Optimizing IP Address Assignment on Network Topologies

Flux Technical Note 2005-04

Jonathon Duerig* Robert Ricci* John Byers† Jay Lepreau*
*University of Utah {duerig,ricci,lepreau}@cs.utah.edu
†Boston University byers@cs.bu.edu

***Abstract*— We consider the problem of optimizing the assignment of IP addresses to nodes in a network. An effective assignment takes into account the natural hierarchy present in the network and assigns addresses in such a way as to minimize the sizes of routing tables on the nodes. Optimized IP address assignment benefits simulators and emulators, where scale precludes manual assignment, large routing tables can limit network size, and realism can matter. It benefits enterprise networks, where large routing tables can overburden the legacy routers frequently found in such networks.**

We formalize the problem that we are considering and point to several of the practical considerations that distinguish our problem from related theoretical work. We then outline several of the algorithmic directions we have explored, some based on previous graph partitioning work and others based on our own methods. A key underpinning of our methods is a concept we call Routing Equivalence Sets (RES), which provides a metric that quantifies the extent to which routes to sets of destinations can be aggregated. We present a comparative assessment of the strengths and weaknesses of our methods on a variety of real and automatically generated Internet topologies.

I. Introduction

Minimizing the size of routing tables on network hosts and routers is a basic problem in networking. A routing table must store a routing table entry that specifies the first hop to each possible destination from a given node. Without route aggregation, each of the n nodes in a network must store a routing table entry for all $n-1$ routes. By defining an aggregation scheme that allows multiple routes with the same first hop to be combined, the size of these routing tables can be reduced by orders of magnitude. A good example of this is CIDR routing, the routing scheme used in the Internet today. In CIDR, a route for an entire IP prefix can be specified with a single table entry.

In most Internet settings, names have already been assigned to hosts, so once routes are computed, minimizing the size of a routing table amounts to compressing the table to correctly specify routes with a minimum of table entries. However, in several other important settings, which we describe, addresses have not been assigned in advance, giving us the additional opportunity to assign addresses in such a way as to minimize the ultimate size of routing tables. This direction is the focus of our work. Given a topology, we seek to produce an IP address assignment that minimizes the sizes of the routing tables on the nodes in the network.

Underlying this problem are elegant graph-theoretical questions, which we describe in more detail in subsequent sections. Our work connects to these theoretical questions, but starts from a problem formulation that captures the complexities and nuances of assigning IP addresses to network interfaces in an environment where CIDR aggregation is used.

Optimizing IP address assignment offers benefits in many areas. In the real world, it can benefit enterprise networks, where large routing tables can overburden the memory-constrained legacy routers frequently found in such networks [25], and reassigning the IP addresses of entire networks is administratively feasible.

Likely of more immediate importance, it benefits simulation and emulation environments, in several ways. In these environments, test topologies typically come from topology generators, which rarely—if ever—annotate them with IP addresses. In network emulation environments such as Emulab [33], IP addresses are required simply to run at all. Assigning addresses manually is tedious and surprisingly error-prone: even for small topologies of a handful of nodes, we have found that Emulab experimenters rarely can provide an address assignment that is even internally consistent.

Consistent but randomly-assigned IP addresses are clearly not representative of the real Internet. Providing more sensibly-structured IP addresses is a requirement for accurate experimental evaluation in areas such as dynamic routing algorithms, firewall design and placement, and worm propagation. At a more basic level, most simulators are fundamentally unrealistic in that they name network nodes instead of network interfaces. In some cases the distinction can be important in specification and evaluation, as we and others have found [30], [3], [16].

Assigning IP addresses is not only a difficult problem for a user or operator to solve manually, but it can also matter to solve it well, as the space consumption for routing tables in simulators is frequently a serious barrier to scaling. Large routing tables have the same negative impact on performance in these environments as they do in reality.

Finally, looking at the key tools used in network research, network topology generators themselves stand to benefit from improved IP address assignment. Today's popular generators, such as GT-ITM [9] and BRITE [22], typically generate connectivity maps that are sometimes annotated with geographical information, but do not provide other useful node

and link annotations such as link bandwidths and delays, or realistic IP addresses. Our methods could serve as a back-end to these topology generators to compute sensible IP address annotations.

This paper makes the following contributions.

- We build upon a theoretical formulation of interval routing to formulate the IP address assignment problem as it needs to be solved by practitioners.
- We devise a general metric, "Routing Equivalent Sets," that quantifies the extent to which routes to sets of destinations can be aggregated.
- We develop three classes of algorithms to optimize IP naming, each with a fundamentally different approach to the problem.
- We implement the algorithms and evaluate them on a number of topologies, both automatically generated and from the real Internet.

The rest of the paper is organized as follows. In the next section we first start by defining a clean theoretical version of the problem, then outline some of the practical issues that complicate it. In Section III we discuss related work. Section IV details the algorithms we have developed, while Section V evaluates their effectiveness. Finally, we discuss future work and conclude.

II. Problem Statement

Our work seeks to produce a global address assignment automatically, i.e. an assignment in which IP addresses are assigned to each network interface in an internetwork. In practice, IP assignment directly impacts the sizes of routing tables, since a set of destinations with contiguous IP addresses that share the same first hop can be captured as a single routing table entry. Our primary aim in computing an assignment is to minimize the total space consumption of routing tables, which in turn helps to minimize packet processing time at routers. As we later discuss in detail, it is also important to consider the running time of an assignment algorithm in evaluating its effectiveness, and effective namespace (bitspace) utilization is another important factor. While our work explicitly assumes shortest-path routing, another possible consideration is that of producing a naming scheme that yields routes with small stretch, i.e. routes used are at worst a constant stretch factor longer than shortest-path routes. We formulate our assignment problem first using the clean conceptual notion of *interval routing*, widely used in theoretical studies, and then describe the additional constraints that CIDR prefixes and CIDR aggregation impose on the problem.

Formally, consider an n-node undirected graph $G = (V, E)$, where we will refer to vertices as hosts, and an edge (u, v) as a pair of outbound interfaces (one from u and one from v). An address assignment $\mathcal{A}$ assigns each vertex in V a unique label from the namespace of integers $\{1, \ldots n\}$. The routing table of vertex u associates one outbound interface (u, v) (next hop) with each label of a every vertex. In this manner, a subset of labels in $\mathcal{A}$ is associated with each outbound interface. Interval routing compacts the routing table by expressing the subset of labels in $\mathcal{A}$ as a set of intervals of integers.

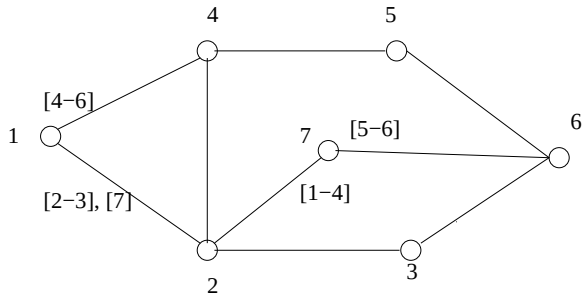

Fig. 1. A 7-node network with interval routes for nodes 1 and 7 shown.

To motivate these definitions, consider the example network depicted in Figure 1, in which nodes are assigned addresses from $\{1, \ldots, 7\}$. Interval routing table entries are shown for the outbound interfaces of nodes 1 and 7, and are depicted as interval labels on the corresponding edges. Node 7 can express its shortest-path routes into two disjoint intervals, one per interface, which corresponds to a routing table of size two. With the given address assignment, node 1 must use a total of three disjoint intervals to exactly specify the routes on its outbound interfaces. Note that in this example, ties between shortest-path routes can be exploited to minimize routing table size. For example, the routing table at node 7 elected to group node 3 on the same interface as nodes 1, 2, and 4 to save a table entry.

In a network using interval routing, the size of the minimal set of intervals is the routing table size, or the *compactness* of the routing table. We denote the number of entries in the routing table of vertex u by k_u. The theory literature has considered questions such as determining the minimum value of k for which an assignment results in routing tables all of size smaller than k [31], [14], [11]. For a given graph, this value of k is defined to be the *compactness* of the graph. We are primarily concerned with the *average* routing table size, so we seek to identify a labeling that minimizes $\sum_{u \in V} k_u$. It is well known that search and decision problems of this form are NP-complete, and several heuristics and approximation algorithms are known [14]. Our focus is on the practical considerations that cause CIDR routing to be a significantly different problem than interval routing; we discuss these differences and our approach next.

Practical Considerations

There are three main differences between the theoretical approach to compact addressing that we have described so far and the actual addressing problem that must be solved in emulation and simulation environments. First, although interval routing is intuitively appealing and elegant, routing table aggregation in practice is performed using the set of classless inter-domain routing (CIDR) rules [13], adding significant complexity. Second, in IP addressing, each individual interface (outbound edge) of a node is assigned an address, not each vertex. This changes what we are naming. Finally,

widely used local-area network technologies such as Ethernet provide all-to-all connectivity, and these networks are best described by *hypergraphs*, not generic graphs. We discuss these practical considerations and their consequences in the problem formulation next.

Classless inter-domain routing (CIDR) specifies aggregation rules that change the problem in the following ways. A CIDR address is an aggregate that is specified as a prefix of address bits of arbitrary length, and encompasses all addresses that match that prefix. This turns out to imply that a CIDR address can express only those intervals of IP addresses that are a power of two in size, and that start on an address that is a multiple of that same power of two, i.e. the interval must be of the form $[c*2^y, (c+1)*2^y]$, for integers c and y. This more restrictive aggregation scheme means that an IP assignment must be carefully aligned in order to fully take advantage of aggregation. In practice, dealing with this alignment challenge consumes many bits of the namespace, and address space exhaustion becomes an issue even when n is much smaller than the set of available names. Note that interval routing runs into no such difficulty. A second difference between interval routing and CIDR aggregation arises in practice because CIDR routing tables accommodate a *longest matching prefix* rule. With longest matching prefix, the intervals delimited by CIDR routing table entries may overlap, but the longest, and consequently most specific, matching interval is used for routing. The potential for use of overlapping intervals is advantageous for CIDR, as it admits more flexibility than basic interval routing.

When IP addresses are assigned, they are assigned to network interfaces, not hosts. For single-homed hosts, this is a semantic distinction, but for hosts with multiple interfaces, such as network routers, this distinction materially impacts address assignment. In this setting, a host may be associated with multiple addresses, which complicates the problem; when a packet is sent to any one of a host's addresses, it is typically expected to take the shortest path to any interface on the host. Thus, it is valuable to be able to aggregate all addresses assigned to a host. This means that we must not only be concerned with how LANs aggregate with each other, but also with how the interfaces on a host aggregate as well.

The networks we consider in simulation and emulation environments are best represented as *hypergraphs*, since they often contain local-area networks such as Ethernet, which enable all-pairs connectivity among a set of nodes, rather than connectivity between a single pair of nodes. A hypergraph captures this, since it is a generalized graph in which each edge is associated with a set of vertices rather than a pair of vertices. As before, when assigning addresses to a hypergraph, we wish to assign addresses to network interfaces. With the hypergraph representation, this becomes more difficult to reason about, since each network edge may be associated with a set of vertices of arbitrary size.

For convenience, we work instead with the dual hypergraph [2]; to find the dual hypergraph of a graph, we create a new graph with vertices that correspond to edges in the original graph, and hyperedges that correspond to vertices in the original graph. Each vertex in the dual hypergraph is incident on the edges that represent the corresponding vertices in the original graph. Thus, by labeling vertices of the dual hypergraph, we are labeling the network LANs and links in the original graph. We label with IP subnets, and then assign an address to each interface from the subnet of its adjacent LAN.

III. Related Work

Methods for optimizing an assignment of names to hosts in a network to minimize routing table size date back to the mid-1980s [31], [12]. In 1987, van Leeuwen and Tan formulated the notion of interval routing [31]; their work and subsequent work studied the problem of computing bounds on the *compactness* of graphs, a measure of the space complexity of a graph's shortest-path routing tables using interval routing [14], [11]. Their work is similar in direction to our problem; however, their work emphasizes worst-case routing table size for specific families of graphs and uses the idealized interval routing approach instead of CIDR.

A more recent direction, mostly pursued in the theoretical literature, is compact routing [1], [6], [29]. By relaxing the requirement of obtaining true shortest paths, compact routing enables much smaller routing tables at the expense of routes with *stretch*: the ratio between the routed source-destination path and the shortest source-destination path. Although these methods appear potentially promising for realistic Internet topologies [21], routing tables that support true shortest-path routes are still the norm in simulated, emulated, and real-world environments.

A different direction related to our work is that of designing routing table compression schemes for network simulators and emulators, to avoid the $O(n^2)$ memory required for precomputing all-pairs shortest-path routes. For example, NIx-Vector-related designs [26], [27] replace static tables with on-demand dynamic route computation and caching. Each packet contains a compact representation of the remaining hops to its destination. This source routing means that routing at each individual node is simple and fast. Depending on the traffic pattern, the size of this global cache can be much smaller than the memory required to pre-calculate all of the routes.

Another practical alternative uses spanning trees [5]. Several spanning trees are calculated, covering most of the edges in the topology. These spanning trees cover most of the shortest-path routes in the topology and a small cache is kept for the remainder. The spanning trees and cache can be stored in a linear amount of memory. While this is a novel routing method, it assumes global information, since routing requires access to the global spanning trees and cache, a potential bottleneck for distributed simulations and emulations.

Finally, there has been work on optimizing Internet routing tables. First, a number of guidelines for CIDR addressing have been proposed to facilitate manual assignment of IP addresses [15], [17] to take advantage of CIDR routing. Second, a method is known which minimizes the number of table

entries for a *given* set of routes and IP addresses. The Optimal Routing Table Construction (ORTC) [8] technique optimally compresses IP routing tables using CIDR rules. ORTC takes a routing table as input and produces a compressed routing table with the same functional behavior, taking advantage of CIDR rules to aggregate routes where possible. For any IP address assignment, ORTC optimally minimizes the number of routes in the routing table. We employ ORTC as a post-processing step in our work.

IV. Exploring the Problem Space

We now explore them problem space by defining three classes of algorithms for finding an approximate solution to this problem. Each algorithm represents a different kind of approximation. They constitute the technical contributions of this paper. In this section, we describe these algorithms, and in Section V, we evaluate their effectiveness in practice.

The goal of each of these algorithms is to build a binary trie, with nodes in the trie corresponding to IP subnets. In general, a binary trie is a special case of a binary tree in which left branches are labeled with '0', right branches are labeled with '1', and nodes are labeled with the concatenation of binary labels on edges along the root to leaf path. Using the trie we build, a node is given an IP address corresponding to its trie label, appended with zeroes to fill out the address in the event the label length is smaller than the desired address length. In this manner, proximate leaves in the trie correspond to proximate addresses in the IP assignment, so this is a natural representation of IP addresses.

Each of our three algorithms approaches the problem from a different direction. First, we describe an algorithm that uses a bottom-up greedy heuristic to creating a binary trie. The second algorithm decomposes the IP address assignment problem into two sub-problems, each of which is conceptually simpler. The third algorithm is a top-down approximation using graph partitioning methods. Finally, we present a way to preprocess the input topology to reduce the size of the problem.

A. Tournament Tree Building

Our first algorithm is based on the idea of mapping a graph directly onto an address assignment tree using a metric that guides the tree construction and a greedy heuristic. We begin by describing the metric, and then we describe the tree construction algorithm that uses this metric.

1) Routing Equivalence Sets: We have devised Routing Equivalence Sets (RES) as a way to characterize the effects aggregating the addresses of sets of vertices. The RES set of a set of destination vertices is the set of source vertices whose first hop is the same to every vertex in the destination set. Each member of the RES set must have the same first hop to all destination vertices, but that first hop can be different for different members of the RES set. More formally, let V be the set of vertices in a graph. Let D be a set of destination vertices. Let $H_x[y]$ be the first hop from source vertex x to destination vertex y. Then we define $\mathrm{res}(D)$ as:

$$\mathrm{res}\,(D) = \{v \in V : \forall d, e \in D, H_v[d] = H_v[e]\}.$$

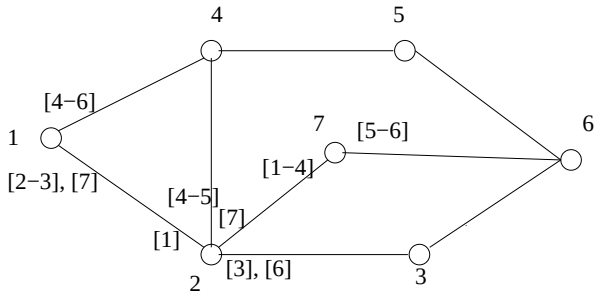

Fig. 2. 7-node network; interval routes for nodes 1, 2 and 7 shown.

To exemplify this definition, consider the small network depicted in Figure 2. This is identical to the network depicted in Figure 1, but with the routing table entries for node 2 shown. Now consider the set $\mathrm{res}(\{5, 6\})$. From the definition, this is the set of vertices whose first hop to node 5 is the same as the first hop to node 6. Nodes 1 and 7 share this property, as do nodes 3 and 4 (routing tables not depicted), but node 2 does not, since its first hop to node 5 is different than its first hop to node 6. Nodes 5 and 6 are in the destination set D itself, so are excluded from $\mathrm{res}(D)$, and thus $\mathrm{res}(\{5, 6\}) = \{1, 3, 4, 7\}$.

We use RES sets as a way to measure the impact of aggregating sets of vertices on routing tables. For example, consider a logical subnet (set of vertices) S_1 for which $|\,\mathrm{res}(\{S_1\})| = n - |S_1|$. In this ideal scenario, every vertex outside of S_1 uses the same outbound route for any vertex inside S_1. Therefore, we can safely assign all of the vertices in S_1 consecutive addresses, while using a single routing table entry for S_1 throughout the rest of the network. Now suppose that we are given a second subnet S_2 that also satisfies the ideal condition $|\,\mathrm{res}(\{S_2\})| = n - |S_2|$. Is S_2 a good candidate for aggregation with S_1? By performing the thought experiment of placing these two sets into a common prefix, there will be some set of vertices which will need separate routes to S_1 and S_2, and another set that does not require separate routes, because the first hop to all vertices in the new set is the same. This latter set is the RES set $\mathrm{res}(S_1 \cup S_2)$. The cardinality of this set tells us how many vertices will thus be able to save a route, giving us a measure of how many routes would be saved by combining two tree nodes. By considering pairwise unions of this form, and greedily pairing off the most promising pairs, we can build an aggregation tree from bottom-up.

One potential issue is that computing a RES set directly from this definition is expensive: computing res(D) has time complexity is $O(n|D|^2)$. However, we can prove that res(D) has a clean recursive decomposition that is amenable to much more efficient computation with the following lemma.

Lemma 1: For any sets D and E, and given res(D) and res(E), $\mathrm{res}(D \cup E)$ can be computed in time O(n).

Proof: First note that by transitivity, for any v and for

all $a, b, c \in V$:

$$(H_v[a] = H_v[b] \wedge H_v[b] = H_v[c]) \rightarrow (H_v[a] = H_v[c])$$

Further, the definition of RES and transitivity imply that for destination set D, and specializing to $v \in \text{res}(D)$ and $d, e \in D$,

$$H_v[a] = H_v[d] \rightarrow H_v[a] = H_v[e]$$

Which means that $\forall v \in V, \forall d \in D$:

$$\text{res}(D \cup \{v\}) = \text{res}(D) \cap \text{res}(\{v, d\})$$

Therefore, given two destination sets D and E, we can select *any* $d \in D$ and $e \in E$ to give the recurrence:

$$\text{res}(D \cup E) = \text{res}(D) \cap \text{res}(E) \cap \text{res}(\{d, e\}).$$

Since $\text{res}(\{d, e\})$ can be computed from the definition in O(n) time, and assuming use of standard set representations that allow intersection in linear time, the lemma follows. ■

2) The Tournament Algorithm: Using the RES metric, we now use a simple greedy algorithm to build a binary address assignment tree from the vertices in the graph. Initially, we start with a forest of n tree nodes, each of which represents a singleton vertex in the graph. In one round of the tournament, we coalesce a pair of tree nodes into an aggregate tree node, reducing the size of the forest by one. After $n - 1$ rounds, the tournament ends after having produced a rooted tree. We define the coalescence operation on an arbitrary pair of tree nodes, each of which represent disjoint subsets of vertices in the graph. Coalescing nodes i and j (representing vertex sets S_i and S_j respectively) entails creation of a new parent node k with i and j as its children, and setting S_k to be $S_i \cup S_j$. When nodes i and j are coalesced, they are removed from the forest and replaced by their parent k.

We determine the order of coalescence operations using a simple greedy algorithm. For any pair of tree nodes i and j, we maintain a score $\text{res}(\{S_i\} \cup \{S_j\})$, and the pair with the highest score is coalesced. Although n^2 pairwise combinations must be considered for each of $n - 1$ rounds, there is an optimization available that cuts the running time by a factor of n. In the first round, we must compute the RES metric for all n singletons, and all n^2 possible combinations for singletons. In subsequent rounds, however, the RES values of most of the possible n^2 combinations have not changed - in fact, the only ones that have changed are those pairs in which i or j were one of the combined nodes. There are at most $2n$ such combinations. So, in fact, we can store all possible combinations in a priority queue, and only update those entries that changed based upon the winners of a given round. Scoring a pairwise combination of two sets with RES can be done in $O(n)$ time as proven in Lemma 1. Thus, rounds after the first run in $O(n^2)$ time, and there are $n-1$ of them, leading to an overall time complexity of $O(n^3)$ for the tournament. Storing the values of combinations under consideration in a priority queue requires $O(n^2)$ space.

B. Spectral Ordering

Our next algorithms derive from the relationship between interval routing and prefix routing. We decompose the problem by first obtaining a good interval naming, then turning this into a prefix naming.

We begin by obtaining a *total ordering* of the vertices in G. This ordering can be used directly for interval routing, or we can build a tree from it for prefix routing. Instead of operating by attempting to combine all pairs of nodes in the graph, the tournament now only considers combining a node with its neighbors in the ordering. In the general case, this means we must only consider $|V| - 1$ possible combinations in the first round, and only need to score two new combinations in subsequent rounds. This reduces the complexity of the tournament to $O(n^2)$. If our ordering places nodes that are similar from a routing perspective, close to each other in the ordering, then the intuition is that the tournament algorithm will still be able to to produce a good assignment tree.

The remainder of this section discusses methods for obtaining an ordering.

1) Given Ordering: The simplest method for obtaining an ordering is to simply use the order given in the topology specification. We expect that for some topology generators and discoverers, this will produce a reasonable ordering. Depending on how the topology is generated, nodes that are linked may tend to be generated in some order that reflects their connectivity to each other. However, since we aim to assign addresses to topologies from any generator, we clearly cannot rely on any properties in the given ordering.

2) Standard Laplacian Ordering: Our next ordering technique is to use a standard technique from graph theory, that of obtaining an ordering using the Laplacian matrix [10] of the graph and the eigenvector associated with the second-smallest eigenvalue of the matrix [23], [18]. We refer to the second-smallest eigenvalue by λ_2 and the associated eigenvector by $\vec{v_2}$. The Laplacian matrix is essentially a representation of the adjacency of nodes in the graph, and thus it contains only *local* information about each node. The vector $\vec{v_2}$ contains a value for each vertex in the graph. These values can be used to generate an approximation for a minimum-cut bipartitioning of the graph. The characteristic value for each vertex is in relation to its distance from the cut, with vertices in the first partition having negative values, and those in the second partition having positive values. By sorting the vertices by their characteristic values, we obtain a spectral ordering.

3) Degree of Routing Equivalence: A limitation of using only the second-smallest eigenvector of the Laplacian is that this captures notions of adjacency, but does not necessarily capture the notions of similarity between vertices from the perspective of routing. We therefore considered an alternative Laplacian-like graph that goes beyond 0/1 adjacency values and instead incorporates real-valued coefficients that reflect the degree of similarity between a pair of vertices. To do so, we introduce a new metric, called Degree of Routing Equivalence (DRE). DRE is defined for a pair of vertices $i, j \in V$, and is

defined as:

$$\mathrm{dre}(i,j) = |\,\mathrm{res}(\{i,j\})|$$

We then construct an n by n matrix containing the DRE for every pair of nodes. In essence, what we have created is similar to the Laplacian of a fully-connected graph, with weights on the edges such that the higher the edge weight, the more benefit it will be to place two nodes together. This more directly captures the routing properties of nodes than the standard Laplacian. As with the Laplacian, we then take a characteristic valuation of the matrix to obtain an ordering.

C. Recursive Graph Partitioning

Recursive graph partitioning is an algorithm that approximates standard practice in assigning IP addresses. On the Internet, the Internet Assigned Numbers Authority (IANA) assigns a block of IP addresses to a Regional Internet Registry (RIR). The RIR assigns addresses to top-level ISPs and local or national registries. These institutions divide their blocks among their clients, many of which are mid-level ISPs. This assignment is applied recursively until a relatively small subnet is assigned to an individual or corporation.

We approximate this process by taking an input topology and creating a tree by recursively partitioning the topology. We partition the input topology, creating some number of subgraphs. Each subgraph is in turn partitioned into sub-subgraphs. At each level, the subgraphs get a smaller subnet, and then assign portions of that subnet to their own subgraphs.

The problem of graph partitioning is well understood and there are a number of libraries which provide good approximations. We have found METIS [20] to be a good linear-time partitioner. Because METIS runs in linear time, the recursive partitioning as a whole has a time complexity of $O(n \cdot \log(n))$.

One characteristic of METIS and graph partitioners in general is that they require the user to specify the number of partitions. We tried several scoring metrics for determining the number of partitions. We found two promising metrics for scoring a partitioning: conductance [19] and ratio cut [32]. Ultimately, we discovered that carefully picking a number of partitions had little impact on the final routing table sizes. Thus, we settled on always partitioning the graph into two subgraphs as the simplest method.

A critical piece that top-down algorithms must have is a useful termination condition. The obvious termination condition is when the partition is trivial (size 1). But a good algorithm must also behave well when the bitspace is exhausted. This is much easier in recursive partitioning than in any of the other algorithms. The recursive partitioner can check to see if a partitioning would result in a subgraph that is larger than the subnet size assigned to it. If such a situation occurs, naive addresses can be assigned to all of the LANs in the graph. Therefore, the current implementation for the recursive partitioner handles bitspace exhaustion more gracefully than the other algorithms that we have described.

D. Preprocessing the Topology

All of the algorithms we propose above have superlinear time complexities, which limits their ability to scale to large topologies. Thus, we have devised a prepass phase that decomposes input topologies into subgraphs to which we can assign addresses independently.

Additionally, there are some structures for which there are simple optimal algorithms for address assignment, like trees. Such structures are relatively common in some types of networks, such as at the edges of enterprise and campus networks, and thus it is worth optimizing these common cases. Another structure that is amenable to decomposition via a prepass is a singly connected component, i.e. a subgraph where removal of a single edge called a bridge breaks the component in two. By identifying dangling trees and bridge edges, the prepass phase naturally divides the graph into a set of smaller biconnected components and trees. Partitioning a graph into biconnected components and trees is a natural decomposition for our methods. The property we exploit is that if each biconnected component is assigned a unique prefix, the internal assignment of addresses within a component does not change the number of routes of any host outside of that component. We discuss technical details of our prepass methodology next, then quantify the costs and benefits of performing the prepass.

1) Hypergraph Biconnectivity and Hypertrees: In Section II, we described why our methods label the *dual hypergraph* of the input topology (also a hypergraph). To perform the prepass described above, we must extend the definitions of biconnectivity and trees into the domain of hypergraphs. There are a number of alternative definitions which potentially apply; we use the following one which best fits our purposes.

For every path p, the function $\mathrm{edges}(p)$ is the set of edges or hyperedges along that path. From here forward, we will use the term 'edge' in a general sense to denote either an edge or a hyperedge. A pair of vertices u and v is said to be *edge-biconnected* if and only if there exist two paths p and q between u and v such that:

$$\mathrm{edges}(p) \cap \mathrm{edges}(q) = \emptyset$$

Similarly, an *edge-biconnected component* is a set of vertices V such that for all u, v in V, u and v are edge-biconnected. An *edge-biconnected partitioning* of a graph G is a partitioning of the vertices of G into partitions $G_1, G_2, \ldots G_n$ such that for all i, G_i is a maximal edge-biconnected component.

Using similar notions, we define a *hypertree* to be a connected subgraph of a hypergraph that contains no cycles. As with trees on regular graphs, it is straightforward to optimally assign IP addresses to a hypertree of a hypergraph.

Using these definitions, our pre-processing step partitions the hypergraph into edge-biconnected components and hypertrees. Fast algorithms for computing such a decomposition on regular graphs are known; by maintaining some additional information about vertices incident to each hyperedge, these methods can be extended to apply to hypergraphs. Once the

decomposition is complete, addresses on the hypertrees are assigned optimally by a special tree-assignment procedure; addresses are assigned on the edge-biconnected components by the procedures described earlier. The super-graph of partitions is created and can be used to label the partitions themselves.

2) Preprocessing – Costs and Benefits: The preprocessing phase has the clear advantage of potentially being able to divide a very large problem into many smaller pieces that we then tackle with the more sophisticated methods defined earlier. A second potential benefit is reduction in routing table size. Because the prepass identifies subgraphs for which optimal assignment is possible, there can be a noticeable advantage when compared with methods that do not perform the prepass and are not capable of optimal assignment to subgraphs.

Somewhat less obviously, the partitioning performed in the prepass typically comes with a cost, in the form of some increase in routing table sizes. Suppose there are two edge-biconnected components A and B which are adjacent. Let a be a vertex within A that borders the component B. If the address of a were in the same subnet as the addresses in B, then the hosts in A would be able to save a routing table entry. Instead of having a route to a, and a different route to B, there could be a single route to them both. But we have assigned a to a different component and to a different subnet than B. Therefore, in general, a cannot be aggregated with B. This is the cost of dividing the graph.

We explore the tradeoffs that the prepass imposes on routing table sizes for generated and Internet topologies in Section V.

E. Putting It All Together

The previous subsections have detailed a large number of algorithms and optional steps. Figure 3 shows how they all fit together.

The recursive partitioner always runs on its own. The three alternative algorithms (Tournament, DRE Eigenvector, Laplacian Eigenvector) can be run with or without the prepass. If the prepass is run, it divides the graph into components, runs the appropriate algorithm on each component, then collates the results as part of the "Trie to IP addresses" stage. On tree components it runs the tree-assigner and runs one of the other three algorithms on non-tree components. Finally, we always run ORTC to optimally compress the routing tables.

Most of these paths will be evaluated in the next section.

V. Results

A. Methodology

We ran experiments on topologies from three sources: two popular router-level topology generators, and the Internet. Our primary interest is the generated topologies because such topologies are prevalent in simulation and emulation. In addition, real-world graphs are more likely to come annotated with real IP addresses.

The first set of topologies are generated by the BRITE [22] topology generator, using the GLP [4] model proposed by

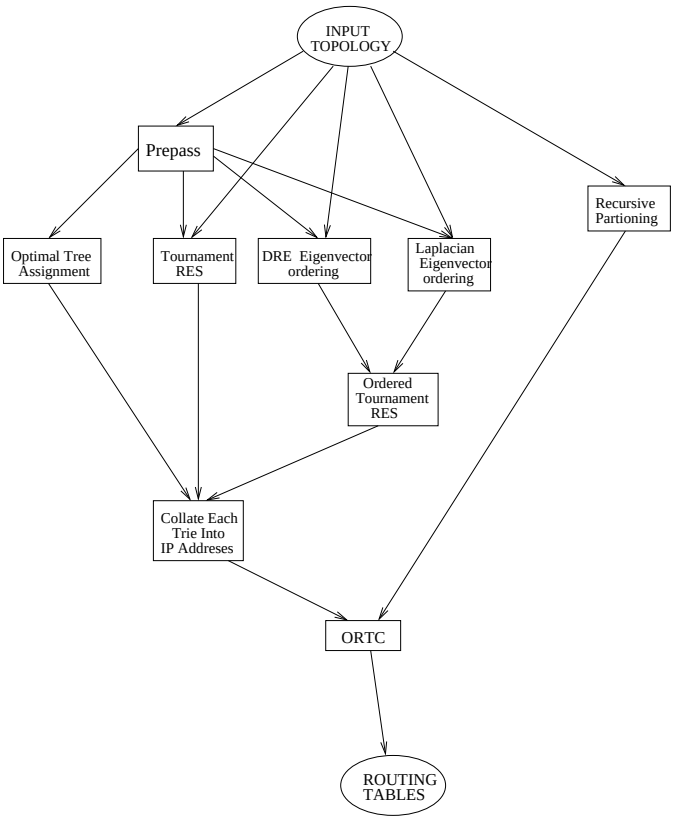

Fig. 3. A flowchart showing how the different algorithms are combined.

Bu and Towsley. These topologies are intended to model the topology within a single organization, ISP, or AS.

The second set of topologies are generated by the GT-ITM [9] topology generator. They model topologies that include several different administrative domains; each topology consists of some number of transit networks and stub networks. Thus, they contain some level of inherent hierarchy.

Finally, we use some real-world topologies gathered by the Rocketfuel topology mapping engine [28]. These are maps of real ISPs, created from traceroute data for packets passing through the ISP for a large set of $(source, destination)$ pairs. All Rocketfuel graphs are of individual transit ASes.

We report on the number of interfaces, rather than the number of nodes, for each topology. Since it is interfaces that must be named, this gives a more accurate view of the complexity of the assignment problem. Our test topologies, like those generated by most topology generators, contain links and not LANs. Thus, the number of interfaces is twice the number of links in the topology.

After generating IP addresses, we generate routing tables and compress them with ORTC [8], which, for a given address assignment, produces a provably optimal routing table. We also run the generated routing tables through a consistency checker to verify their correctness. The run times reported are for the assignment of addresses, as routing table generation and compression need to be performed no matter what the assignment algorithm is.

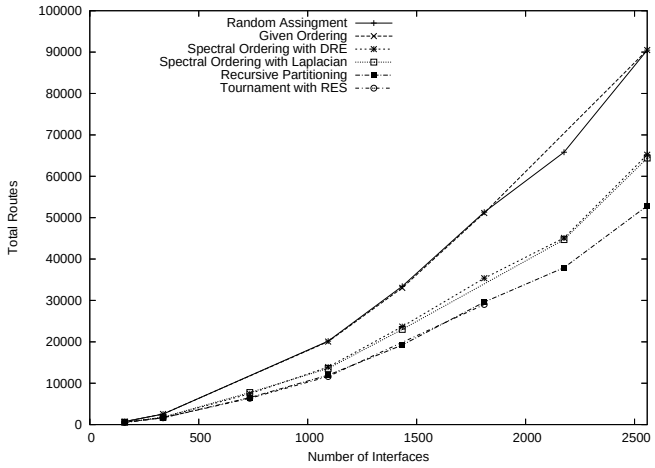

Fig. 4. Global number of routes for a variety of assignment methodologies. Topologies are from the BRITE topology set.

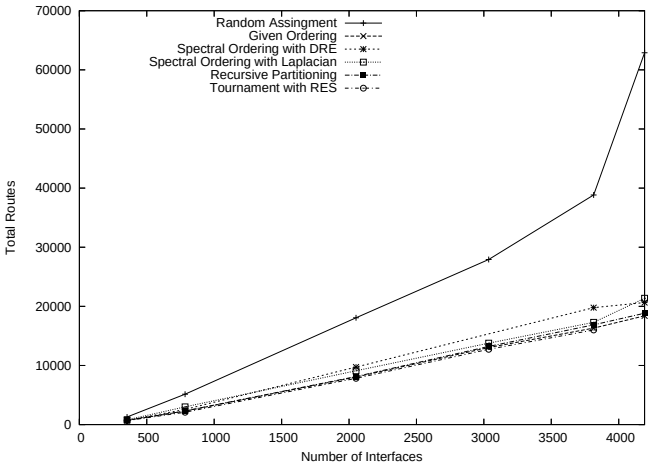

Fig. 5. Global number of routes for a variety of assignment methodologies. Topologies are from the GT-ITM topology set.

All of our experiments were run on Pentium IV Xeon processors running at 3.0 GHz, with 1 MB of L2 cache and 2 GB of main memory on an 800 MHz front side bus.

B. Full-Graph Algorithms

We begin by comparing results for the set of assignment methodologies without using the pre-pass stage. Due to the high time and memory complexity of many of these methods, we are limited in the size of topologies we can compare. In addition, some of the algorithms run out of bitspace for larger topologies—in other words, the trees they build produce leaves that are too deep to be represented by an IPv4 address. This can occur when the trees built are very unbalanced, leaving large parts of the IP space unused. Investigating ways to constrain the bitspace used by the tournament algorithms will be a priority in future work.

1) BRITE topologies: Figure 4 shows the global routing table size produced by each method for the BRITE topology set. There are a number of interesting things in this graph. First, note that all methods do significantly better than random assignment, with the best at around 2500 interfaces (recursive partitioning) saving 42% of routes over random. Also note that for this topology generator, the ordering produced by the generator is indistinguishable from a random ordering.

The two spectral methods perform very similarly to each other; this is surprising, because the Laplacian matrix contains only local information, while the DRE matrix contains global routing-specific information. This suggests that global knowledge is not as useful in this situation as one might think.

Finally, we see that recursive partitioning and tournament RES also perform similarly. However, RES stops at 1800 interfaces—at this point, it begins creating trees that are too deep to be represented as IP addresses. This shows clearly that we will need to bound the bitspace used by the tournament algorithm if it is to be a viable strategy.

2) GT-ITM topologies: Figure 5 shows results from the GT-ITM topology set. This time, the route savings are more pronounced—the best improvement we see over random assignment is a 70% decrease in the number of routes. However, the various assignment methodologies are much more clustered—most result in similar routing table sizes. It is interesting to note that this time, the given ordering, while the poorest, is competitive with the more sophisticated orderings. We believe this to be an artifact of how GT-ITM generates its topologies.

TABLE I

NUMBER OF ROUTES GENERATED FOR THE ROCKETFUEL TOPOLOGIES.

Algorithm	EBONE	Tiscali
Random	27031	33104
Given Ordering	15904	18891
Spectral Laplacian	18128	22761
Spectral DRE	15581	20579
Recursive Partitioning	11630	16802
Tournament RES	11427	16354

3) Rocketfuel topologies: The Rocketfuel results can be seen in Table I for two European networks, EBONE and Tiscali. Like the BRITE topology set, the tournament RES and recursive partitioning algorithms perform similarly, with a slight advantage going to tournament RES.

4) Run Time Comparison: Finally, Figure 6 shows the run times for the BRITE topology set for our various assignment algorithms. Here, recursive partitioning is the clear winner—its run time looks roughly linear, while all other methods have a quadratic curve to their run time.

Clearly, these methods show recursive partitioning as the preferred method among those evaluated. While it is sometimes bested by tournament RES in terms of number of routes, it has the best run time scaling of all methods, with near-best performance in terms of number of routes.

C. Pre-pass effects

We now evaluate the pre-pass and its effects on the the address assignment area. We take two of the algorithms algorithms above—the DRE spectral ordering, and the RES

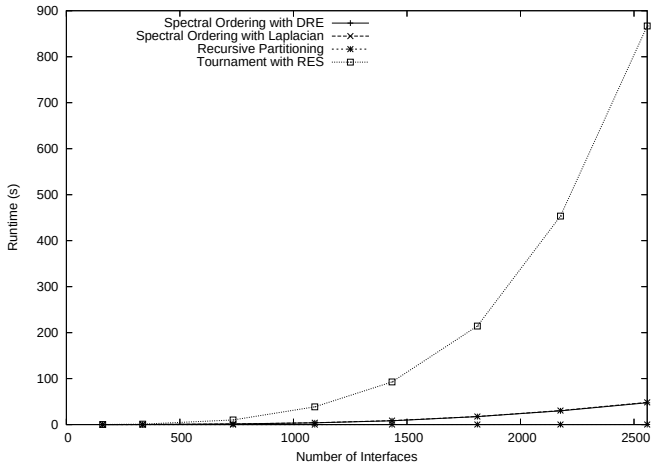

Fig. 6. Run times for a variety of assignment methodologies. Topologies are from the BRITE topology set.

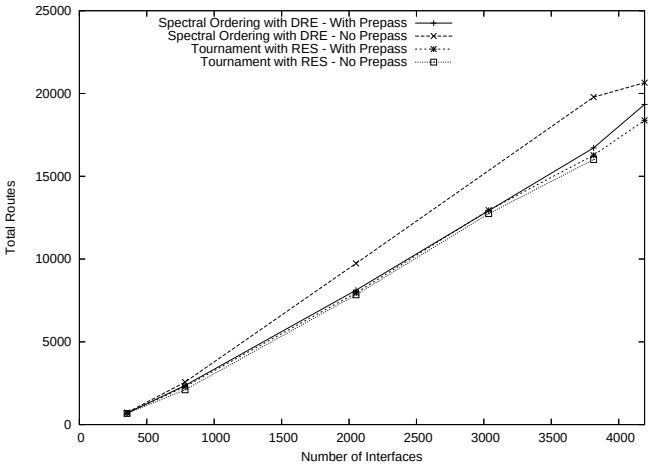

Fig. 7. Routing table sizes for the GT-ITM graphs, with and without the pre-pass.

TABLE II

HISTOGRAM OF PREPASS COMPONENT SIZES FOR THREE GRAPHS.

Component size	GT-ITM	BRITE	Rocketfuel
1	4	15	29
2–10	49	22	23
11–20	1	4	1
21–30	1	4	
91-100	1		
321-330		1	
420-430			1
# of components	56	46	54
# of links	392	547	543

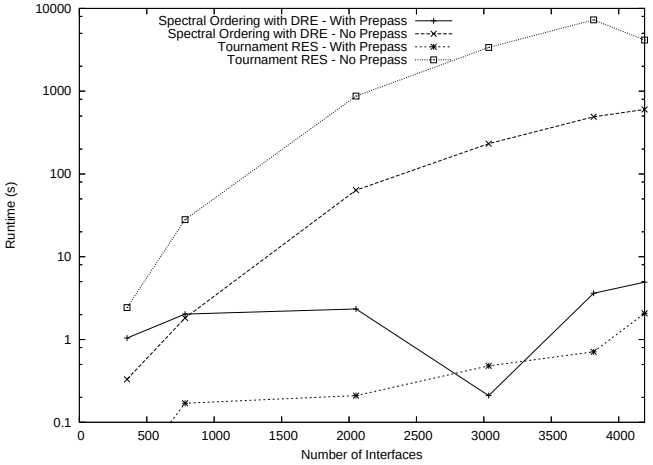

Fig. 8. Run times with and without the pre-pass. Note that the y-axis is in a logarithmic scale.

tournament—and study the effects of the pre-pass on them. Due to the similar results above between DRE and Laplacian orderings, we do not consider the latter here. We use the GT-ITM graphs for these comparisons. The other topology sets cause the tournament-based algorithms to use too many bits, even for small topologies, so a way to bound the number of bits they use will be required to make these graphs feasible.

We expect to see three effects. First, the pre-pass finds tree-like structures and assigns to them optimally, which should tend to improve results. Second, by making some decisions locally instead of globally, we will clearly make some sub-optimal decisions. Third, by reducing the number of nodes fed to portions of the assignment tool chain with quadratic time complexity, we should see a reduction in the run time for those types of assignment.

1) Components Found: We pick one topology from each of the three sets as a representative; the ones chosen have the most similar numbers of links. Table II shows a histogram of the sizes of the components into which the prepass divides these input topologies. The smaller the largest component, the better the running time of the quadratic algorithms. We can see from this table that the prepass has varying levels of effectiveness for the different topology types. For the GT-ITM topology, the largest component is roughly one-fourth of the topology size, while on the other topologies, the largest components are three-fifths and four-fifths of the size of the original topology. In all topologies, the majority of the components are smaller than 10 nodes. In all three topologies, the largest components are biconnected. The smaller components are dominantly trees.

2) Routing Table Size: In Figure 7, we can see that, for these graphs, the positive and negative effects of the pre-pass largely balance each other out. The tournament RES algorithm sees very little net change, and the DRE spectral ordering sees a slight improvement.

3) Run Time Benefit: Figure 8 shows the run time improvement due to the pre-pass. Note that the improvement is so dramatic that the y-axis is shown in a logarithmic scale. By splitting up the graph into smaller, simpler pieces, the pre-pass is able to bring the run times for the algorithms down to a feasible level.

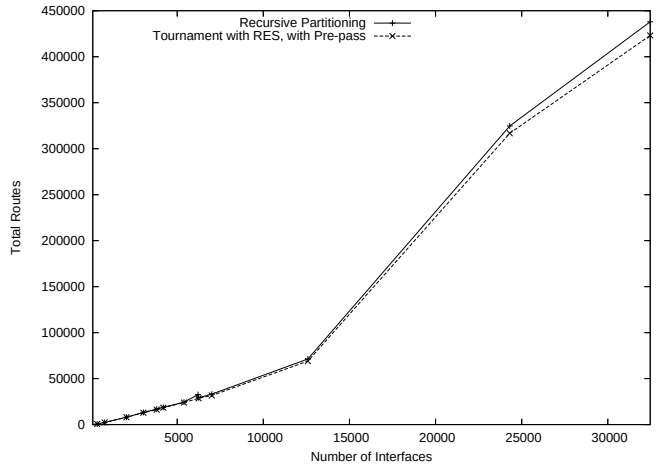

Fig. 9. Total number of routes on large graphs.

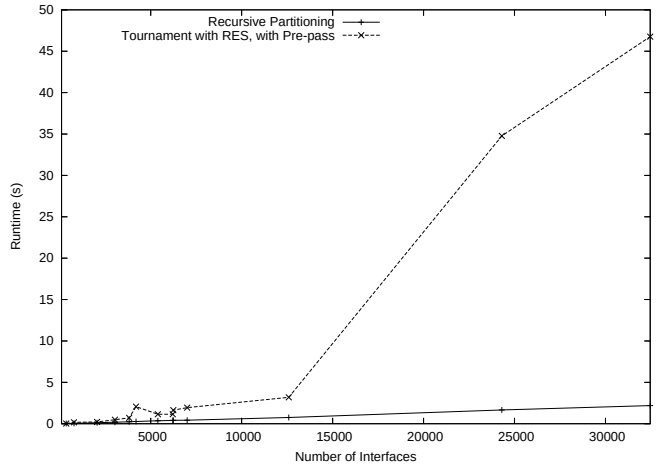

Fig. 10. Run times on large graphs.

D. Large Graphs

Finally, we compare the best of the full-graph algorithms, recursive partitioning, to the best of the pre-pass algorithms, Tournament RES. Since both of these scale much better than most of their counterparts, we were able to run them on much larger graphs. Figure 9 shows the number of routes for these experiments, and Figure 10 shows the run times. The two algorithms are nearly equal in the number of routes that they produce, with the slight advantage again going to tournament RES. For graphs under 12,500 interfaces, their run times are comparable and very low, but for the largest graphs, recursive partitioning shows much better scaling.

E. Summary of results

Figure 11 summarizes the results across all algorithms, showing the number of routes for one large, representative topology from each generator, and for both Rocketfuel graphs. The number of routes for each topology is normalized to the number of routes for the random address assignment. There are two clear trends from these results: First, in most topologies the spectral orderings perform worse than the given ordering. Second, the recursive partitioning and RES tournament methods consistently yield the fewest routes, with RES usually having a slight edge. The clear conclusion is that the latter two methods are superior.

VI. Future Work

The top priority of future work will be to find ways to gracefully handle limited bitspace. This issue must be dealt with both when the IP tries are being converted to addresses, and in each of the tournament algorithms. In the tournament algorithms, we are developing techniques to constrain the amount of bitspace used within a given constraint. Converting IP tries to addresses will involve a tradeoff between efficient bitspace usage and smaller routing tables. Since these two goals are measured in different units, arriving at such a balance is difficult: judging the relative value of a bit of namespace as compared with a route will require study and tuning. Making these decisions in the context of the tournament algorithms is particularly difficult, since we must make them with incomplete knowledge; because we are building tries from the bottom up, we know only about how efficiently bits are allocated in the subtrees below us, but not how efficiently they will be allocated in the nodes above us.

The tournament algorithms that we use are greedy, so there is almost certainly room to improve them. This improvement may take the form of pruning the solution space by not considering combinations that we know cannot or are not likely to be part of the optimal solution. It may also take the form of introducing some lookahead, so that the algorithms have some room to trade lower scores in one round for higher scores in a later round.

One of the potential causes of the spectral ordering decomposition's relatively poor performance is that ordering based on a single eigenvector is in essence a single partitioning of the graph. Recursively ordering each partition based on the eigenvector may improve the spectral orderings greatly; the success of the recursive partitioning algorithm suggests that this may be the case. We can also try to adapt well-known algorithms which approximate Minimum Linear Arrangement [7], [24] to the ordering problem. Because transforming the ordering to a tree is the cheapest part (in time) of the ordering algorithms, there may be ways of improving the results while keeping the total runtime low.

Our results so far neither give us an intuitive feel nor quantitative metrics for two important aspects of IP naming. First, how close to optimal are we getting? Second, how close to real life address assignments are we getting? Studying real-world topologies for which we have full IP address data will help us approach both these problems. We are working on getting such data for enterprise and ISP networks. Developing a "realism" metric is a challenging open problem, but more tractable will be to develop a metric that reflects the degree to which two namings differ in important ways.

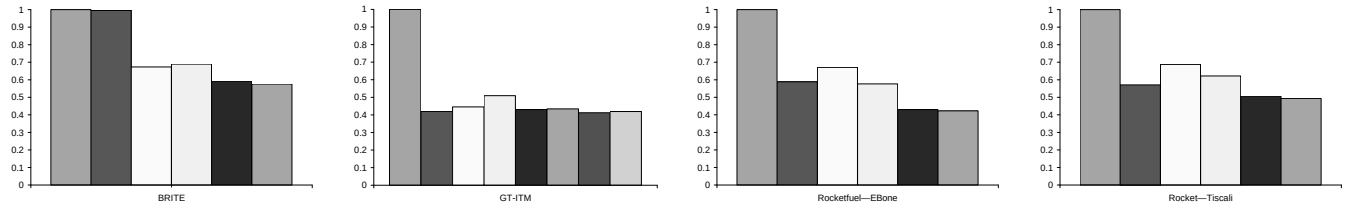

Fig. 11. Summary comparison of global routing table sizes resulting from different algorithms. The results are normalized to the largest table size, which results from *random assignment*. From left to right, the bars illustrate the results of *random assignment*, *given ordering*, *spectral ordering with Laplacian*, *spectral ordering with DRE*, *recursive partitioning* and *tournament RES*. For GT-ITM, in the last two bars we also show the results of *tournament RES with pre-pass*, and *spectral ordering with DRE and pre-pass*.

VII. Conclusion

We have investigated challenges associated with annotating an Internet connectivity map with IP addresses in such a way as to minimize the size of routing tables on hosts and routers. There is a large body of related theoretical work. However, none of it adequately handles the complexities of CIDR aggregation: longest prefix matching, the need to name network interfaces instead of hosts, and the nuances of addressing hosts on LANs with all-pairs connectivity. We argue that these factors must be considered in realistic simulation and emulation environments, and our experimental results indicate that they impose a challenging set of constraints beyond those imposed by a more basic interval routing problem.

The methods that we proposed and implemented attacked the problem from a number of angles. We investigated the use of graph partitioning tools, novel metrics which quantified the "routing similarity" between sets of vertices in the graph, algorithms for computing a spectral ordering across the vertex sets, and a prepass phase that can significantly reduce the effective problem size. All of our methods produce routing tables that are far better than those that result from naive, randomly chosen assignments, but no one method dominates along all of the relevant performance axes on all topologies, e.g., running time, bitspace consumption, and number of routes. Tournament RES consistently produces the smallest number of routes. Recursive partitioning consistently runs the fastest, and produces routing tables that are close in size to those produced by tournament RES. On the other hand, the best of the methods we propose are suitable for annotating a topology consisting of thousands of interfaces with IP addresses in a matter of minutes. Therefore, we are hopeful that our methods are suitable for incorporation either as a back-end to existing topology generators, or as a front-end to existing simulation and emulation environments that take connectivity maps as input, or both.

Acknowledgments

We thank Shang-Hua Teng for helpful discussions regarding spectral orderings and for advice on working with the Laplacian of DRE values.

References

[1] B. Awerbuch, A. Bar-Noy, N. Linial, and David Peleg. Improved routing strategies with succinct tables. *J. of Algorithms*, 11(3):307–341, 1990.

[2] C. Berge. *Graphs and Hypergraphs*, volume 6, pages 389–390. North-Holland, Amsterdam, second edition, 1976.

[3] Robert Braden. RFC 1122: Requirements for internet hosts—communication layers. Technical report, IETF, 1989.

[4] T. Bu and D. Towsley. On distinguishing between Internet power law topology generators. In *Proc. INFOCOM*, pages 1587–1596, July 2002.

[5] J. Chen, D. Gupta, K. V. Vishwanath, A. C. Snoeren, and A. Vahdat. Routing in an Internet-scale network emulator. In *MASCOTS*, pages 275–283, 2004.

[6] L. Cowen. Compact routing with minimum stretch. In *SODA: ACM-SIAM Symposium on Discrete Algorithms*, pages 255–260, 1999.

[7] J. Diaz, J. Petit, and M. Serna. A survey of graph layout problems. *ACM Computing Surveys*, 34(3):313–356, 2002.

[8] R. Draves, C. King, S. Venkatachary, and B. Zill. Constructing optimal IP routing tables. In *Proc. of IEEE INFOCOM*, pages 88–97, 1999.

[9] K. Calvert E. Zegura and S. Bhattacharjee. How to model an internetwork. In *Proc. of IEEE INFOCOM*, pages 594–602, March 1996.

[10] M. Fiedler. Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal*, 23(98):298–305, 1973.

[11] M. Flammini, J. van Leeuwen, and A. Marchetti-Spaccamela. The complexity of interval routing on random graphs. *The Computer Journal*, 41(1):16–25, 1998.

[12] G. N. Frederickson and R. Janardan. Designing networks with compact routing tables. *Algorithmica*, 3:171–190, 1988.

[13] V. Fuller, T. Li, J. Yu, and K. Varadhan. RFC 1519: Classless inter-domain routing (CIDR): an address assignment and aggregation strategy, September 1993.

[14] C. Gavoille and D. Peleg. The compactness of interval routing. *SIAM J. on Discrete Mathematics*, 12(4):459–473, 1999.

[15] E. Gerich. RFC 1466: Guidelines for management of IP address space, May 1993.

[16] M. Hibler et al. Feedback-directed virtualization techniques for scalable network experimentation, May 2004. University of Utah Flux Group Technical Note, http://www.cs.utah.edu/flux/papers/virt-ftn2004-02.pdf.

[17] K. Hubbard, M. Kosters, D. Conrad, D. Karrenberg, and J. Postel. RFC 2050: Internet Registry IP Allocation Guidelines, November 1996.

[18] M. Juvan and B. Mohar. Optimal linear labelings and eigenvalues of graphs. *Discrete Applied Mathematics*, 36(2):153–168, 1992.

[19] R. Kannan, S. Vempala, and A. Vetta. On clusterings: Good, bad and spectral. *Journal of the ACM*, 51(3):497–515, May 2004.

[20] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. on Scientifi c Computing*, 20(1):359–392, 1999.

[21] D. Krioukov, K. Fall, and X. Yang. Compact routing on Internet-like graphs. In *Proc. IEEE INFOCOM*, pages 209–219, 2004.

[22] A. Medina, A. Lakhina, I. Matta, and J. Byers. BRITE: An approach to universal topology generation. In *MASCOTS 2001*, pages 346–356, August 2001.

[23] B. Mohar. The Laplacian spectrum of graphs. In Y. Alavi, G. Chartrand, O. Ollermann, and A. Schwenk, editors, *Graph theory, combinatorics, and applications*, volume 2, pages 871–898, New-York, 1991. John Wiley and Sons, Inc.

[24] S. Rao and A. W. Richa. New approximation techniques for some ordering problems. In *SODA '98: Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 211–218, Philadelphia, PA, USA, 1998. Society for Industrial and Applied Mathematics.

[25] Jennifer Rexford. Personal communication, May 2005.

[26] G. F. Riley, M. H. Ammar, and R. Fujimoto. Stateless routing in network simulations. In *MASCOTS 2000*, pages 524–531, 2000.

[27] G. F. Riley and D. Reddy. Simulating realistic packet routing without routing protocols. In *PADS'05: Proc. of the 19th Workshop on Principles of Advanced and Distributed Simulation*, pages 151–158, Washington, DC, 2005. IEEE Computer Society.

[28] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies with Rocketfuel. In *SIGCOMM 2002*, pages 2–16, Pittsburgh, PA, 2002.

[29] M. Thorup and U. Zwick. Compact routing schemes. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 1–10, 2001.

[30] J. Touch, Y.-S. Wang, L. Eggert, and G.G. Finn. A virtual internet architecture. In *ACM SIGCOMM Workshop on Future Directions in Network Architecture (FDNA 2003)*, Karlsruhe, Germany, August 2003.

[31] J. van Leeuwen and R.B. Tan. Interval routing. *The Computer Journal*, 30:298–307, 1987.

[32] Y. Wei and C. Cheng. Ratio cut partitioning for hierarchical designs. *IEEE Trans. on Computer-Aided Design*, 10(7):911–921, July 1997.

[33] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*, pages 255–270, Boston, MA, December 2002.