`. Therefore, we draw separate graphs for pairs with different approximation methods. We draw a graph for the results with the router and PoP clustering and with the /24 prefix-level clustering. Overall, path stitching shows consistently better performance with very small absolute errors (below 5 ms.) Path stitching with router and PoP clustering performs very close to iPlane below 35 ms, and gradually shows better performance afterwards. Path stitching with inferred AS path and /24 clustering shows better performance than iPlane only after 50 ms. In both plots, the performance of iPlane does not improve much beyond 50 ms. As we do not have access to the source code, we cannot provide further explanation. One conjecture we have is iPlane does not incorporate any data filtering mechanisms as suggested in [7] or in [8], and anomalously large delays have an impact on the tail of the distribution.

VII. Conclusions

In this work, we have presented and evaluated *path stitching*, a new approach to end-to-end Internet performance estimation. Existing measurement systems are naturally limited by the number of available vantage points. Instead of deploying yet another measurement system, we have described how multiple data sets from existing infrastructures can be used together to provide relatively accurate performance measurements of uncharted portions of the Internet. We show that our path-stitching approach performs comparably to existing systems that require extensive new data collection campaigns.

Acknowledgements

We would like to thank Christian Lumezanu, David Choffnes, Geoffrey Voelker, Jongyul Kim, Ravi Sundaram, Rubén Cuevas Rumín, and the anonymous reviewers for their valuable feedback. We thank Harsha Madhyastha for sharing binary codes of the iPlane. We are also grateful for Minjae Hwang's help with data processing. This work is supported by the IT R&D program of MKE/IITA [2007-F-038-03, Fundamental Technologies for the Future Internet].

References

[1] "RIPE Routing Information Service," http://www.ripe.net/ris.
[2] "The RouteViews project," http://www.routeviews.org.
[3] S. Agarwal *et al.*, "Matchmaking for online games and other latency-sensitive P2P systems," in *ACM SIGCOMM*, Aug. 2009.
[4] S. Banerjee *et al.*, "The interdomain connectivity of planetlab nodes," in *PAM*, Antibes Juan-les-Pins, France, Apr. 2004.
[5] F. Dabek *et al.*, "Vivaldi: A decentralized network coordinate system," in *ACM SIGCOMM*, Sept. 2004.
[6] Y. Hyun *et al.*, "The CAIDA IPv4 routed /24 topology dataset - Apr'08," http://www.caida.org/data/active/ipv4_routed_24_topology_dataset.xml.
[7] J. Ledlie *et al.*, "Network coordinates in the wild," in *NSDI*, 2007.
[8] D. Lee *et al.*, "Path stitching: Scalable and systematic internet-wide path and delay estimation from existing measurements," Computer Science Department, KAIST, Tech. Rep. CS-TR-2009-313, July 2009.
[9] F. T. Leighton *et al.*, "Method for predicting file download time from mirrored data centers in a global computer network," May 2001.
[10] H. V. Madhyastha *et al.*, "iPlane: An information plane for distributed services," in *USENIX OSDI*, Nov. 2006.
[11] ——, "Structural approach to latency prediction," in *ACM SIGCOMM IMC*, Oct. 2006.
[12] ——, "iPlane nano: Path prediction for peer-to-peer applications," in *USENIX NSDI*, Apr. 2009.
[13] Z. M. Mao *et al.*, "On AS-level path inference," in *ACM SIGMETRICS*. Banff, Canada: ACM Press, June 2005.
[14] W. Muhlbauer *et al.*, "Building an as-topology model that captures route diversity," in *ACM SIGCOMM*, Sept. 2006.
[15] J. Qiu *et al.*, "AS path inference by exploiting known AS paths," in *IEEE GLOBECOM*, Nov. 2006.
[16] S. Ren *et al.*, "ASAP: an AS-Aware Peer-relay protocol for high quality VoIP," in *IEEE ICDCS*, Jul. 2006.
[17] Y. Shavitt *et al.*, "DIMES: Let the Internet measure itself," *SIGCOMM CCR*, vol. 35, no. 5, pp. 71–74, 2005.
[18] N. Spring *et al.*, "Measuring ISP topologies with Rocketfuel," in *IEEE/ACM TON*, 2004.
[19] B. Wong *et al.*, "Meridian: A light weight network location service without virtual coordinates," in *ACM SIGCOMM*, Aug. 2005.