RouteSeer: Topological Placement of Nodes in Service Overlays

Sridhar Srinivasan, Ellen Zegura
Networking and Telecommunications Group
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332
Email: {sridhar,ewz}@cc.gatech.edu

***Abstract*— Overlay networks are being increasingly used to deploy new services on the Internet. As opposed to peer-to-peer overlays, these infrastructure or service overlays offer the opportunity of placing the overlay nodes and selecting the links between them. There has been very little work done in the area of node placement in overlay network design. In this work, our objective is to study the overlay node placement problem based on a specific performance objective, namely, overlay link resiliency. An overlay link is called resilient if there exists an intermediate overlay node through which a connection can be established even if there is a failure in the underlying network links between the overlay nodes.**

In this paper, we propose an algorithm, called RouteSeer, to solve the overlay node placement problem. We split the problem into two parts, placing some overlay nodes called client proxies "close" to the clients of the overlay service and placing intermediate nodes to provide resilient paths between the client proxies. RouteSeer heuristically places the intermediate overlay nodes by only examining the routing tables at the client proxies and does not require global topology information. In our simulations and experiments on the Internet, we show that RouteSeer can improve on previous schemes by 50-100%.

I. Introduction

Overlays have emerged as the preferred way to provide new network services to users, as well as to deploy older services such as multicast that have proven difficult to provide in the network layer. An overlay network comprises overlay nodes that are responsible for routing and forwarding, connected by overlay links that correspond to paths in the underlying network. The client hosts of the overlay may also be the overlay nodes, forming a peer-to-peer overlay network used for services such as file sharing. Peer-to-peer networks suffer from well-known performance limitations due to the frequent high rate of churn of nodes. Alternatively, the client hosts may be distinct from the overlay nodes, allowing for infrastructure overlays made up of dedicated nodes that are managed by an overlay service provider. Our focus is on the design of these service overlays, hereinafter referred to simply as overlays.

By their nature, overlays are designed to support a particular service objective. Overlay network design includes the problems of node placement and selection of overlay links between nodes.[1] Where necessary or appropriate, the design may include on-line mechanisms to dynamically modify the links in response to changing network conditions or client behavior [1]. Nodes may also be added or removed, however that is likely to occur on fairly long time scales in a service overlay. The addition of a node may, for example, require contractual negotiations with an underlying network service provider and/or a co-location facility.

Much prior work has been devoted to the problem of link selection and network maintenance, with an assumption that the overlay nodes are given ([2], [3], [4], [5]). In contrast, we are interested in the node placement problem. Node placement clearly constrains the ability of the overlay to meet a particular performance objective, regardless of how well links are chosen. Further, we focus on the design objective of network resiliency, by which we informally mean the robustness of the overlay network to failures in underlying network links. These failures may be hard, i.e., links that go down, or soft, i.e., links that suffer periods of very poor performance. Our decision to focus on network resiliency derives from the following observations. Service overlays are compelling because they improve upon the service offered by the underlying native network. The ability to mask underlying network failures is a useful service in its own right, as well as the basis for allowing overlays to provide more sophisticated services. The increasing use of multi-homing [6] provides evidence that clients are willing to pay for increased resiliency.

Overlay network resiliency is achieved by designing the overlays such that connectivity between overlay nodes is not affected by the failure of underlying links. A simple method to achieve this is to ensure that different overlay links do not share common network paths. In such a scenario, if any network link fails, no more than one overlay link is affected and the overlay remains functional.

One approach to this problem involves active measurements between potential overlay node locations, with the objective of minimizing the number of shared network links between overlay links [7], [8]. Based on measurements, a set of overlay node locations are chosen. Naturally, these proposals require measurement overhead to identify node locations. More significantly, they require that the overlay network provider already have access to the full set of potential node locations, since this is needed to conduct the measurements.

[1] In contrast, peer-to-peer overlays have little or no control over the placement of the nodes.

In this work, we propose a technique called RouteSeer to select locations for overlay nodes that comprise these overlays. RouteSeer is based on examining routing tables at a few locations on the network and our technique can examine the entire address space, rather than a limited set of locations, for potential overlay node locations while requiring much less active probing of the network than previous proposals. The main contributions of this work are in proposing analytical methods for identifying potential overlay node locations.

We split the problem of overlay node placement into two parts: the first part is to place overlay nodes "close" to the clients. For this we use existing proposals for placing nodes or services in proximity to the clients [9], [10]. The second part of the problem is the placement of overlay nodes to such that overlay links are disjoint, i.e., do not share common networks paths. provide resilient paths between nodes. For this, we first examine the information provided by routing tables at the set of overlay nodes located "close" to clients of the service overlay. Under certain assumptions of ideal network routing, we show how this information can be used to select overlay node locations that minimize the sharing of network paths. We then extend this method to more realistic networks by relaxing our assumption of ideal network routing behavior and incorporating some active probing of the network. Our simulations indicate that we can achieve significant improvements (30-100%) in creating overlays with non-overlapping network paths using our technique.

The rest of the paper is organized as follows: In the next section we discuss the background of the overlay node placement problem and define the problem more formally. In Section III, we explain the working of RouteSeer. We evaluate the RouteSeer algorithm in Section IV using simulations on a wide variety of topologies, and conclude in Section VII.

II. Problem Formulation

Node placement and overlay construction are important problems that exist in varying degree in the design of all overlays. The location of the overlay nodes and the overlay connectivity can both be critical to the performance of the overlay. Consider a network such as the Resilient Overlay Network (RON) [11] in which one of the design goals is to recover from path outages. The RON network is quite sparse, i.e, composed of a small number of nodes. The placement of the overlay nodes in this network is crucial to the performance of the overlay, while the connectivity between the overlay nodes is not an issue as usually, all nodes are aware of each other. To illustrate the nature of the problem that poor placement of overlay nodes can cause, consider a small network as shown in Figure 1. In this figure, the overlay nodes are the solid circles which connect among themselves using overlay links shown by dashed paths. The node A is connected to the rest of the overlay network through link i. If this link fails, then node A is effectively cut off from the overlay network until the network layer recovery provides a path to the other overlay nodes through the other links leading out of A. If, on the other hand, one of the other overlay nodes,

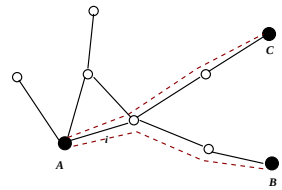

Fig. 1. An example of poor overlay node location

either B or C, was reachable through one of the remaining links (other than i), A would continue to remain connected to the overlay despite link i's failure.

Conversely, a peer-to-peer network, which is a special type of an overlay network, the number of participating nodes is large, i.e., the network is dense. In this scenario, the location of the individual nodes is not as important as the path that the packet that is being forwarded takes, in other words, the construction of the overlay is more important than the placement of the individual nodes. The large number of nodes participating in the overlay creates an inherent diversity of paths between the overlay nodes and hence diminishes the importance of node placement.

Based on these observations, we classify overlay networks based on the relative importance of node placement and overlay construction into two classes.

- **Sparse overlay networks:** which are composed of a small number of nodes. The number of overlay hops traversed by packets from a source to a destination are usually small.
- **Dense overlay networks:** which have a large number of nodes participating. In these networks, the number of overlay hops from a source to a destination are variable.

The node placement problem is critical for the overlays that can be classified as Sparse overlays as the performance of the overlay network is sensitive to the placement of the nodes. Each client interacts with one or a few overlay nodes and this interaction dictates the performance that the client sees from the overlay. In case of the Dense overlay networks, the placement of the individual overlay nodes is not of much importance for the reasons outlined above. On the other hand, since a majority of a packet's time is spent traversing links between several overlay nodes. In this case it is more important to ensure that the overlay construction creates efficient overlay links. It must be noted that the placement of the overlay nodes affects the overlay links that can be created, but overlay node placement plays a smaller part of this than in the Sparse overlay networks.

Many of the proposed applications for overlay networks tend to be sparse and use a single or constant number of intermediate overlay nodes to achieve their performance [11], [7]. We also believe that solving this problem can provide insight into placing overlay nodes in Dense overlay networks. For these reason, our objective in this work to address the overlay node placement in Sparse overlay networks.

The Overlay Network Design problem can be stated as follows: Given an undirected graph $G = (V, E)$, a set of clients

$C \subset V$, find a set $O \subset V \setminus C$ of overlay nodes such that a performance metric m over all possible $i, j \in C$ and $k \in O$ is optimized. The metric m could be in terms of the latency between nodes, the stress of the links connecting the overlay nodes, etc.

If we consider resiliency of overlays to network link failures as our performance metric, clearly it is sufficient to ensure that the paths between the overlay nodes are disjoint. If we define the *direct path* between $i, j \in V$ as the shortest path between i and j and an *indirect path* between i and j using k as the direct path from i to k and the direct path from k to j for some $k \in V \setminus C$ [do we need a reference for this?], we say an overlay path between nodes i and j is resilient if there exists an indirect path through some node $k \in V \setminus C$ which shares no common network nodes with the direct path between i and j. Then the overall overlay resiliency can be defined as the total number of overlay paths that are resilient. Another useful metric of overlay resiliency we use is the total number of intermediate vertices that are common to the direct and indirect paths for each pair of overlay nodes. We will formally define the problems based on these overlay resiliency metrics next.

As we discuss in Section III, it is helpful to fix the locations of some of the overlay nodes based on the locations of the clients of the overlay service. Let C be the set of overlay nodes whose locations are fixed. We call these nodes *client proxies* as they effectively act as proxies for the clients that connect to them.

We can now formally define the problem statement in the following manner: Given an undirected graph $G = (V, E)$ and a set $C \subset V$ of client proxies. Define $S(i, j)$, $\forall i, j \in V$, to be the set of vertices that lie on the shortest path from i to j. Define $overlap(i, j, k) = |(S(i, j) \cap S(i, k)) \setminus \{i\}| + |(S(i, j) \cap S(k, j)) \setminus \{j\}|$ $\forall i, j \in C$ and $k \in V \setminus C$. Intuitively, overlap counts the number of common network vertices in the direct path between i and j and the indirect path using an overlay node k. The Minimum Overlap problem can then be stated as finding the set of overlay nodes of size n that minimizes the total overlap across all pairs of client proxies. More formally,

Minimum Overlap Problem 1: Select $O \subset V \setminus C$, $|O| = n$, such that

$$\sum_{i,j \in C, i \neq j} \min_{k \in O} overlap(i, j, k)$$

is minimum over all possible sets O of size n.

Note that the actual set of overlay nodes is union of the client proxies and the overlay nodes, $C \cup O$. The above formulation requires that we know the overlap for each possible element of O to compute the minimum possible overlap. This in practice could require knowledge of the paths between every node in the graph, which in turn requires complete knowledge of the graph topology and this is not possible in many cases. Therefore, we define the following problem which is similar to the Minimum Overlap problem but which in practice can be heuristically solved using only routing table information available locally at each client node, without knowledge of the entire graph topology.

Maximal Disjoint Path Problem 1: Let $I\{\}$ be an indicator function with $I\{a\} = 1$ if a is true and 0 otherwise. Select $O \subset V \setminus C$, $|O| = n$, such that

$$\sum_{i,j \in C, i \neq j} \min_{k \in O} I\{overlap(i, j, k) > 0\}$$

is minimum over all possible sets O.

The Maximal Disjoint Path problem tries to find a set of overlay nodes of size n that maximizes the number of client proxy pairs that have a resilient overlay path.

Until now, we have been using the term "node" in a generic sense as vertices in a network graph. The actual overlay nodes are individual machines connected to the Internet and their placement in the network is the problem we are trying to address in this work. The placement problem is actually in two parts, deciding which Autonomous Systems (ASes) to locate the nodes in, and then to decide where in the AS to place the node. We will primarily focus on the first part, i.e., selecting the ASes in which to place the intermediate overlay nodes. Our heuristic can be easily extended to work in the intra-AS case as well.

For the remainder of this paper, we will assume that the network graph is the Autonomous System (AS) graph of the Internet and that the nodes represent ASes and the paths that we refer to represent AS paths. In the next section, we outline our heuristic, called RouteSeer which solves the Maximal Disjoint Path problem for the AS graph. We will point out the way in which the RouteSeer algorithm can be used for placement of overlay nodes in the intra-AS graph as well.

III. RouteSeer

As discussed in the previous section, solving the Minimum Overlap problem requires knowledge of the complete network topology which is usually not available. Active probing of all possible locations for the intermediate nodes has a prohibitive cost and is not practical in most scenarios. In this section, we describe RouteSeer, a technique to place overlay nodes for service overlay networks using routing table information. The RouteSeer algorithm is designed to solve the Maximal Disjoint Path problem outlined in the previous section rather than the Minimum Overlap problem because we limit our algorithm to use only information locally available to the nodes. We show in our evaluation that the RouteSeer algorithm gives good solutions to the Minimum Overlap problem as well.

A primary requirement for the RouteSeer algorithm is that the overlay nodes have multiple (> 1) egress links. It is clear that this condition is necessary to provide disjoint paths. At its core, RouteSeer uses the information in routing tables at the client nodes to make decisions about the placement of overlay nodes for resiliency. The intuition behind the RouteSeer algorithm is quite simple: if we consider the shortest path tree constructed with a particular node as the source, once a particular outgoing link is chosen for a path p_1 to a destination d_1, any shortest path p_2 to a destination d_2 using

a different outgoing link will not intersect the path p_1. In RouteSeer, the routing table information at the client nodes is used to construct an approximation[2] of the shortest path tree and the same idea is applied. We now describe the RouteSeer algorithm in detail.

A. The RouteSeer Algorithm

For any service overlay network to be useful to clients, the placement of the overlay nodes should reflect the locations of the clients, i.e., nodes providing service to the clients should be close to them. Client nodes can be transient in that they can connect to the overlay network for short periods of time. This behavior must be taken into consideration when placing the overlay nodes, which are assumed to be long-lived. For this reason, RouteSeer takes a three-step approach to placing overlay nodes in the network.

1) First step: Client Proxies: In the first step, we place special overlay nodes, called Client Proxies or CPs, "close" to the clients of the overlay network. Close could be in terms of the number of hops, latency or any metric that the overlay appliction requires. The main function of these client proxies is to act on behalf of the clients that connect to them and ensuring that the CPs are close in terms of the relevant metric improves the performance perceived by the clients. The advantage of using client proxies instead of the actual clients are that client proxies can also help in routing and forwarding of packets on behalf of other overlay nodes. Moreover, the use of client proxies frees the clients from running any routing protocol to discover the routes to destination nodes as the client proxies can perform this function for them. The client proxies are assumed to be long-lived and so the placement of the other overlay nodes can be better optimized for the set of client proxies and thus, for the clients as well. Using client proxies, the problem space can also be reduced from finding resilient paths between each pair of clients to finding resilient paths between each pair of client proxies. For the purposes of our heuristic, we assume that the client proxies have already been placed based on various methods such as those proposed by Krishnamurthy and Wang [9] or Barford et al [10].

2) Second Step: Get potential locations: RouteSeer, in the second step, attempts to place overlay nodes such that they can be used to provide path diversity for the paths between client proxies. We begin by describing a simplified version of RouteSeer for which we make the following two assumptions:

1) The network layer uses shortest path routing when forwarding packets.
2) Network paths are symmetric, i.e., the network path from a node i to a node j is the same as the path from j to i.

We will later relax these assumptions and explain the modifications required. Before we explain the main algorithm, we first explain a simple data structure that we use that we call a *reachability table*.

[2]Due to the use of policy-based routing, routes in the Internet do not necessarily follow the shortest path.

TABLE I
FORWARDING TABLE

Address Prefix	Link
1.0.0.0/16	a
2.0.0.0/16	b
3.0.0.0/16	c
4.0.0.0/16	c
5.0.0.0/16	a

TABLE II
REACHABILITY TABLE

Link	Address Prefixes
a	1.0.0.0/16
	5.0.0.0/16
b	2.0.0.0/16
c	3.0.0.0/16
	4.0.0.0/16

3) Reachability table: By our initial assumption, each proxy has several outgoing links on which traffic to the rest of the Internet is routed. The forwarding tables at each of these proxies are represented in the usual manner by tuples of the form <address prefix, next hop>as shown in Table I. Each next hop in the table corresponds to one of the network layer links of the node. We process the forwarding tables to generate a reachability table (shown in Table II) where the entries are of the form ¡next hop, address prefix¿. Each entry in the table contains the set of address prefixes that are reachable from that node using the particular link, e.g., in the figure, the address prefixes 1.0.0.0/16 and 5.0.0.0/16 can be reached from the node by using link a.

4) Main algorithm: Consider a pair of client proxies CP_i and CP_j as shown in Figure 2. We construct reachability tables for both nodes from the respective forwarding tables. Assume that traffic from CP_i destined for CP_j is forwarded along link b. This information can be found from the forwarding table for CP_i. If we want to achieve overlay path resiliency for the path from CP_i to CP_j, clearly the path to the overlay node that acts as a relay cannot be along link b, i.e., the intermediate overlay node cannot be in any of the address prefixes for which CP_i would use link b to forward the packets. Using the reachability table, we can eliminate all such address prefixes by removing the entry for link b. We call the set of address prefixes that remain V_{ij}. By our first assumption of shortest path routing, any path which begins with link b will not intersect any path beginning with a link other than b. Thus, we

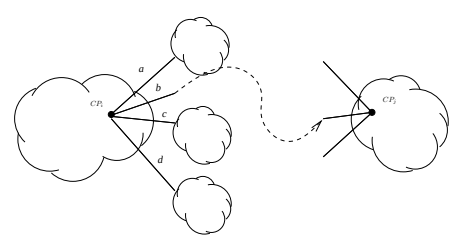

Fig. 2. Path between client proxies CP_i and CP_j

can be assured that if the overlay node is located in an address prefix which belongs to V_{ij}, the path from CP_i to the overlay node will not intersect the path from CP_i to CP_j. For duplex communication, we need to ensure that the reverse path from CP_j to CP_i should also not intersect the path from the overlay node to CP_i or the path from CP_i to the overlay node. By our second assumption of path symmetry, CP_i receives traffic from CP_j on the same link b on which it sends traffic to CP_j, and so the paths would be link-disjoint. Node CP_j can compute a corresponding valid set V_{ji} based on information in its forwarding tables in the same manner. The intersection of the sets V_{ij} and V_{ji}, $A_{ij} = V_{ij} \cap V_{ji}$, gives the set of address prefixes in which the overlay nodes that provide path resiliency for the path from CP_i to CP_j could be located. It is possible to have either A_{ij} or one of V_{ij} or V_{ji} be empty in which case the path between this pair of client proxies cannot have a resilient overlay path between them.

When considering the sets of address prefixes to create the intersection, we restrict the prefix length to be at most 24 bits. The primary reason for this is to limit the number of address prefixes to consider to a managable size. In a recent BGP routing table dump, the number of prefixes with length greater than 24 bits was approximately 1% which also indicates that this prefix length is a good number to choose. We ignore any prefixes longer than 24 bits in the intersection even if such prefixes exist. In case of address prefixes that overlap, we take the longest matching prefix that is in both sets to be the result of the intersection, e.g., if V_{ij} contains the prefix 192.168.0.0/16 and V_{ji} contains 192.168.0.0/20, then their intersection A_{ij} will contain 192.168.0.0/20 which is the longest matching prefix that matches both address prefixes.

Once the set A_{ij} is created for a pair of client proxies CP_i and CP_j, we store this information in a table called the Global Address Table (GAT). This table has a row for each possible address prefix and each row stores the set of CP pairs for which this prefix appear in the intersection set. For example, if the prefix 192.168.0.1/24 appears in the set A_{ij}, then we add the pair (i, j) to the set of pairs stored in the row corresponding to 192.168.0.1/24 in the GAT. Thus, this table tracks the number of client proxy pairs for which a particular address prefix is a potential overlay location. It is to be noted that this table also stores address prefixes at the granularity of 24 bit prefixes. If a particular address prefix spans multiple /24 address prefixes, all are incremented appropriately.

5) Third Step: Compute feasible locations: This procedure of creating the set A_{ij} and incrementing the GAT is performed for all possible pairs of client proxies i and j. At the end of this process, each entry in the GAT contains the number of paths for which the address prefix is a potential location of an overlay node. From the 2^{24} available address prefixes, we need to extract a set of K address prefixes that can provide path resiliency to all pairs of client proxies. This problem is exactly the Minimum Set Cover problem, with the set S of all pairs of client proxies which is to be covered. Each address prefix covers a subset of S (the set of paths stored in the corresponding row in the GAT) and our objective is to find a cover of S which is at most of size K. This problem is known to be NP-complete [12]. We use the greedy heuristic outlined in Figure 3 to extract a set of K address prefixes that cover the maximum number of client proxy pairs.

```
S_A = computeLocations(GAT, K)
S_A = ∅
Assign all address prefixes in GAT to S
while | S_A | ≤ K do
  Sort S according to counts in GAT.
  Assign max ranked address prefix from S to p
  S_A = S_A ∪ − p ″
  for all paths for which p is a potential
  location
    decrement all other address prefixes which
    have this path
  end
end
```

Fig. 3. Heuristic for computing K potential overlay node locations

The algorithm maintains a set of accepted address prefixes S_A which is initialized to null. The address prefixes from the GAT which are copied to S form the available address prefixes from which the next address prefix is to be selected. The algorithm works in the following manner: iteratively, the address prefixes in S are ranked in order of the number of client proxy paths for which the address prefixes are potential overlay node locations. The highest ranking address prefix p is removed from S and added to S_A. Since p is a potential location for a set of client proxy pairs which no longer have to be considered by the remaining address prefixes, all remaining address prefixes that also are potential locations for these pairs have their counts decremented. This process is repeated K times to extract K address prefixes.

This set of address prefixes returned by the algorithm may not provide path resiliency to all client proxies, but we impose this limit to reduce the number of potential locations that have to be examined to finally place the overlay nodes. By imposing this limit, some of the paths may not have overlay nodes that can provide path resiliency but by changing the value of K, this number can be minimized.

Note that the procedure considers all paths between the client proxies to be equal. The same algorithms can be run in scenarios where the paths have different weights. The only difference is that when the address prefixes in the GAT are being updated, instead of incrementing by one, the address prefixes can be updated using the weight of the path.

The final set of address prefixes returned by RouteSeer can be used to place the overlay nodes or they can be used as the starting point for further probing to incorporate other performance metrics in the overlay paths such as latency and bandwidth.

B. Effect of assumptions

The above discussion of RouteSeer has been in the context of the assumptions made in Section III-A.2. We now discuss the effect of removing those assumptions one-by-one. Firstly,

we relax the second assumption of symmetric paths as paths on the real Internet are known to be asymmetric.

If the path taken by traffic from CP_i to CP_j is different from the path taken by traffic from CP_j to CP_i, a problem arises in the second step of RouteSeer. When the link b and its associated address prefixes are eliminated from consideration for V_{ij}, it was under the assumption that the reverse traffic from CP_j also traverses the same link. Now that this is not likely to be true anymore, we need to add another mechanism to discover the link that the reverse traffic would use. To do this, both nodes CP_i and CP_j send traceroutes to each other. By monitoring the individual links, CP_i can discover the link through which the traffic from CP_j reaches it. Assuming that this is link c, we now eliminate both links b and c and their associated address prefixes from inclusion into V_{ij}. Similarly, V_{ji} would eliminate the links on which it sends traffic to CP_i as well as the link from which it receives traffic from CP_i. The remainder of the RouteSeer algorithm proceeds as before.

The first assumption made was that shortest path routing was used in the network. This assumption is violated in the Internet due to the use of policy routing. We note that with this assumption, the locations returned by RouteSeer ensure that the path resiliency provided is total, i.e., no intermediate links are shared between the direct and indirect paths. If this assumption is violated, it is likely that some of the overlay paths would share links with the direct path between client proxies. We evaluate the extent to which this happens in Section IV.

IV. Evaluation

In this section, we evaluate the performance of the RouteSeer algorithm on various topologies and discuss the results.

A. Methodology

All our simulations were performed using a simulator built with the help of libraries from the *p-sim* simulator [13]. The simulator takes as input a network topology and parameters for the number of overlay nodes required and the nummber of client proxies in the network. It then places the client proxies randomly in the network topology and computes the routing tables for each proxy using Dijkstra's shortest path algorithm. The simulator then proceeds to place the required overlay nodes in the network based on the specified placement scheme and evaluates the overlap that results.

For many of the experiments discussed below, we use AS network topologies generated from two sources. One source is the RouteViews project [14] which has BGP peering sessions with many ASes in the Internet. We call this the RouteViews topology in out experiments. The second source we use is the Internet Topology project [15] which aggregates BGP information from RouteViews, RIPE and Looking Glass servers to create a more complete topology than from a single source. We call this topology the IRL topology. Note that the graphs formed by these network topologies do not include any routing policy information except for the IRL topologies which marks ASes that offer transit and those that function purely as stub ASes. We use this information when running Dijkstra's Shortest Path algorithm by not allowing these stub ASes to appear in the middle of any shortest paths. We investigate the effects of policy routing in Section IV-C.

We evaluate the performance of the RouteSeer algorithm with respect to a random placement of the overlay nodes and to a placement scheme suggested by Han et al [7]. To implement this, we first randomly select a large pool of nodes that is at least twice the number of required overlay nodes. We then run the clustering-based heuristic proposed in [7] to create the required number of clusters. From each cluster, we select one node at random as the representative of that cluster. This gives us the required number of overlay nodes. It should be noted that the clustering heuristic can sometimes create clusters that are one larger than the number required, giving this placement scheme an advantage.

1) Parameters of the Simulations: Currently, there are few commercial service overlays deployed in the Internet and only a few research testbeds. It is difficult to predict the eventual size of these networks when they are deployed and so to test with any one specific value for the number of overlay nodes would be wrong. To get an estimate of how many overlay nodes we should consider, we looked at some examples of such networks. Research networks like the RON testbed have approximately 17 nodes, and the NLANR Web proxies also have a similar number deployed. On the other hand, Akamai, whose network of content servers could be considered to be a service overlay, has approximately 15K servers deployed. Clearly, with such a density of nodes, placement of these nodes for network resiliency is not of much importance as there is a high likelihood that an intermediate node exists for almost every path. It is unlikely that a new service network would be of this size. A more likely scenario for deployment would be a network like Planet-Lab, which is composed of approximately 400 nodes at various locations across the globe. Thus, in our evaluation, we look at networks ranging from 10 nodes to 500 nodes, which covers the range of possible uses of a service overlay.

A concurrent question is the number of intermediate nodes that need to be deployed. Note that the client proxies are also part of the overlay network and these intermediate nodes that are to be placed using RouteSeer can almost be considered to be "overhead", although it is highly likely that clients would connect directly to these nodes as well. Thus, it is reasonable to conclude that the number of intermediate nodes, depending on the performance that is desired, would be quite small. With that in mind, we perform our evaluations with the number of intermediate overlay nodes ranging from 2 intermediate nodes to 50 intermediate nodes for the larger networks.

B. Evaluation on the AS topology

In our first set of simulations, we compare performance of RouteSeer and the clustering heuristic over the AS topology of the IRL dataset. In Figures 4 to 6, we plot the performance of the three placement algorithms as we vary the number of intermediate nodes used from 2 to 50. For the 10 client

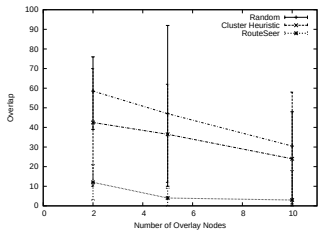

Fig. 4. Overlaps using 10 Client Proxies

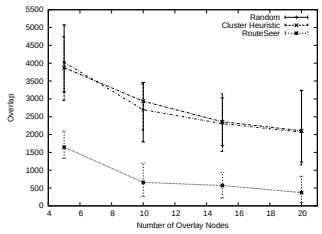

Fig. 5. Overlaps using 100 Client Proxies

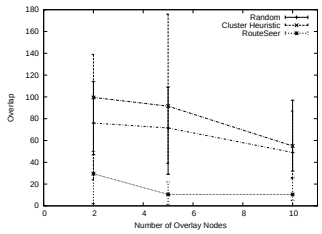

Fig. 8. Overlap using 10 Client Proxies with Policy Routing

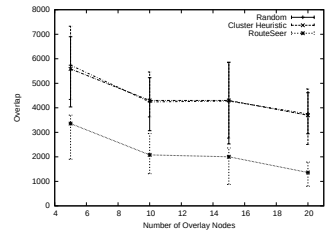

Fig. 9. Overlap using 100 Client Proxies with Policy Routing

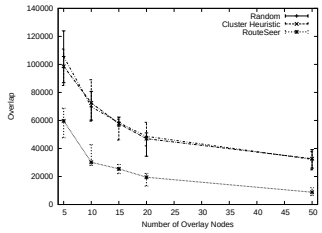

Fig. 6. Overlaps using 500 Client Proxies

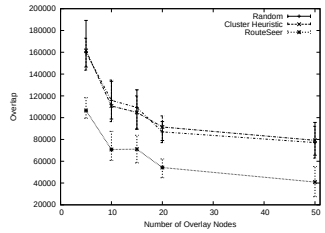

Fig. 10. Overlap using 500 Client Proxies with Policy Routing

proxies case, we restrict the number of intermediate nodes to a maximum of 10 nodes based on our prior discussion of the number of intermediate nodes.

We plot the number of intermediate nodes used on the x-axis and the number of overlapping nodes on the y-axis. The smaller the overlap, the better the performance of the particular set of intermediate nodes. For each value of the intermediate nodes, we repeat the simulation 10 times with different seeds, effectively trying 10 different sets of client proxies. We show the minimum, median and the maximum values of the 10 trials in the graphs plotted. From the figures, we see that the placement due to the RouteSeer algorithm is consistently better than the cluster heuristic. As more overlay nodes are used, the overlap for all placement schemes reduces, primarily due to the increased number of choices for each path that more intermediate nodes offer.

In fact, for a particular set of client proxies, the improvement is usually 50-100%. To show this more clearly, in Figure 7, we

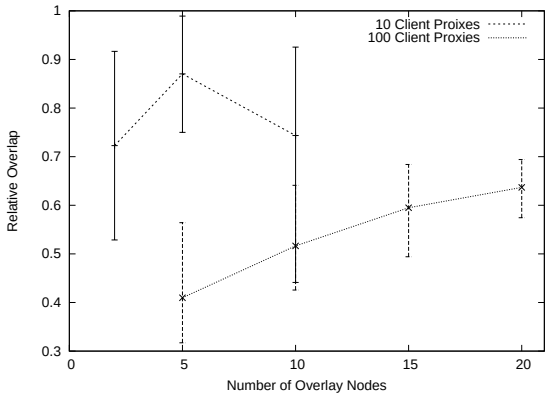

Fig. 7. Relative overlap of RouteSeer and Cluster heuristic using 100 Client Proxies

plot the relative difference between RouteSeer and the cluster heuristic as

$$(Overlap_{ClusterHeuristic} - Overlap_{RouteSeer})/Overlap_{ClusterHeuristic}$$

for the case of 10 and 100 client proxies using the same set of client proxies for both the RouteSeer placement and the Cluster heuristic placement. We compute the relative difference for each seed for a specific value of the overlay nodes and plot the median relative difference on the y-axis. The minimum and maximum relative differences observed are shown by the error bars for each data point. In case of the 100 client proxies the relative difference shows an increasing trend indicating that RouteSeer's performance relative to the Clustering heuristic improves as the number of overlay nodes increase even though the actual overlap reduces for both schemes. In case of the 10 client proxies, the trend is inconclusive, potentially due to the small number of overlay paths, but it is significant that even for this case, RouteSeer consistently performs better.

C. Effect of Policy Routing

An assumption made initially while describing the RouteSeer algorithm was that routes leaving a node through a particular link would not intersect routes leaving through other links, i.e., shortest path routing is being used. Routing on the Internet is affected by policies that individual ASes apply to routing packets through their domain [16] and this results in paths that do not necessarily follow the shortest path through the AS graph. To have a complete evaluation of RouteSeer, we need to quantify the effect of routing policies on the algorithm's operation.

There is a large body of work on understanding and measuring the effects of routing policies in the Internet. We attempt to recreate the effect that routing policies would have on routing on our simulation. For this, we use work done by Gao [17] on identifying AS relationships to mark the

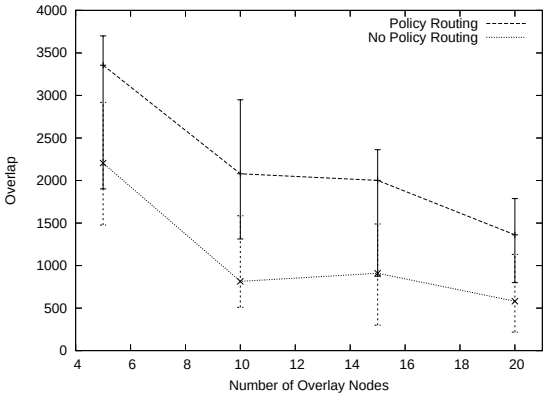

Fig. 11. Overlap using 100 Client Proxies with and without Policy Routing

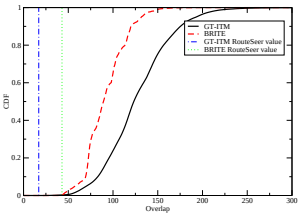

Fig. 12. CDF of Overlaps of representative BRITE and GT-ITM graphs

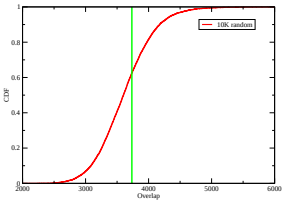

Fig. 13. Distribution of Overlaps of 10K random sets with 100 client proxies and 5 intermediate nodes.

edges between ASes as either customer-provider, provider-customer or peer-peer edges. We also use a simplified version of work by Subramanian et al [18] on modelling AS paths to define routes on this annotated AS graph. We only allow paths of the form [(customer-provider)*, (peer-peer), (provider-customer)*] where the '*' allows for multiple instances of a particular type of an edge. This simple model allows us to impose some basic policies on the paths in the AS graph. Note that this model does not cover all possible routing policies and must be viewed as a simple aprroximation.

Based on this model, we perform a set of experiments similar to those in Section IV-B. We use the RouteViews topology for this as this topology has AS relationship data readily available. We plot the results of running RouteSeer on this topology in Figures 8, 9 and 10. We observe that the RouteSeer algorithm continues to perform significantly better than the cluster heuristic or random placement. We see the same trends in these results as in the experiments with the IRL dataset.

We also performed some of these experiments without taking into account policy routing for the same topology and with the same seeds for the 100 client proxies experiment as in Figure 9. The results of these are plotted in Figure 11. We observe that our simplified policy routing model has increased the total path overlap. This increase is to be expected as the number of possible paths between pairs of nodes is reduced due to the removal of edges from consideration when computing shortest paths using policy routing.

D. Quality of RouteSeer solutions

In the previous sections we have seen that the RouteSeer algorithm performs better than the cluster heuristic, but the question remains as to how good is the solution returned by RouteSeer in absolute terms. In other words, how close is the RouteSeer solution to the optimal? Answering this question requires an exhaustive search of all possible solutions which is not practical for the large AS topology. We take two different approaches to this question. In the first, we generate synthetic topologies where we can practically run an exhaustive search for the optimal solution and compare the solution generated by RouteSeer to it. In the second, we randomly generate a large number of random solutions for the AS topologies and compare with the RouteSeer solution.

1) Synthetic Topologies: To evaluate the quality of the RouteSeer placements, we use a set of small topologies composed of approximately 250 nodes[3] in which we attempt to place 4 intermediate nodes for a set of 20 client proxies. We chose this particular set of parameters as it is possible to try all combinations quickly to find the optimal set of intermediate nodes in these topologies (the number of combinations is approximately 103 million). To create these topologies, we use two different types of topology generators, GT-ITM [19], a structural generator and BRITE [20], which we use as a degree-based topology generator [21]. Using two different types of generators allows us to test our algorithm under somewhat different circumstances as small topologies from a single generator may not match the all properties of the Internet. For our simulations, we generated several different topologies using each generator. For each topology, we computed all possible solutions and compared them with the results produced by RouteSeer.

We plot the results of one representative run of the GT-ITM topology and of the BRITE topology in Figure 12. In the figure, we plot the CDF of the possible solutions as a function of the overlap for each set. We observe that the placement selected by RouteSeer is very close to the optimal irrespective of the type of topology generator used to create the synthetic topologies.

2) AS topology: For evaluating the RouteSeer algorithm on a realistic AS topology, we use the same topology that was used in Section IV-B. We create 10,000 different sets of intermediate nodes of size 5 and evaluate the overlap of each of these sets. We plot the results as a CDF of the solutions as a function of the overlap in Figure 13. The vertical line indicates the overlap of the set generated using the Cluster Heuristic. The overlap of the set generated using RouteSeer is 1214 which lies outside the left of graph implying that the solution RouteSeer generated is significantly better than all the random solutions.

[3]The GT-ITM topologies used 245 nodes whereas the BRITE topologies used 250 nodes.

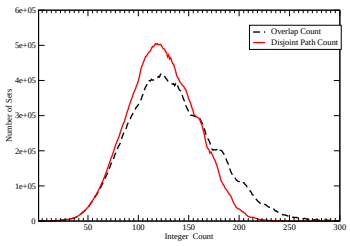

Fig. 14. Distribution of Overlaps of all possible sets of 200 nodes

E. Maximal Disjoint Path problem vs. Minimum Overlap problem

As discussed in Section II, the RouteSeer algorithm solves the Maximal Disjoint Path problem. Since the Minimum Overlap problem is similar to the problem that RouteSeer attempts to solve, it is useful to find out how well it performs in solving the Minimum Overlap problem as well. Solving the Minimum Overlap problem requires knowledge of the complete network topology which may not be available, while the RouteSeer algorithm uses only local information. Thus, if the RouteSeer algorithm provides good solutions for the Minimum Overlap problem, it could be used to solve this problem as well.

To evaluate the performance of RouteSeer on the Minimum Overlap problem, we use the same simulation setup as the previous experiment with the synthetic topologies. We compute the overlap of each set of intermediate nodes as well as the number of paths between client proxies that have non-zero path overlap (minimizing this is the same as finding a solution to the Maximal Disjoint Path problem). In Figure 14 we plot the histogram of the number of solutions on the y-axis that have a particular value of overlap on the x-axis as the solid line. We also plot the histogram of the number of solutions on the y-axis that have a particular number of paths with non-zero overlap on the x-axis using the dashed line. We observe that the solution provided by RouteSeer is close to the optimal for both problems indicating that the RouteSeer algorithm provides good solutions to both the Maximal Disjoint Path and Minimum Overlap problems.

F. Internet Experiments

We performed a small-scale experiment on the Internet using traces from the skitter project [22]. The data is composed of traceroutes performed from 20 different locations to a large number of destinations on the Internet (varying from 51K to over 900K destinations). We map the IP addresses in the traceroutes to their corresponding ASes using data on the prefixes advertised by each AS in the RouteViews BGP tables according to the procedure outlined in the skitter project. After performing this mapping and removing repeated ASes, we obtain the AS-level paths from each location to all the destination ASes. We use these AS-level paths to compute the AS routing tables of each source location. We then run the

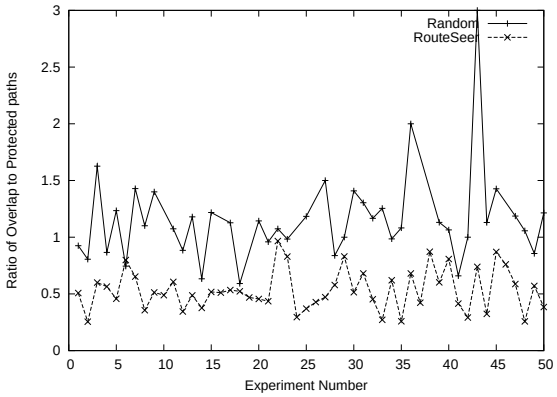

Fig. 15. Ratio of Overlap to Protected paths for the skitter locations

RouteSeer algorithm using the source locations as the client proxies. We use the generated routing tables to select the five overlay node locations which we evaluate using the same skitter traces. For the evaluation, we compute the AS overlap of the direct path between two locations and the indirect path using the overlay node which provides the minimum overlap.

We performed 50 different experiments by selecting a set of 15 locations at random out of the available 20 to act as the client proxies. For each set we compute the locations according to RouteSeer and also by picking five ASes at random. We evaluate the selection and plot the ratio of the overlap to the number of protected paths for overlay locations selected by RouteSeer and those selected at random in Figure 15. Since many of the traceroutes were incomplete, not all paths were available and hence, we report the number of paths that we tested and the overlap resulting from them rather than the total number of paths. From the figure, we see that RouteSeer performs significantly better than random in most cases. In only one instance was the Random selection slightly better than RouteSeer. On further investigation, we observed that this was primarily due to five locations in the experiment which have only a single link to the Internet were all present in the set used for this datapoint. Note that these locations violate the assumption that each client proxy have multiple access links. There are also a few instances where the number of protected paths using random placement is zero. This indicates that we could not find a valid traceroute to the that particular set of ASes selected. These sets have no points corresponding to Random placement in the figure (e.g., the 10th).

V. Discussion

The RouteSeer algorithm presented in this work uses the routing table information to create a snapshot of the local network topology in order to place the overlay nodes. We have assumed that the routing table is both static and complete, i.e., the entries do not change and the entire routing table for all client proxies is available. Clearly, the routing table of an AS can change due to many external factors such as route advertisements, failures and withdrawals. It can also change due to decisions made at the AS such as Intelligent

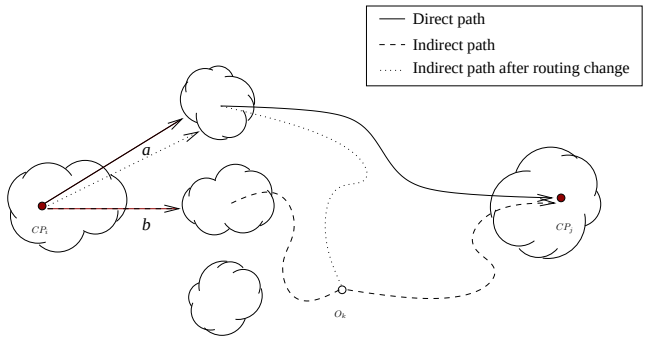

Fig. 16. A routing change removes protection of path between client proxies CP_i and CP_j through overlay node O_k

Route Control (IRC) for load balancing, pricing decisions and performance. The completeness of the routing tables available for use in RouteSeer is not guaranteed. The routing tables available may have entries for only a subset of ASes and some of these maybe wrong as well. We first discuss the challenges presented by changes in the routing table and later we point out some ways to work around incomplete routing tables.

A. Route Changes

The route changes that we are primarily concerned about are those that switch the path to an overlay node onto the same outgoing link as the link used to reach the destination client proxy that the overlay node provides resilience for. This scenario is shown in Figure 16 in which the path from CP_i to CP_j (shown by the solid line) is protected by the indirect path (shown by the dotted line) through overlay node O_k. A route change in CP_i's AS switches the network path to O_k onto the link a used by the direct path to CP_j. Such a route change negates the effect of the overlay node and leaves the direct path to the destination unprotected.

For our purposes, we call only those routing changes that remove protection from a protected direct path as failures. There are cases when a routing change does not affect the protection offered, for example, if the direct path was not protected or if there exists more than one indirect path and the routing change only affected one of them. We also do not consider the routing changes that cause the direct path to fail as the indirect path provides protection for this scenario.

Changes in routes due to external factors such as failures in the network cannot be controlled or predicted. For this reason, a proactive approach to dealing with these failures is better for minimizing service interruption. We envision one of two proactive approaches being taken for this: either by ensuring that each direct path is protected by two or more indirect paths that are themselves disjoint from each other, which may not be possible in many cases or by adopting a probabilistic approach described next.

In this approach, the RouteSeer algorithm is run after an "observation" period during which the probabilities of using different access links to reach a particular destination are measured. Once this period is over, when running the RouteSeer algorithm the address prefixes in Global Address Table (GAT) are marked according to the probabilities of using the particular access links to reach them rather than just being a potential location or not. When computing the feasible locations, the weight of each address prefix is computed by adding together the partial weights rather than the number of paths for which the address prefix is a potential overlay location.

As part of future work, we intend to evaluate these approaches to handling failures.

B. Incomplete Routing Information

A straightforward approach to dealing with partial routing information is to attempt to recover the missing information through an extensive series of probes. For example, to construct the AS level routing information of a particular location, a series of AS-level traceroutes can be performed to all routable ASes in the Internet. This set of ASes can be obtained by examining the routing tables available at repositories such as RouteViews. Once the set of traceroutes is collected, the AS-level routing tables can be constructed based on the number of next-hop ASes that appear in the traces. Clearly, this method requires time and resources to perform but the results can be reasonably accurate barring route changes that happen during the measurement period and the errors that appear in mapping IP addresses to their corresponding ASes. We intend to conduct a measurement study to quantify these errors and their effects on RouteSeer.

VI. Related Work

Recently, Han et al in [7] performed a measurement based study into the question of how to place overlay nodes to obtain overlay network resiliency in the face of network layer disruptions such as network node or link failures. The approach that the authors take is to try to obtain overlay path independence between different overlay paths such that a single network node or link failure would not disrupt more than one overlay path. The authors measure the direct path performance from a set of Planet-lab nodes to a set of web-sites and also to and from a set of looking-glass enabled routers. Based on these measurements, they compute the performance of the indirect path from a Planet-lab node to an intermediate looking-glass node and from that node to the destination Planet-lab node or website with respect to path independence and latency. The authors use this information to answer the questions: how many different ISPs should have overlay nodes in them and in each ISP how many overlay nodes should be placed to get path independence.

The approach outlined in this work is measurement based and is performed from a set of Planet-lab nodes which are used as the clients of this overlay service. The measurements for this technique require a substantial overhead in active probes and also requires access to all the possible intermediate overlay node locations. It is also not clear how the initial set of intermediate overlay locations can be selected in the general case.

Another closely related work [8] also examines the same problem using a graph-theoretic approach. The problem the authors consider is that of placing relay nodes in ISPs to ensure path diversity between origin-destination pairs. They define a measure of penalty between a direct path from source to destination and an indirect path through a relay node to indicate the number of links that are present in both paths. Their objective is to reduce this penalty over all source-destination pairs. For this, they propose two heuristics based on incrementally adding nodes to the set of relay nodes. This work assumes that the network topology is available to compute the penalty for the potential relay nodes.

VII. Conclusions

In this work, we have focused our attention on the node placement in infrastructure or service overlays. We studied this problem using overlay network resiliency as the performance objective and proposed an algorithm, RouteSeer to solve this problem. The RouteSeer algorithm splits the node placement problem into two parts. In the first part, it places nodes called client proxies "close" to the clients of these infrastructure overlays using solutions proposed in the literature. In the second part, it uses the local routing tables available at the client proxies to decide the locations of the intermediate overlay nodes that provide indirect paths which do not overlap with the direct network path between the client proxies. Using only local information ensures that we do not have to depend on possibly incorrect global topology data which may not be available in many instances. We showed in our experiments that RouteSeer can perform significantly better than existing methods for selecting overlay node locations.

References

[1] J. Fan and M. Ammar, "Dynamic topology reconfiguration of overlay networks: Structure and approximation of optimal policies," in *Infocom 2006*, Apr 2006.

[2] Z. Duan, Z. Zhang, and T. Hou, "Service overlay networks: Slas, qos and bandwidth provisioning," in *ICNP 2002*, Nov 2002.

[3] R. Braynard, D. Kostic, A. Rodriguez, J. Chase, , and A. Vahdat, "Opus: an overlay peer utility service," in *Proceedings of the 5th International Conference on Open Architectures and Network Programming (OPENARCH)*, June 2002.

[4] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel, "Almi: An application level multicast infrastructure," in *USITS 2001*, Mar 2001.

[5] S. Banerjee, C. Kommareddy, K. Kar, S. Bhattacharjee, and S. Khuller, "Construction of an efficient overlay multicast infrastructure for real-time applications," in *Infocom 03*, April 2003.

[6] A. Akella, B. Maggs, S. Seshan, A. Shaikh, and R. Sitaraman, "A measurement-based analysis of multihoming," in *ACM SIGCOMM 2003*, Aug 2003.

[7] J. Han, D. Watson, and F. Jahanian, "Topology aware overlay networks," in *Infocom 2005*, Mar 2005.

[8] M. Cha, S. Moon, C. Park, , and A. Shaikh, "Positioning relay nodes in isp networks," in *IEEE INFOCOM (Poster Session) 2005*, Mar 2005.

[9] B. Krishnamurthy and J. Wang, "On network-aware clustering of web clients," in *ACM Sigcomm*, Aug 2000.

[10] P. Barford, J. Cai, and J. Gast, "Cache placement methods based on client demand clustering," in *Infocom 2002*, Jun 2002.

[11] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris, "Resilient overlay networks," in *Proc. 18th ACM SOSP*, Oct 2001.

[12] M. R. Garey and D. S. Johnson, *Computers and Intractability*. W. H. Freeman, 1979.

[13] S. Merugu, S. Srinivasan, and E. Zegura, "p-sim: A simulator for peer-to-peer networks," in *Proceedings of IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, Oct 2003.

[14] R. Project, "http://www.routeviews.org."

[15] B. Zhang, R. Liu, D. Massey, and L. Zhang, "Collecting the internet as-level topology," in *ACM SIGCOMM Computer Communication ReviewVolume 35 , Issue 1*, Jan 2005.

[16] H. Tangmunarunkit, R. Govindan, S. Shenker, and D. Estrin, "The impact of routing policy on internet paths," in *Infocom 2001*, April 2001.

[17] L. Gao, "On inferring autonomous system relationships in the internet," in *IEEE/ACM ToN*, Dec 2001.

[18] L. Subrmanian, S. Agarwal, J. Rexford, and R. H.Katz, "Characterizing the internet hierarchy from multiple vantage points," in *Infocom 2002*, Jun 2002.

[19] K. Calvert, E. Zegura, and S. Bhattacharjee, "How to model an internetwork," in *Infocomm*, 1996.

[20] A. Medina, A. Lakhina, I. Matta, , and J. Byers, "Brite: An approach to universal topology generation," in *In Proceedings of the International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems- MASCOTS '01*, Aug 2001.

[21] H. Tangmunarunkit, R. Govindan, S. Jamin, S. Shenker, and W. Willinger, "Network topology generators: Degree-based vs. structural," in *Proceedings of ACM SIGCOMM*, Aug 2002.

[22] "http://www.caida.org/tools/measurement/skitter/."