A Quantitative Comparison of Graph-based Models for Internet Topology

Ellen W. Zegura and Kenneth L. Calvert and Michael J. Donahoo

Abstract—
Graphs are commonly used to model the topological structure of internetworks, to study problems ranging from routing to resource reservation. A variety of graphs are found in the literature, including fixed topologies such as rings or stars, "well-known" topologies such as the ARPAnet, and randomly generated topologies. While many researchers rely upon graphs for analytic and simulation studies, there has been little analysis of the implications of using a particular model, or how the graph generation method may affect the results of such studies. Further, the selection of one generation method over another is often arbitrary, since the differences and similarities between methods are not well understood.

This paper considers the problem of generating and selecting graph models that reflect the properties of real internetworks. We review generation methods in common use, and also propose several new methods. We consider a set of metrics that characterize the graphs produced by a method, and we quantify similarities and differences amongst several generation methods with respect to these metrics. We also consider the effect of the graph model in the context of a specific problem, namely multicast routing.

Keywords: Scalability, Network modeling, Internetworking

1 Introduction

1.1 Background

The explosive growth of the Internet has been accompanied by a wide range of internetworking problems related to routing, resource reservation, and administration. The study of algorithms and policies to address such problems often involves simulation or analysis using an *abstraction* or model of the actual network structure and applications. The reason is clear: networks that are large enough to be interesting are also expensive and difficult to control, therefore they are rarely available for experimental purposes. Moreover, it is generally more efficient to assess solutions using analysis or simulation — *provided* the model is a "good" abstraction of the real network and application. It is therefore remarkable that studies based on randomly-generated or trivial network topologies are so common, while rigorous analyses of how the results scale or how they might change with a different topology are so rare.

The state of the art in network modeling includes:

- Regular topologies, such as rings, trees and stars (e.g., [12, 44, 64])
- "Well-known" topologies, such as the ARPAnet or NSFnet backbone (e.g., [5, 54, 65])
- Randomly generated topologies (e.g., [55, 58, 61])

The limitations of each of these are obvious: well-known and regular topologies reflect only parts of current or past real networks; random topologies may not reflect any (past, present or future) real network. Also clear from the cited references is the diverse set of problems that rely on network models to evaluate performance. Furthermore, most researchers seem to be aware of the perils of reaching conclusions about real networks based on these models; it is typical for papers to include a disclaimer to this effect.

To illustrate the important role that the network model can play in assessing algorithms, consider the following results:

- Doar and Leslie found that the efficiency of their dynamic multicasting algorithms was reduced by as much as a half when using random graphs versus using hierarchically structured graphs designed to reflect some of the properties of real internetworks. (See Figure 5 in [26].)
- Wei and Estrin found that the traffic concentration in core-based multicast routing trees is comparable to traffic concentration in shortest-path trees for a network model with average node degree of about 3.0, but the traffic concentration is almost 30% higher in core-based trees when average node degree increased to 8.0. (See Figure 9(a) in [63].)
- Mitzel and Shenker found that multicast resource reservation styles compared quite differently in the quantity of resources reserved for linear, tree and star topologies. (See Tables 4 and 5 in [44].)

It should be clear from these examples that the network model *does* matter: the conclusions reached about the suitability and performance of algorithms may vary depending on the methods used to model the network.

A variety of criteria may be applied to assessing a network model, depending in large part on the intended use. For example, if the purpose is to stress-test the algorithm, the model should generate instances which are, in some sense, "difficult." For the problem of routing, this may mean topologies with moderate node degree and many routing choices. If the purpose is to model a particular (static) network (e.g., a campus or corporate network), then the model should accurately reflect the current topology.

Historically, large networks such as the Public Switched

E. Zegura, K. Calvert, and M. Donahoo are with the College of Computing, Georgia Institute of Technology.

Telephone Network have grown according to a topological design developed by some central authority or administration [7]. In contrast, there is no central administration that controls —or even keeps track of— the detailed topology of the Internet. Although general characteristics of its topology are known, it is impractical, if not impossible, to construct a detailed topological map of the Internet due its decentralized administration and sheer scale.[1] Further, the structure of the Internet is changing at a rapid rate, limiting the accuracy of any snapshot of topology. These considerations make it difficult to rigorously validate the "realism" of any graph model. In such cases, where we have some —albeit incomplete— knowledge of the current and expected future structure of the network, the model should reflect the known properties, and instantiate the remainder of the topology in some reasonable, random fashion.

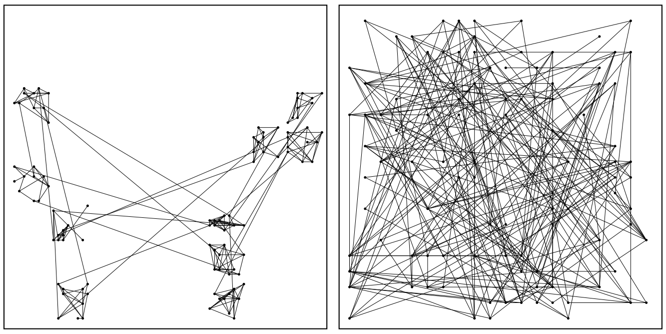

Figure 1: The dangers of visual representation

1.2 Overview and Roadmap

In this paper we consider the question of how to generate graph models that have path characteristics like those of the Internet. We also address the issue of the significance of differences between graphs generated with different methods. Our goal is to provide information that can be used in selecting a graph to use for a particular purpose. We begin by describing the general topological characteristics of the Internet (Section 2). It is this structure that the reader should keep in mind as we describe the common methods to model internetwork topology.

In Section 3 we describe three classes of graph generation methods: flat random, regular, and hierarchical. Common methods in the literature fall primarily into the flat random and regular classes. Within the hierarchical class, we propose a new Transit-Stub model having properties of hierarchy, locality and routing policy, as found in real internetworks.

We then analyze the outputs of the various generation methods (Section 4), with the goal of providing a basis for selecting a generation method, comparing one method to another, or comparing the graphs generated by a method to (a set of) known real topologies. We propose a set of topological properties as a basis for such comparisons. Note that a rigorous system for quantifying the characteristics of models is essential; relying on visual representation (e.g., noting that a layout bears a visual resemblance to geographic maps of the Internet) can be terribly misleading. For example, the two topologies in Figure 1 each have 100 vertices and approximately the same number of edges (230 versus 231), a fact that is counter to the visual impression.

Given a set of generation methods (Section 3) and a set of metrics on the generated graphs (Section 4), an obvious task is the comparison of one method to another. Before tackling that question, however, we must determine the effect of the method parameters on the metrics of the generated graphs *within* a particular method; Section 5 describes our investigation of this relationship. A statistical comparison of metric distributions *across* the different generation methods provides a basis for some observations about method similarities and differences; these are presented in Section 6. We augment these *general* observations by investigating the effect of the generation method in the context of a *specific* problem typical of those of for which graph models are used, namely multicast routing. These results are presented in Section 7.

2 Internet Domain Structure

Throughout this paper it is assumed that the goal is to model the *paths* (i.e., sequences of nodes) along which information flows between nodes in an internetwork. Typically, undirected graphs are used to represent the set of these paths, with nodes representing *switches* or *routers*, and edges representing forwarding paths between switches. A forwarding path may be a direct (physical) link, or it may be a shared medium; we do not distinguish these two cases. For example, an FDDI ring to which four IP routers are connected would be represented as a clique of four nodes. In an internetwork there may be many paths between a pair of nodes; at any time one of these is considered the "best," or primary path, typically because it is the shortest (by some metric) of the possible paths. When we mention "the path between two nodes," we refer to this primary path.

We do not model individual hosts; it is therefore sensible to have a switching node with an edge to only one other node, a situation that is not uncommon in actual internetworks. We also do not model link characteristics (e.g., bandwidth); such information can be easily included as an overlay to the basic topological structure, as needed for studying particular problems.

Today's Internet can be viewed as a collection of interconnected *routing domains* [19], which are groups of nodes that are under a common administration and share routing information. A primary characteristic of these domains is *routing locality*: the path between any two nodes in a domain stays entirely within the domain. Each routing

[1] Through interdomain routing protocols it is possible to obtain information about high-level connectivity covering a significant fraction of the Internet here but assembling complete end-to-end information —i.e. down to the level of last-hop routers— would require the cooperation of hundreds of different domain administrations.

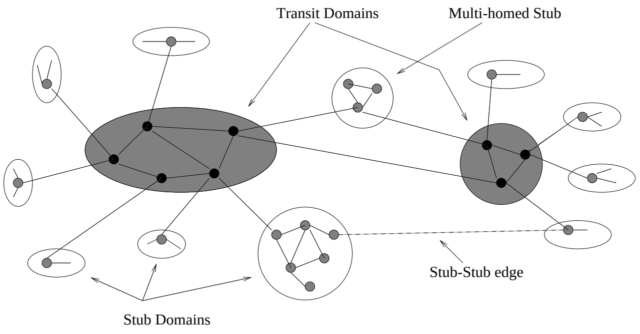

Figure 2: Example of Internet domain structure

domain in the Internet can be classified as either a *stub* domain or a *transit* domain. A domain is a stub domain if the path connecting any two nodes u and v goes through that domain only if either u or v is in that domain; transit domains do not have this restriction. The purpose of transit domains is to interconnect stub domains efficiently; without them, every pair of stub domains would need to be directly connected to each other. (See Figure 2.)

A transit domain comprises a set of *backbone* nodes, which are typically fairly well connected to each other. In a transit domain each backbone node also connects to a number of stub domains, via *gateway* nodes in the stubs. Some backbone nodes also connect to other transit domains. Stub domains can be further classified as single- or multi-homed. Multi-homed stub domains have connections to more than one transit domain. Single-homed stubs connect to only one transit domain. Some stubs also have links to other stubs.

The routing characteristics of the current Internet can be summarized by the following general principles regarding the path between two arbitrary nodes u and v:

- If u and v are in the same domain, the path between them remains entirely within that domain.
- If u is in domain U and v is in domain V, the path from u to v goes from U, through zero or more transit domains, to V.

Routing domains range in size from one or two nodes to hundreds or thousands of nodes.

To summarize, the primary structural characteristic affecting the paths between nodes in the Internet is the distinction between stub domains and transit domains: a path connecting two nodes in different stub domains will never pass through any stub domain other than those two. In other words, there is a *hierarchy* imposed on nodes.[2] In the next section, we consider methods of generating graphs that model the structure of internetworks; as we shall see, some commonly-used models fail to capture these restrictions.

3 Graph Generation Methods

When modeling an entity with some details that are unknown, or may vary, the standard approach is to substitute nondeterminism for the unknowns, generate multiple instances, and analyze the set of results. In modeling Internet topology, this corresponds to generating multiple graphs to be used in simulation or analytic studies. In this section, we consider three classes of generation methods. The first are *flat random* graph methods, which construct a graph by probabilistically adding edges to a given set of nodes, with no structure amongst the nodes. The second class of generation methods are *regular*, in the sense that they generate graphs with specific structure, for example rings and meshes, and thus have no randomness at all. The third class of models are *hierarchical*, building larger network structure from smaller network components. These methods represent a compromise between the extremes of the flat random and regular methods: they impose some high-level structure, and fill in details at random.[3]

3.1 Four Flavors of Flat Random Methods

The networking literature contains a variety of flat (i.e., non-hierarchical) random methods used to model internetworks. All are variations on the same basic method: a set of vertices is distributed in a plane, and an edge is added between each pair of vertices with some probability. In the **Pure Random** (or simply **Random**) method, this probability is a fixed number p. While the Pure Random method does not explicitly attempt to reflect the structure of real internetworks, it is attractive for its simplicity and is commonly used to study networking problems.

Other methods add edges with a probability that is some function of the distance between the nodes. After the Pure Random Method, perhaps the most common generation method is the **Waxman** method [61], with the probability of an edge from u to v given by:

$$P(u,v) = \alpha e^{-d/(\beta L)}$$

where $0 < \alpha, \beta \leq 1$, d is the Euclidean distance from u to v and L is the maximum distance between any two nodes. An increase in α increases the number of edges in the graph, while an increase in β increases the ratio of long edges to short edges [61]. Several variations on the Waxman method have been proposed [26, 63]; because they are essentially equivalent to the Waxman method, our study includes only the original method.

We propose two new methods that are also intended to capture locality by relating edge probability to distance between vertices. Our **Exponential** method uses:

$$P(u,v) = \alpha e^{-d/(L-d)}.$$

[2] Two refinements of this structure in the present Internet are worth mentioning. First, there may be multiple strata of transit domains, i.e. some transit domains connect only to other transit domains and not to stub domains. This gives rise to a three-level (or even deeper) hierarchy. Second, top-level transit domains are not interconnected pairwise randomly, but rather come together in cliques called exchanges. Such an exchange is implemented by a single network, to which nodes from many transit domains are connected.

[3] Georgia Tech Internet Topology Models (GT-ITM) is a package for generating the flat random and hierarchical models described in this section. It is publically available at http://www.cc.gatech.edu/projects/gtitm/

Method	Edge Probability
Pure Random	p
Waxman	$\alpha e^{-d/(\beta L)}$
Exponential	$\alpha e^{-d/(L-d)}$
Locality	α if $d < r$ β if $d \geq r$

Table 1: Flat random graph methods

so that the probability of an edge approaches zero as the distance between two vertices approaches L. Our **Locality** method partitions the (potential) edges based on length, and assigns a different (fixed) probability for each equivalence class of edge lengths. For the case of two equivalence classes, the parameter r defines the boundary:

$$P(u, v) = \begin{cases} \alpha & \text{if } d < r \\ \beta & \text{if } d \geq r \end{cases}$$

One nice feature of the Locality method is that we have been able to extend some analytic results from the large body of work on pure random graphs [10] to this model; these results deal with properties of the graphs (e.g. connectedness) as the number of nodes becomes large.

To summarize, Table 1 indicates the probability of an edge between two vertices at Euclidean distance d for each flat random method in our study. Note that for most of these methods, the effect of the various parameters on properties of interest is indirect. For example, suppose one is modeling a real network with known average node degree of 4.0. What values of α and β should be chosen in the Waxman method? The Exponential method? We return to such questions in Section 5.

Note also that in each of these methods, every pair of vertices is treated equally with respect to addition of edges. Thus, although it is possible to control the approximate number of edges, it is *not* possible to control the *configuration* of the edges. This has implications for the generation of large *sparsely-connected* graphs using these methods. In particular, for any of the edge probability functions in Table 1, the probability that a graph is connected decreases as number of nodes in the graph increases. As a result, it is difficult to create connected graphs with large numbers of nodes *and* low average node degree. One alternative is to generate partially-connected graphs and then apply various "repair" techniques to connect the graph. Obviously this will produce graphs whose structure is not entirely random, because it has been influenced by the repair process. Another option is to use the random methods to generate smaller graphs that are then connected into a hierarchical structure. (See Section 3.3.)

3.2 Regular Graphs

Regular graphs are often used in analytic studies of algorithm performance because their structure makes them tractable. We consider Linear Chains, Rings, Stars and Meshes; Figure 3 gives an example of each topology with nine nodes. The general structure should be clear from these examples.

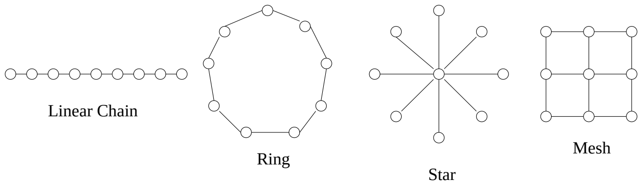

Figure 3: Examples of regular graphs

3.3 Hierarchical Methods

The flat random and regular graph methods represent extremes in the sense that the former offer little control over the structure of the resulting graphs, while the latter are extremely rigid in their structure. Neither captures the hierarchy that is present in real internetworks, though both may reflect some notion of *locality* if certain nodes are more likely to be connected than others. We now describe two methods of creating hierarchical graphs by connecting smaller components together according to a larger-scale structure.

3.3.1 N-Level Method

The N-level hierarchical method constructs a graph by iteratively expanding individual nodes into graphs. Beginning with a connected graph, each node in the graph is replaced by a connected graph. The edges of the original graph are then re-attached to nodes in the replacement graphs.

More precisely, in constructing the flat random graphs, we divide a unit square in the Euclidean plane into equal-sized square sectors, the number of which is determined by a *scale* parameter S; each node in the graph is then assigned to one of the S^2 squares. In constructing an N-level hierarchical graph, the top-level graph is constructed in this fashion using scale parameter S_1. Then each square containing a node is subdivided again, according to the second-level scale parameter (S_2), and a graph is constructed using that top-level square as the unit plane. Figure 4 illustrates the initial iterations of the process. The result is that the scale of the final graph is the product of the scales of the individual levels, and edge lengths are roughly determined by edge level. For example, in a three-level graph, "top-level" edges (i.e., part of the original graph) are typically longer than second-level edges, while the second-level edges are typically longer than third-level edges.

It is possible to define a simple "routing policy" based on the principles outlined in Section 2, and add information to the graph to reflect it. This generally takes the form of an edge metric designed so that the "shortest" path between two nodes, in terms of that metric, obeys the hierarchy constraints. For example, in the N-level method, the path between nodes within the same level-k domain should stay entirely within that domain. We implemented this by associating with each edge a *routing policy weight*, in addition to the Euclidean edge length, to use in constructing policy-based shortest paths. One of the advantages of the N-level method is its simplicity. Because of the way nodes are laid out in the plane with this method, Euclidean edge lengths are a good approximation for policy weights that enforce domain locality. A disadvantage of this method is that the *nodes* are of only one type, and thus paths in these graphs may not have the form described in Section 2.

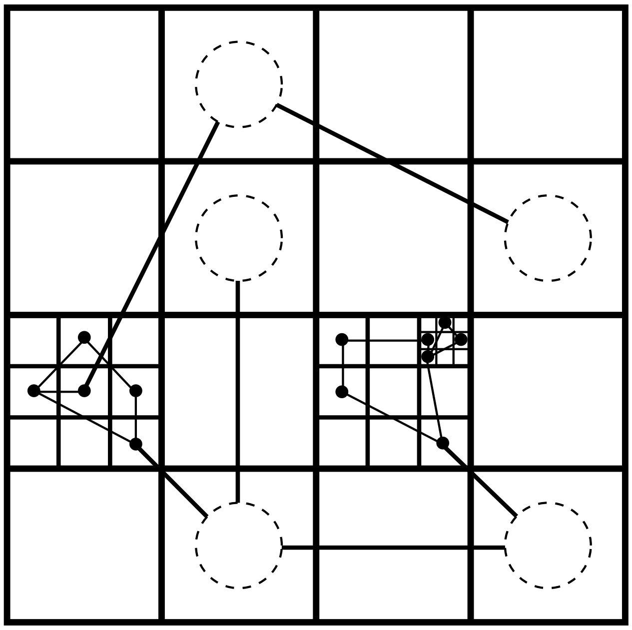

Figure 4: N-Level hierarchical layout

3.3.2 Transit-Stub Method

Our Transit-Stub method is a new way to produce hierarchical graphs, by interconnecting transit and stub domains. We first generate a connected random graph, using any one of the methods discussed earlier; each node in that graph represents an entire transit domain. Each node in that graph is then replaced by another connected random graph, representing the backbone topology of one transit domain. Next, for each node in each transit domain, we generate a number of connected random graphs representing the stub domains attached to that node. Each of these stub domains has an edge to its transit node. Finally, we add some "extra" connectivity, in the form of edges between pairs of nodes, one from a transit domain and one from a stub, or one from each of two different stub domains. The number of "extra" edges of each type is controlled by parameters to the method. Clearly, if the random graphs generated are all connected, this construction results in a connected graph. Doar has independently proposed a graph generation method that has a somewhat similar hierarchical structure to this method [15].

The size of the graph (number of nodes) and the distribution of nodes between transit and stub domains in this method is controlled by the following parameters:

Param.	**Meaning**	**Ex.**
T	number of transit domains	6
N_t	(avg) nodes/transit domain	15
K	(avg) stub domains/transit node	12
N_s	(avg) nodes/stub domain	8
	total nodes	8730

As the table shows, it is not difficult to generate rather large graphs with this method. Moreover, the average node degree of the whole graph depends mainly on the number of edges in the component graphs; if the component graphs are small enough, it is possible to generate large graphs with small average node degrees. In the example above, if the average node degree is 3 for transit backbones and 2.5 for stub domains, the average node degree of the entire graph will be about 2.75.

Note that the parameters given are *averages*. In our implementation, we randomize the numbers of nodes in the transit domains while preserving the average [4]. The values of K and N_s for each transit node and stub domain are also randomized. Our implementation also allows different random graph methods and parameters to be plugged in to generate transit and stub domain subgraphs with different characteristics.

Once the complete graph is constructed, integer edge weights are assigned to the edges so that the hierarchical routing policies of Section 2 are enforced when shortest paths are calculated using the weights. The calculation of these weights is described in the Appendix.

The transit-stub method, like the other methods, assigns each node of the graph to a point in the Euclidean plane, so that a "length" metric may be assigned to each edge. This is done in such a way that nodes in the same domain are "near" each other, but domains may overlap in the plane. In particular, stub domains may overlap with the transit domains to which they are connected. This means that the policy weights assigned to the edges may have little to do with the Euclidean "length" of the edges—unlike graphs generated using the N-level method.

4 Metrics

In this section, we consider several metrics that can be used to compare and evaluate graphs. The metrics fall into two categories: *topological* metrics, which are independent of any particular application, and *application-specific* metrics, which depend on topology and application. In the next section, the effect of method parameters on the topological metrics is discussed; in Section 6, the different generation methods are compared with respect to these metrics. In Section 7, the methods are compared using some metrics specific to multicast routing.

[4] This is done by iteratively choosing a transit domain, and if its current number of nodes exceeds 1, decrementing it and then choosing another random domain and incrementing its number of nodes. This process "spreads out" the values while keeping the mean constant.

4.1 Topological Metrics

We first consider attributes inherent in the graphs themselves; these are useful for characterizing and classifying methods independent of any particular application. Some of these metrics are derived from shortest paths; we consider both "hop" metrics, in which each edge has unit weight, and "length" metrics, in which each edge has weight equal to its Euclidean length. We further consider composite metrics in which the shortest paths are *determined* using one metric, but *evaluated* using a different metric. For example, in the hierarchical methods, we might determine the shortest paths using the routing policy weights, then evaluate the maximum hop count for these (policy-determined) routes between nodes.

For a graph with n nodes and m edges, we consider the following topological properties:

- **Average node degree.** $(2m/n)$.
- **Diameter.** The *diameter* of a graph is the length of the longest shortest-path between any two vertices [11]. Informally, a low diameter generally corresponds to shorter paths. We consider several variations on the diameter, depending on the metric used to construct and evaluate the shortest paths.
  The **hop-diameter** is the length of the longest shortest-path between any two vertices, where the shortest paths are computed and evaluated using hop count as the metric.
  The **length-diameter** is the length of the longest shortest-path between any two vertices, where the shortest paths are computed and evaluated using Euclidean length as the metric.
  The **hop-length-diameter** is a composite metric. The shortest paths are determined using hop count, then measured using Euclidean length. The diameter is the longest Euclidean length amongst the paths found using hop count. This metric is interesting because routing algorithms often use hop count to find paths, but propagation delay is proportional to physical path length (represented by Euclidean distances).
  For the hierarchical methods, the **policy-hop-diameter** uses the routing policy weights to construct the shortest paths, then uses the hop count to evaluate the length of the paths. The diameter is the length of the longest path.
  Similarly, the **policy-length-diameter** uses the routing policy weights to construct the shortest paths, then uses the Euclidean length to evaluate the length of the paths. The diameter is the length of the longest path.
- **Number of biconnected components.** A biconnected component (or bicomponent) is a maximal set of edges such that any two edges in the set are on a common simple cycle [11]. The number of bicomponents is a measure of the degree of "connectedness" or "edge redundancy" in a graph. Generally, a smaller number of bicomponents corresponds to a larger number of paths between nodes in the graph.

We considered a number of other possible metrics, including some that were not scalar (e.g., node degree distributions). We chose to focus on these characteristics because they are quantitative abstractions of some aspects of the structure of the graph, they are relatively easy to measure, and they are informative. They are not necessarily independent. In particular, increasing average node degree correlates with decreasing diameter and fewer bicomponents.

	Edges	Hop-diameter	Bicomps
L. Chain	$n-1$	n	n
Ring	n	$\lfloor n/2 \rfloor$	1
Star	$n-1$	2	n
Mesh	$2(n-\sqrt{n})$	$2(\sqrt{n}-1)$	1

Table 2: Properties of regular graphs

As an example, Table 2 gives selected properties of Linear Chains, Rings, Stars and Meshes, with n nodes. While a Ring and a Chain differ by only one edge, they have substantially different hop-diameter and number of bicomponents. (Removing one edge in a Ring breaks the symmetry.) A Star has the smallest hop-diameter of these four regular graphs. In all cases, the properties are completely determined by the number of nodes n.

4.2 Example Application Metrics

To more closely relate the evaluation of network models to a specific intended use, we also consider two metrics that are relevant to the performance of multicast routing algorithms. Briefly, a multicast routing *instance* is a subset S of the graph nodes designated as multicast group *sources* and another subset R designated as multicast group *receivers*; these two sets may intersect.

For a given graph and instance, a multicast routing algorithm defines a set of *distribution trees*, one per source, which determine the path followed by each packet sent by a source on its way to all the receivers. A distribution tree is generally derived from the shortest-path trees provided by the underlying internetwork (unicast) routing protocol.

For a group with s senders and r receivers $(1 \leq s, r \leq n)$ we measure the following quantities for multicast routing algorithm A:

- **Packet-hops ratio.** This metric considers the packet-hops, that is, the number of edges traversed by multicast packets in the routing trees defined by A, from all sources to all destinations. The actual value of the metric is the ratio of this quantity to the packet-hops used by another routing algorithm that constructs a distribution tree by using a shortest-path tree rooted at each source and including all receivers as leaves.[5]

[5] There are a couple of reasons for measuring the ratio to the shortest-path tree number, rather than the absolute number. For one, the ratio serves to normalize results, allowing comparisons across instances with different numbers of sources and receivers. In addition, shortest-path trees are the basis for the presently-deployed Internet multicast routing algorithm DVMRP [25, 48], thus the ratio allows an evaluation of how algorithm A compares to an accepted standard.

- **Delay ratio.** This metric considers the maximum delay, measured in hops, from any source to any receiver in the distribution trees defined by A. As above, the "raw" value is normalized to get the metric value, by dividing by the maximum delay obtained using a shortest-path algorithm. Note that this ratio is always at least 1.0.

5 Parameter Selection

We have described methods for generating graphs and metrics that can be used to evaluate the properties of a graph. We now proceed to use these metrics to evaluate the graphs generated by each method. To compare the metrics *across* methods, we must have some reasonable way of "normalizing" so that the differences that we observe are (in some sense) inherent to the method. In this section, we describe a methodology for comparing methods that consists of fixing the number of nodes and the average number of edges. For most methods, a target number of edges can be achieved by various combinations of parameter values, so we investigate how the metric values vary with the parameters. We conclude this section by selecting parameters for each method, avoiding the extremes of the parameter space.

5.1 Evaluation Methodology

To compare methods, we normalize by fixing a value for n, the number of vertices, m, the number of edges, and L, the maximum distance between two nodes. Specifically, we select $n = 100$ and $m = 175$; this corresponds to an average node degree of 3.5. The scale of the Euclidean plane in which the nodes of each graph are placed is 100 by 100, thus $L = 100\sqrt{2} \approx 141$. Later in the paper we investigate the effect of changing n and m.

In each method, we explore the combinations of parameters that yield the desired fixed number of edges, while also producing connected graphs with reasonable frequency. (After generating each graph, we check it for connectivity, keeping only those graphs that are connected.) After characterizing the combinations of parameters that can produce connected graphs with the target number of edges, we pick one particular set of parameters for each method and determine the additional properties of interest for graphs generated using these parameters.

5.2 Flat Random Methods

For all of the properties we will examine, the most useful result would be an analytic expression giving the value of the property as a function of the model parameters and the graph size. Unfortunately, most of the methods do not admit a simple analytic expression, even for simple properties. There are two exceptions to this: first, a wide body of work on the properties of (pure) random graphs has been developed, particularly graphs with large numbers of nodes ($n \to \infty$) [10]; second, we have extended a number of these results to the Locality method [9]. For the remaining methods, we rely on empirical results.

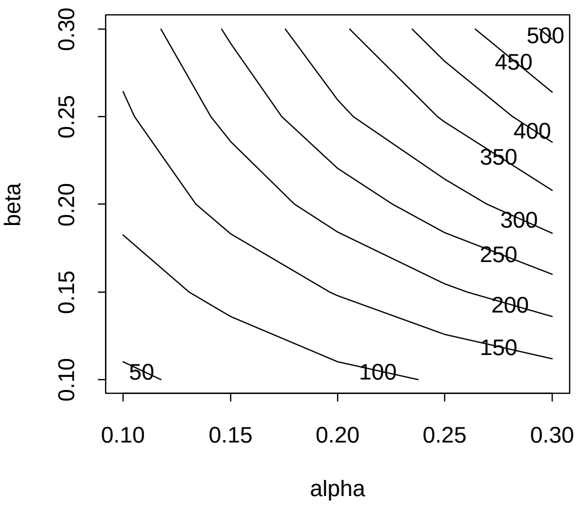

Figure 5: Effect of α and β on number of edges in Waxman method

The expected node degree in the Pure Random method is given by $p(n-1)$, thus $p = .035$ will produce graphs with $m = 175$ edges. The expected degree of a node in the Exponential method is $E[n\alpha e^{-D/(L-D)}] = n\alpha E[e^{-D/(L-D)}]$, where D is a random variable giving the distance between two vertices. The expected node degree in the Exponential model scales linearly with α. Empirical results indicate that the average number of edges is approximately 175 when $\alpha = .06$; about three out of 100 of these graphs are connected.

The relationship between the number of edges and the parameters of the Waxman method is somewhat more complex. Through empirical studies, we have determined that the parameter L has essentially *no effect* on the number of edges, for fixed α and β, because a change in L is accompanied by a change in the distance d between an arbitrary pair of vertices, keeping the quantity d/L essentially constant. The practical implication is that a user of this method can generate vertices in any convenient scale without affecting the graphs that are generated.

For a target number of edges, many combinations of α and β can achieve the target. Figure 5 shows the contour lines of equi-number of edges as a function of α and β, for $n = 100$.

We explore the effect of the parameters α and β by selecting four combinations that produce $m = 175$ edges. We generate 100 connected graphs for each combination, and measure the metrics described earlier. Figure 6 shows the comparison using a box plot to indicate the median over the 100 values (with a white line), the 25% and 75% range boundaries (with a box), the 5% and 95% range boundaries (with "whiskers") and any outliers with single lines. The

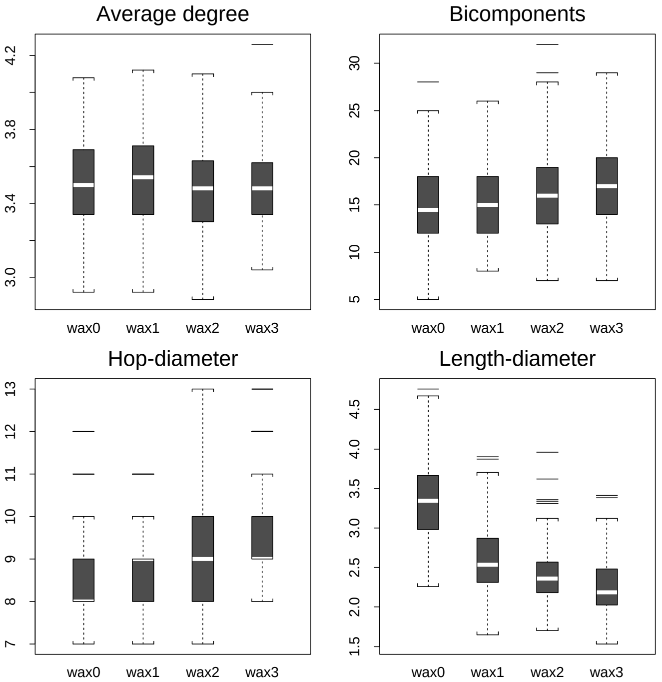

Figure 6: Effect of parameters in Waxman method

average node degree plot serves as a check that the graphs achieve the target of 175 edges (i.e., average node degree 3.5). The other properties are the number of bicomponents and the diameter measured in both hops and length. The length-diameter has been normalized by dividing the Euclidean length by the scale.

We will use box plots extensively to present scalar metrics, since they neatly summarize information about the distribution, and also support comparisons across methods. The reader should be careful, however, in interpreting the data for discrete metrics. For example, the hop-diameter for the `wax0` method is almost always either 8 or 9; it does not take on continuous values as one might infer from the box plot.

The value of α ranges from 0.1 (`wax0`) to 0.4 (`wax3`). The β values are chosen to achieve the target of 175 edges: for $\alpha = 0.1$, $\beta = 0.28$; for $\alpha = 0.4$, $\beta = 0.1$. As α increases (and thus β decreases), the graphs have more short edges, leading to longer hop-diameter, shorter length-diameter and slightly more bicomponents. For comparisons to other methods, we select $\alpha = .2$ and $\beta = .15$.

For the Locality model, we determined that a choice of $\alpha = .2$, $\beta = .005$ and $r = .25 \times L$ will produce an average node degree of 3.5. We have also experimented with variations in α and β in the Locality method. We observed the same general trends as in the Waxman method: more short edges (higher α) results in longer hop-diameter, shorter length-diameter and more bicomponents. The differences in length-diameter can be extreme in the Locality method, since α and β can be chosen to force the extremes of *no* edges of length greater (or less) than r.

We summarize the choice of parameters for each method in Table 3.

Method	Parameters
Random	$p = .035$
Waxman	$\alpha = .2, \beta = .15$
Exponential	$p = .06$
Locality	$\alpha = .1, \beta = .005, r = (.25 \times L) = 35.35$

Table 3: Selected parameters for each flat random method

5.3 Hierarchical Methods

One of the advantages of the hierarchical methods is their ability to generate *large* graphs efficiently, while maintaining low average node degree. The flat methods require the average node degree to grow as n grows in order to generate connected graphs with reasonable efficiency. The increase in average node degree in turn affects other parameters such as diameter and number of bicomponents. In contrast, the formula for the average node degree of a transit stub graph (ignoring "extra" edges) is

$$\frac{2(E_t + (1 + E_s N_s)K)}{1 + KN_s}$$

where E_t and E_s are the edge densities (number of edges per node) of the transit and stub domains, respectively.[6] Thus, it is possible to parameterize the random graph generation methods to achieve almost any overall average node degree.

Consider now the effect of varying the number and sizes of the domains for the transit-stub and N-level methods, with $N = 2$. Note that there is no obvious way to "normalize" the edge probabilities when the number and size of domains is varied.[7] Thus, we make observations about the general trends that hold over large portions of the edge probability space; some extreme values of edge probability may generate graphs that do not follow these general trends.

In the transit-stub method, we vary the number and size of transit domains, while keeping the number and size of stubs fixed, then vary the number and size of stub domains while keeping the number and size of transits fixed. For example, Figure 7 shows metrics for three configurations of 200-node transit-stub graphs with average node degree approximately 3.5. Note that for this comparison, we are using the diameter metrics that construct the routes using the routing policy weights. The configurations are as follows:

[6] In our implementation, the randomization of the number of nodes in the domains tends to increase the number of edges slightly over what the above formula predicts. This is because the number of possible edges in a graph grows quadratically with the number of nodes. Thus, although the mean number of *nodes* per domain is preserved by the randomization process, the number of additional edges contributed by domains with a greater-than-average number of nodes will be greater than the deficit left by domains with fewer-than-average edges. If parameters are always chosen so that domain size is limited, this randomization effect will also be limited.

[7] One possibility is to keep constant the ratio of the number of intra- to inter- domain edges. However, one can easily come up with other plausible alternatives. We are not convinced that there is a "best" way to do this normalization.

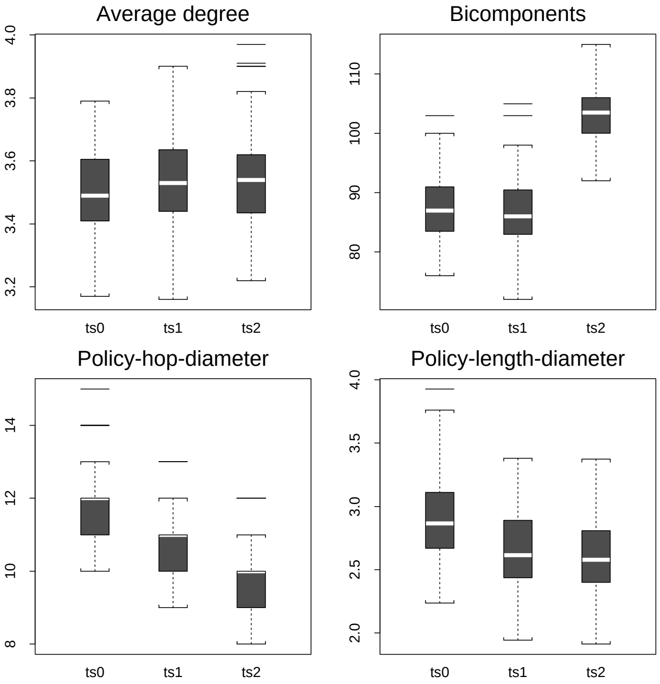

Figure 7: Varying domains in Transit-Stub graphs

Config	# xit domain	nodes/ xit	# stubs/ xit	nodes/ stub
`ts0`	2	4	4	6
`ts1`	1	8	4	6
`ts2`	2	4	6	4

If we call `ts0` the baseline, then `ts1` corresponds to fewer/larer transit domains and `ts2` corresponds to more/-smaller stub domains.

From this example, and others, we observe the following:

- More/smaller transit domains yields more bicomponents and *larger* hop and length diameter. This is consistent with intuition: more and smaller transit domains result in longer paths to get from one stub domain to another.
- More/smaller stubs yields more bicomponents and *smaller* hop and length diameter. Thus the effect on diameters is opposite for stubs than for transit domains. This is also consistent with intuition: smaller stub domains result in shorter paths within the stub, while the length of the paths across the transit domains remain constant.

The 2-level method has just one type of domain. More and smaller domains in this model have the same effect as more/smaller stubs; that is, more/smaller domains lead to more bicomponents and smaller hop and length diameter.

We selected the following parameters to use in comparing the hierarchical methods to the flat random methods: each transit-stub graph has 1 transit domain of 4 nodes, 2 stub domains per transit node, and 12 nodes per stub domain. None of these graphs have any "extra" transit-stub or stub-stub edges, so each stub domain connects to exactly one transit domain. Each 2-level graph has 10 domains with an average of 10 nodes each. In both hierarchical methods, the pure random method was used to generate all of the domain graphs.

6 Method Comparisons

In this section, we compare one graph generation method to another. We pay particular attention to the metrics that serve as *discriminators*, demonstrating one method to be clearly different from another. Our comparison is based on 100 connected 100-node graphs for each method, using parameters chosen (see Section 5) so that the average of the graph average node degrees is 3.5 across the graphs of each type.

6.1 Graph Properties

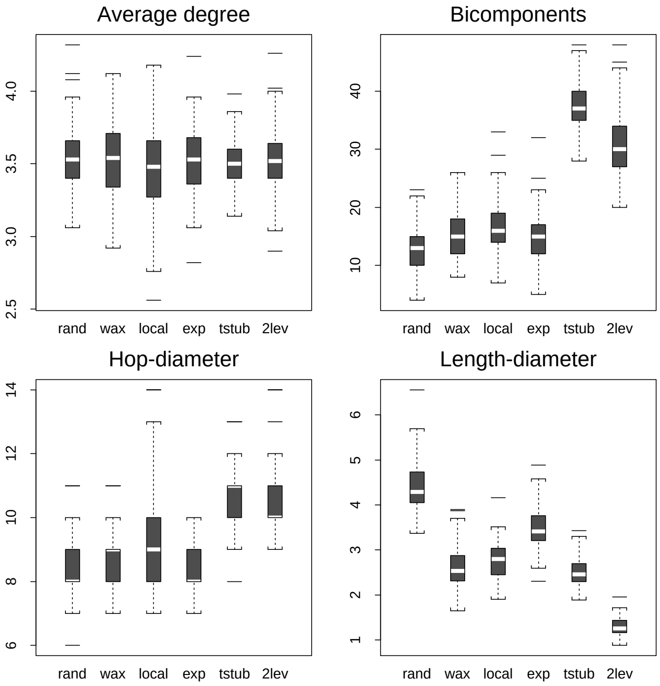

Figure 8: Comparison of flat random and hierarchical methods

Figure 8 shows box plots of topological metrics for the four random methods (rand, wax, local, exp) and the two hierarchical methods (tstub, 2lev). The salient features of these plots are the following:

- The most significant difference in the flat methods is in the length-diameter. For the parameters chosen, the Pure Random method is clearly different from the other models, with much longer length-diameter. The Random method is insensitive to edge length when adding edges to a graph, while the other methods (with the given parameters) favor shorter edges over longer edges. The Random method is therefore more likely than the other methods to have long edges and long paths.

- The Exponential method is next most likely to have long paths. The Exponential method does include length in the probability of an edge, however for the parameters chosen, the probability falls off more slowly with increasing edge length than in the other methods.
- The hierarchical methods differ from the flat methods most significantly in the bicomponents metric, with nearly twice as many bicomponents, on average, in hierarchical graphs. The hierarchical graphs also tend to have longer hop-diameter and shorter length-diameter. The hierarchical structure explains the diameter results: edges are mostly constrained to be (short) intradomain edges, resulting in longer hop-based paths, but more direct (thus, shorter) length-based paths.
- The two types of hierarchy reflected in Transit-Stub and 2-Level methods differ from one another on several metrics, with Transit-Stub graphs tending to have more bicomponents and longer length-diameter. For these parameters, the transit domains serve to "separate" stub domains, leading to more bicomponents and longer length-based paths.

To further compare the two hierarchical methods, we look at a larger topology.[8] We consider graphs of 600 nodes, with average node degree 4.1. We normalize across the two hierarchical methods by fixing the number and size of the domains to 75 domains of 8 nodes each. In the Transit-Stub method, three of these domains are transit domains and 72 are stub domains, with an average of three stubs per transit node. We further select the edge probability parameters so that the average number of "lowest" level intradomain edges are approximately the same. Thus, the number of intra-stub edges in Transit-Stub is approximately equal to the number of intra-neighborhood edges in 2-Level.

Figure 9 shows properties for the hierarchical methods, including diameter using policy weights to construct the paths. Two Transit-Stub results are shown: those labeled tstub have no extra edges, while those labeled tstub-ex have five extra Transit-Stub edges and five extra Stub-Stub edges. These results show that Transit-Stub graphs tend to have more bicomponents than 2-level graphs, though as expected the difference is reduced as extra edges are included. The policy-hop-diameter is essentially the same for all three results, but the policy-length-diameter is more than 50% larger in the 2-level method. The difference in policy-length-diameter can be attributed to the lack of a "backbone" in the 2-level graphs. Routes in a 2-level graph may need to traverse multiple neighborhoods to get from source to destination, while the Transit-Stub routes are more likely to make good use of the backbone to construct shorter routes. Further, the policy-length-diameter is insensitive to the addition of extra edges; the extra edges will only reduce the diameter of a given graph if they happen to provide an alternative, shorter route between nodes that are currently separated by the longest shortest path.

[8] Note that the hierarchical methods are well-suited to the generation of large topologies; indeed $n = 100$ nodes is somewhat too small to generate an "interesting" hierarchical graph.

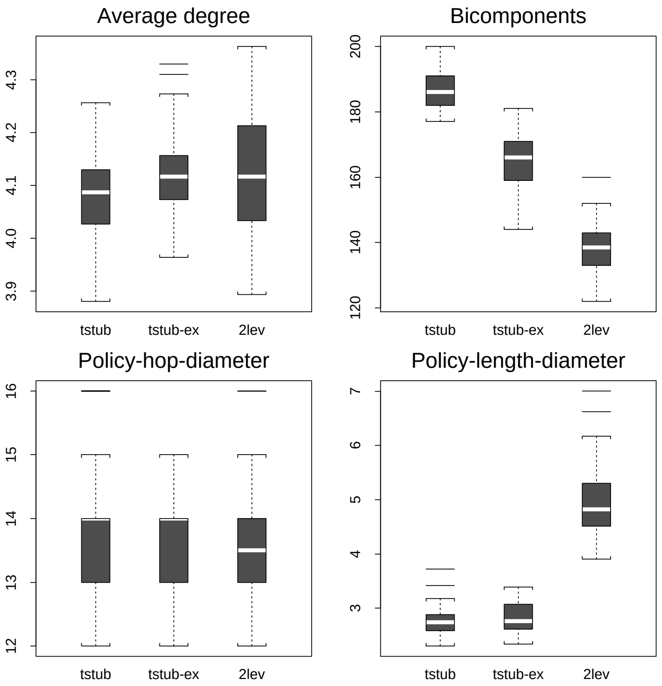

Figure 9: Hierarchical methods

6.2 Statistical Analysis

We now turn to statistical analysis of the topological metric data for the models. Our aim is to go beyond the qualitative, visual comparisons of the previous section to provide a rigorous test for determining the extent to which two methods differ with respect to the characteristics of the graphs they generate. We quantify the similarity of the graphs generated with different methods by performing pairwise comparisons of the methods for each metric. Given two methods and a metric, these comparisons are designed to answer the following questions:

- Does the metric data come from the same distribution?
- What is the probability that a random instance generated by one method has a smaller value for the metric than a random instance from the other method?

6.2.1 Statistical Methods

To answer the first question, we utilize the Kolmogorov-Smirnov two-sample test (KS test) [32] which tests the hypothesis that two independent samples are drawn from the same population. For a particular metric, if the distribution of all methods appears identical, then the selection of the method for that metric is less important; conversely, for methods whose metric values are not the same, extra consideration must be given to the appropriate method to use. To determine method interrelationships, we perform a two-sided KS test with $\alpha = 0.05$ for each metric, pairwise over the generation methods. Thus, the null hypothesis is that the samples are drawn from the same distribution, and the test will indicate whether the null hypothesis can be rejected with confidence level $1 - \alpha = 0.95$.

Distribution testing allows only a yes/no answer about the similarity of the metric value distributions for two methods. We want to extend our analysis to answer the question of *how* similar or dissimilar the graphs generated by the methods are. In addition we want to derive a measure of confidence in conclusions of metric value relationships between methods. That is, how confident are we in concluding that the metric value for one method is less than the metric value for another? To determine this, we assess the probability that a randomly generated instance from method F has a lower value of metric M than a randomly generated instance from method G. The closer this *intermetric* probability is to 0.5, the more similar the methods are, relative to metric M. A high (low) probability indicates that the value for metric M of an instance of F is likely to be lower (higher) than the value for G.[9]

Let $f(x)$ and $g(x)$ be the probability distribution function of generation method F and G, respectively, for metric M. For any random instance of graph models F and G with metric value f and g, respectively, the probability that f is at most g is given by:

$$P\{f \leq g\} = \int_{-\infty}^{\infty} (\int_{-\infty}^{y} f(x)dx)g(y)dy$$

We used the measured distribution data from the experiments in the previous section to numerically perform the integration.

6.2.2 Results

For conciseness, we combine the results of the KS-Test and intermetric probability into a single table format. We opt for a pictographic representation of the intermetric probability data, using a "pie" that is proportionally more filled in to represent larger probabilities.[10] For the KS-Test, we add an asterisk to those tabular entries that do not reject the null hypothesis of two sample sets having the same distribution.

Each table contains results comparing a variety of generation methods for a particular metric. Each row corresponds to a particular graph size, and each column corresponds to a particular pair of methods.[11] Within a column, two values for average node degree are considered, 3.5 and 5.0. Each entry in the table gives the pie-chart representing the intermetric probability. Pie charts with an asterisk indicate that the two methods being compared have equivalent metric value distributions according to the KS test. N/A entries indicate that we did not perform experiments for this particular graph size and node degree.

[9] Other methods exist for comparison of sample distribution (e.g. χ^2 [49]). We chose to use a method that is somewhat more simple to evaluate and is not sensitive to factors such as selection of bin size.

[10] Clearly some information is lost by this representation, however we believe that actually makes it easier to reach meaningful conclusions about the results.

[11] The columns are labeled Pr(X < Y) where X and Y are the first letter of the method names being compared. For example, Pr(R < L) means that the table entry gives the intermetric probability that Random is less than Locality.

As an example, consider the top entry in the first column of Table 4. For this entry, 100 Random graphs are compared to 100 Locality graphs, all of average node degree 3.5. The pie indicates that a Random graph has approximately a 62.5% chance of having a smaller hop-diameter than a Locality graph, assuming both graphs have 50 nodes and an average node degree of 3.5. The lack of an asterisk indicates that the KS-Test rejected the hypothesis that 50 node, 3.5 average node degree Random and Locality graphs have the same distribution of hop-diameter.

Tables 4, 5 and 6 give the flat-to-flat comparison for three diameter metrics: the hop-diameter, the length-diameter and the hop-length-diameter. Recall that the hop-length-diameter is determined by first constructing shortest paths using hop count as the distance metric, then determining the length of the longest resulting path using the Euclidean length of the edges as the length metric. Hop-length-diameter is useful since routing algorithms often rely on hop count to construct paths, but Euclidean length is proportional to propagation delay on the path.

Tables 7 and 8 compare the flat methods to the hierarchical methods for the hop-diameter and the length-diameter, with Transit-Stub denoted T and 2-level denoted N. For the hierarchical-to-hierarchical comparisons, we only use 600 node graphs with node degree 4.2. Table 9 compares the Transit-Stub and 2-level hierarchical methods using the hop- and length-diameters, in addition to the bicomponent metric.

The interesting observations which can be made from these tables are:

- We confirm, using statistical methods, some of the observations we made in Section 6.1 based on visual inspection of the boxplot data. For example, the differences between the flat methods are more profound for the length-diameter measure than the hop-diameter measure: the pies in Table 5 that are mostly empty or nearly entirely filled, while the pies in Table 4 are closer to half-filled.
- The comparisons tend to be preserved with changes in graph size and average node degree, for the parameters selected.
- The comparison of the flat methods based on length-diameter is strikingly similar to the comparison based on hop-length-diameter, with both metrics revealing significant differences across methods.
- The ordering of the methods differs significantly with respect to length-based and hop-based evaluation metrics. In fact, the most likely ordering is generally inverted in going from one to the other. For example in Table 7 (hop-diameter) all entries are close to 1, while in Table 8 (length-diameter) almost all are close to 0. The reason for the inversion is the difference in probability of long paths for the various generation methods. That is, graphs with the most long edges will most likely have the smallest hop-diameter; however, graphs with mostly long edges will most likely not yield the most direct Euclidean path.
- There are significant differences between the 2-Level

method and the Transit-Stub method as demonstrated in the pair-wise comparisons of Table 9. Note again the inversion of probabilities for hop and length-based metrics.

Graph Size	Pr(R < L) 3.5	5.0	Pr(R < W) 3.5	5.0	Pr(R < E) 3.5	5.0
50	◕	◑	◔ *	◔ *	◔ *	◔ *
100	◕ *	◕	◑	◑	◑ *	◔ *
200	N/A	◕	N/A	◕	N/A	◔ *
Graph Size	**Pr(L < W) 3.5**	**5.0**	**Pr(L < E) 3.5**	**5.0**	**Pr(W < E) 3.5**	**5.0**
50	◔	◔	◔	◔	◔ *	◔ *
100	◔ *	◔	◔	○	◔ *	◔ *
200	N/A	◕	N/A	◑	N/A	◔

Table 4: Hop-Diameter for Flat-to-Flat Comparison

Graph Size	Pr(R < L) 3.5	5.0	Pr(R < W) 3.5	5.0	Pr(R < E) 3.5	5.0
50	○	○	○	○	◔	◔
100	○	○	○	○	○	◔
200	N/A	○	N/A	○	N/A	◔
Graph Size	**Pr(L < W) 3.5**	**5.0**	**Pr(L < E) 3.5**	**5.0**	**Pr(W < E) 3.5**	**5.0**
50	◕	◕	◕	◕	◕	◕
100	◑	◑ *	●	◕	●	◕
200	N/A	◔	N/A	◕	N/A	●

Table 5: Length-Diameter for Flat-to-Flat Comparison

Graph Size	Pr(R < L) 3.5	5.0	Pr(R < W) 3.5	5.0	Pr(R < E) 3.5	5.0
50	○	○	○	○	◔	○
100	○	○	○	○	○	○
200	N/A	○	N/A	○	N/A	○
Graph Size	**Pr(L < W) 3.5**	**5.0**	**Pr(L < E) 3.5**	**5.0**	**Pr(W < E) 3.5**	**5.0**
50	◕	◕	●	◕	◕	◕ *
100	◑ *	◑ *	●	◕	●	●
200	N/A	○	N/A	◕	N/A	●

Table 6: Hop-Length-Diameter for Flat-to-Flat Comparison

Graph Size	Pr(R < T) 3.5	5.0	Pr(R < N) 3.5	5.0	Pr(L < T) 3.5	5.0	Pr(L < N) 3.5	5.0
100	●	●	●	●	◕	●	◕	●
200	N/A	●	N/A	●	N/A	●	N/A	◕
Graph Size	**Pr(W < T) 3.5**	**5.0**	**Pr(W < N) 3.5**	**5.0**	**Pr(E < T) 3.5**	**5.0**	**Pr(E < N) 3.5**	**5.0**
100	◕	●	●	●	●	●	●	●
200	N/A	●	N/A	◑ *	N/A	●	N/A	◕

Table 7: Hop-Diameter for Flat-to-Hierarchical Comparison

Graph Size	Pr(R < T) 3.5	5.0	Pr(R < N) 3.5	5.0	Pr(L < T) 3.5	5.0	Pr(L < N) 3.5	5.0
100	○	○	○	○	◑	◕	○	○
200	N/A	○	N/A	○	N/A	◑	N/A	○
Graph Size	**Pr(W < T) 3.5**	**5.0**	**Pr(W < N) 3.5**	**5.0**	**Pr(E < T) 3.5**	**5.0**	**Pr(E < N) 3.5**	**5.0**
100	◑ *	◕	○	○	○	◔	○	○
200	N/A	◕	N/A	◔	N/A	○	N/A	○

Table 8: Length-Diameter for Flat-to-Hierarchical Comparison

7 Multicast Routing

Comparisons of methods using topological metrics has provided evidence of differences between methods; however, ultimately it is important to know if these differences affect performance for real networking problems. That is, how do methods compare for problems of interest? To determine this effect, we consider the problem of multicast routing. We assume the reader is familiar with the concepts of multicasting, and center-based (or shared-tree) multicast routing. (See [16] for details). Many different generation methods have been used to study multicast routing algorithms, including the Waxman method [61] and well-known topologies such as the ARPAnet [63]. Thus, it is important to determine the effect, if any, the method has on results.

7.1 Multicast Routing

Many metrics exist for evaluating multicast routing algorithms and policies. For simplicity, we consider the packet-hops and maximum delay, as defined in Section 4. In our experiments, we consider ten graphs, each 200 nodes with average node degree 5.0, generated by three methods (Random, Exponential, and Transit-Stub). We chose these particular methods to have representation that include the pure random method, a flat method that is sensitive to edge length, and a hierarchical method.

Hop-diameter	Length-diameter	Bicomponents
◔ *	●	○

Table 9: Pr(T < N): Hierarchical Comparison

To test the effect of graph type on performance, we perform a series of "runs" on each graph: each run consists of generating a multicast instance with 1 to 50 sources and 1 to 50 receivers. The total number and identity of the sources and receivers are chosen randomly. After an instance has been generated, we evaluate the performance (packet-hops and maximum delay) with each node as the center of the shared tree. We record the performance for the optimal center, as well as the performance averaged over all centers. Finally, we normalize the optimal and average performance values by taking the ratio to the performance of the shortest-path multicast routing trees for the instance. For each graph, we construct 100 such runs. Within a method, we combine the results of the 100 runs over each of 10 graphs, to get 1000 data points per method.

7.2 Results

Figure 10 gives the box plots for the ratio of the optimal and average center performance to the shortest path performance for packet-hops and maximum delay. Note that while the average center ratios are somewhat similar for all methods, the optimal center ratios differ significantly between the flat and hierarchical methods in the following manner:

- For optimal packet-hops and delay, the means differ between the Random and Transit-Stub by 13% and between the Exponential and Transit-Stub by 2%; however, the means for optimal packet-hops and delay between the flat graphs only differ by 1%.
- The packet-hops and delay results for the Transit-Stub graphs exhibit a much tighter distribution than either flat graph.
- The hierarchical variance is an order of magnitude less that either flat variance for the optimal measures and half for the average measures. Also, note that the flat graphs' 0-50th percentile ranges correspond to the entire range of the Transit-Stub graphs. This indicates that, at least for these measures, fewer Transit-Stub graphs may be needed to achieve a significant sample size in analyzing results from the transit-stub graphs than from the flat graphs.
- Using mean equivalence hypothesis testing (t-test [47]) with $\alpha = 0.05$, we determine the means of the flat methods for the average packet-hops and delay measures to be equivalent, but the means of either flat method compared to the Transit-Stub method are not equivalent. Note that our large sample size (1000 graphs/runs) relaxes the need for normality in this type of hypothesis testing [45].

Figure 11 shows the distribution of multicast performance for each metric. Note that, as expected, the frequency distributions for the two flat methods are very similar, and the Transit-Stub frequency distribution is dissimilar to the flat methods. Also note that the frequency distribution for the Transit-Stub method is tightly centered around 1.0 for both optimal metrics. Without more extra Transit-Stub or Stub-Stub edges, the structure of the Transit-Stub graph limits the routing tree possibilities; therefore, the optimal center routing tree is similar to the shortest path routing tree.

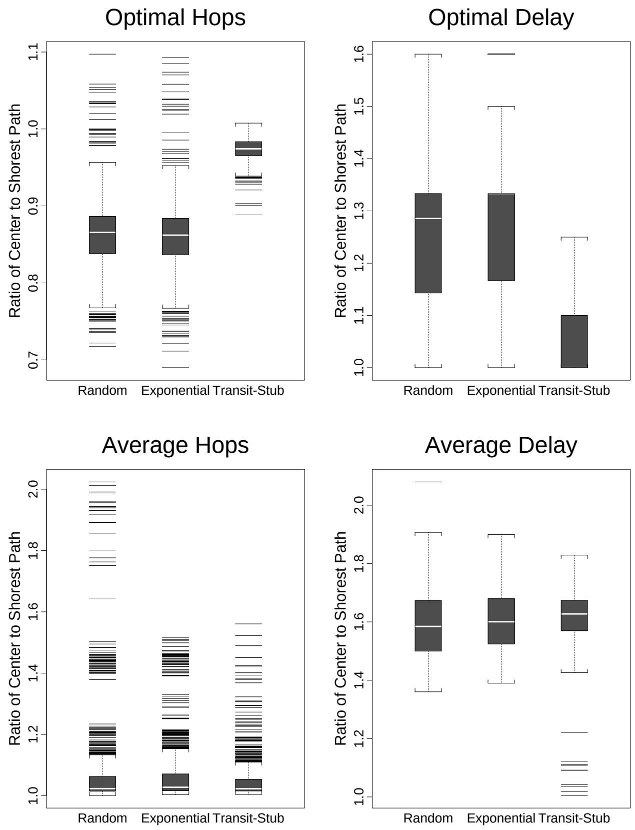

Figure 10: Multicast Performance

This demonstrates that generation method does have a significant effect on performance in an actual networking problem. Specifically, problems involving optimality should pay close attention to the method selected because the method affects relative performance as well as performance variability.

8 Conclusions and Future Work

We set out to examine and improve upon methods for generating graphs to model Internet topology. We have accomplished the following:

- Surveyed and analyzed the behavior of graph generation methods commonly used in studies of networks. The literature contains numerous random and regular topologies used to study networking algorithms, with little guidance for the user of network models. We have outlined the role of the parameters in each method and included some plots that can be used for parameter lookup by users of these methods.
  We conclude that the Pure Random method is significantly different from the three other random methods. The other methods can all be parameterized to

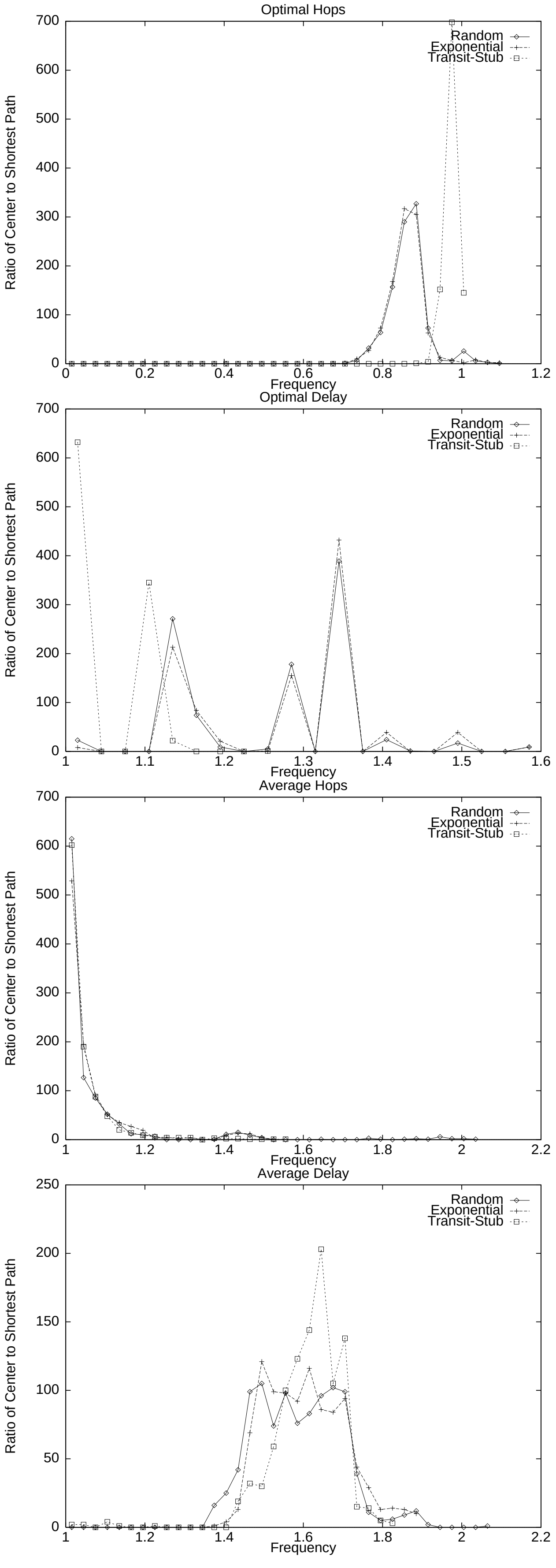

Figure 11: Multicast Performance Distribution

favor short edges over long edges. The Pure Random method has no such control, therefore it generates edges that are (comparatively) longer, leading to longer paths (in terms of Euclidean distance). Given the tendency towards locality in internetworking connectivity, methods that include edge length in determining edge probability are more realistic. For the parameters we chose, the Exponential method is next most likely to have long edges; the other two methods are similar in all properties we considered.

- Identified a fundamental limitation in all of the random graph methods.

  We found that it is not practical to generate even moderate-sized random graphs (e.g., $n = 1500$) that are connected and have realistic average node degrees (e.g., less than about 6), without resorting to methods of repairing unconnected graphs. Based on what we know of real topologies, this limitation is significant, especially if a model is to be used to obtain quantitative results that are valid for the Internet.
- Developed a hybrid generation method, the Transit-Stub method, capable of creating large graphs by composing smaller random graphs.

  By imposing a structure resembling the administrative structure of the Internet, the Transit-Stub method allows creation of large random graphs having realistic average node degree. Moreover, edge weights can be assigned to these graphs in such a way that intra- and interdomain paths in the graph behave in a realistic manner. (See the Appendix for details on the assignment of edge weights.) Finally, the Transit-Stub method allows relatively direct control over attributes such as hop diameter and average node degree.
- Compared flat random and hierarchical methods based on quantitative measures, including topological metrics and metrics specific to multicast routing.

As the Internet continues to grow in size and importance, realistic models of network topology will be critical for meaningful assessment of all kinds of algorithms and policies. One goal of our future work is to understand how measurements taken on modest-sized graphs can be *scaled* to apply to much larger networks. Work is in progress to establish a public repository of information about real network topologies, including both graph models and measurements. The graph-generating tools described in this paper are already in place and being used by various research groups.

Acknowledgments

The authors would like to thank the anonymous reviewers for their helpful comments.

References

[1] M. Ahamad. *Multicast Communication in Distributed*

Systems. IEEE Computer Society Press Technology Series, 1990.

[2] C. Alaettinoğlu and A. U. Shankar. An approach to hierarchical and inter-domain routing with on-demand tos and policy resolution. In *Proceedings of 1993 International Conference on Network Protocols, San Francisco*, pages 72–79, October 1993.

[3] B. Awerbuch, O. Goldreich, D. Peleg, and R. Vainish. A trade-off between information and communication in broadcast protocols. *Journal of the ACM*, 37:238–256, 1990.

[4] A. Ballardie, P. Francis, and J. Crowcroft. Core based trees. *Proceedings of ACM SIGCOMM '93*, 1993.

[5] J. Behrens and J.J. Garcia-Luna-Aceves. Distributed, scalable routing based on link-state vectors. *Proceedings of ACM SIGCOMM '94*, pages 136–147, 1994.

[6] N. Belkeir and M. Ahamad. Low-cost algorithms for message delivery in dynamic multicast groups. In *IEEE Ninth International Conference on Distributed Computing*, pages 110–117, June 1989.

[7] J. Bellamy. *Digital Telephony*. John Wiley, 1991.

[8] K. Bharath-Kumar and J. M. Jaffe. Routing to multiple destinations in computer networks. *IEEE Transactions on Computers*, COM-31:343–351, Mar 1983.

[9] S. Bhattacharjee, K. Calvert, and E. Zegura. Extending random graph theory to topologies with hierarchy and locality (in preparation). Technical report, College of Computing, Georgia Tech, 1997.

[10] B. Bollobás. *Random Graphs*. Harcourt Brace Jovanovich, 1985.

[11] J.A. Bondy and U.S.R. Murty. *Graph Theory with Applications*. North-Holland, 1976.

[12] L. Brakmo, S. O'Malley, and L. Peterson. TCP Vegas: New techniques for congestion detection and avoidance. *Proceedings of ACM SIGCOMM '94*, pages 24–35, 1994.

[13] R. Cahn, P.C. Chang, P. Kermani, and A. Kershenbaum. INTREPID: An integrated network tool for routing, evaluation of performance, and interactive design. *IEEE Communications Magazine*, pages 40–47, July 1991.

[14] K. Calvert, R. Madhavan, and E. W. Zegura. A comparison of two practical multicast routing schemes. Technical report, Georgia Tech, College of Computing, GIT-94-25.

[15] K. Calvert, E. Zegura, and M. Doar. Modeling Internet topology. *IEEE Communications Magazine*, June 1997.

[16] Kenneth L. Calvert, Ellen W. Zegura, and Michael J. Donahoo. Core selection methods for multicast routing. In Kia Makki and Niki Pissinou, editors, *Proceedings of the ICCCN '95*, pages 638–642. IEEE, IEEE Computer Society Press, Sept. 1995.

[17] W. Cheswick. *Firewalls and Internet Security: Repelling the Wily Hacker*. Addison-Wesley, 1994.

[18] D. Clark, S. Shenker, and L. Zhang. Supporting real-time applications in an integrated services packet network: Architecture and mechanism. *Proceedings of ACM SIGCOMM '92*, pages 14–26, 1992.

[19] David Clark. Policy routing in internet protocols. Internet Request for Comments 1102, May 1989.

[20] S. E. Deering. Host extensions for ip multicasting. Internet Request for Comments 1112, August 1988.

[21] S. E. Deering. *Multicast Routing in a Datagram Internetwork*. PhD thesis, Stanford University, California, USA, 1991.

[22] S. E. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu, and L. Wei. An architecture for wide-area multicast routing. In *Proceedings of ACM SIGCOMM '94*, pages 126–135, 1994.

[23] S. E. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu, and L. Wei. Protocol independent multicast (PIM): Motivation and architecture. Working Draft, March 1994.

[24] S. E. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu, and L. Wei. Protocol independent multicast (PIM), sparse mode protocol specification. Working Draft, March 1994.

[25] Stephen E. Deering and David R. Cheriton. Multicast routing in datagram internetworks and extended lans. *ACM Transactions on Computer Systems*, 8(2):85–110, May 1990.

[26] Matthew Doar and Ian Leslie. How bad is naive multicast routing? In *Proceedings of IEEE INFOCOM '93*, pages 82–89, 1993.

[27] C. Alaettinoğlu et al. Introducing MaRS, a routing testbed. *Computer Communication Review*, 22(1):95–96, January 1992.

[28] H. Fowler and W. Leland. Local area network traffic characteristics, with implications for broadband network congestion management. *IEEE J. Selected Areas in Communications*, pages 1139–1149, 1991.

[29] E. Fuchs and P. Jackson. Estimates of distributions of random variables for certain computer communications traffic models. *Communications of the ACM*, 13(12):752–757, December 1970.

[30] F. Harary. The maximum connectivity of a graph. *Proc. Nat. Acad. Sci. U.S.A.*, pages 1142–1146, 1962.

[31] A. Heybey. The network simulator. Technical report, MIT, 1990.

[32] Joseph E. Hill and August Kerber. *Models, methods, and analytical procedures in education research*. Wayne State University Press, 1967.

[33] M. Imase and B. Waxman. Dynamic Steiner tree problem. *Siam J. on Discrete Math.*, 3:369–384, 1991.

[34] R. Jain. *The Art of Computer Systems Performance Analysis*. John Wiley and Sons, Inc., 1991.

[35] R. M. Karp. Reducibility among combinatorial problems. In R. A. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972.

[36] S. Keshav. REAL: A network simulator. Technical report, Department of Computer Science, UC Berkeley, 88/472.

[37] S. Khuller, B. Raghavachari, and N. Young. Balancing minimum spanning and shortest path trees. In

Symposium on Discrete Algorithms (SODA), 1993.

[38] D. Knuth. *The Stanford GraphBase: A Platform for Combinatorial Computing.* Addison-Wesley, 1994.

[39] V. P. Kompella, J. C. Pasquale, and G. C. Polyzos. Multicast routing for multimedia communication. *IEEE/ACM Transactions on Networking*, 1, 1993.

[40] L. Kou, G. Markowsky, and L. Berman. A fast algorithm for Steiner trees. *Acta Informatica*, 15:141–145, 1981.

[41] W. Leland, M. Taqqu, W. Willinger, and D. Wilson. On the self-similar nature of Ethernet traffic. *Proceedings of ACM SIGCOMM '93*, pages 183–193, September 1993.

[42] MaRS Project, Dept. of CS, University of Maryland. *MaRS Version 1.0 Programmer's Manual.*

[43] MaRS Project, Dept. of CS, University of Maryland. *MaRS Version 1.0 User's Manual.*

[44] D. Mitzel and S. Shenker. Asymptotic resource consumption in multicast reservation styles. *Proceedings of ACM SIGCOMM '94*, pages 226–233, 1994.

[45] Douglas C. Montgomery. *Design and Analysis of Experiments.* John Wiley & Sons, third ed. edition, 1991.

[46] J. Moy. MOSPF: Analysis and experience. Internet Draft, July 1993.

[47] Bernard Ostle and Richard W. Mensing. *Statistics in Research.* Iowa State University Press, third edition, 1975.

[48] C. Partridge, D. Waitzman, and S. Deering. Distance vector multicast routing protocol. Internet Request for Comments 1075, November 1988.

[49] V. Paxson. Empirically derived analytic models of wide-area TCP connections. *IEEE/ACM Transactions on Networking*, 2(4):316–336, August 1994.

[50] V. Paxson and S. Floyd. Wide-area traffic: The failure of poisson modeling. In *Proceedings of ACM SIGCOMM '94 Symposium*, pages 257–268, August 1994.

[51] D. Peleg and E. Upfal. A trade-off between space and efficiency for routing tables. *Journal of the ACM*, 36:510–530, 1989.

[52] V. J. Rayward-Smith and A. Clare. On finding Steiner vertices. *Networks*, 16:283–294, 1986.

[53] J. Rekhter. Forwarding database overhead for interdomain routing. *Computer Communication Review*, 23(1):66–81, January 1993.

[54] S. Sibal and A. DeSimone. Controlling alternate routing in general-mesh packet flow networks. *Proceedings of ACM SIGCOMM '94*, pages 136–147, 1994.

[55] D. Sidhu, T. Fu, S. Abdallah, R. Nair, and R. Coltun. Open shortest path first (OSPF) routing protocol simulation. In *Proceedings of ACM SIGCOMM '93*, pages 53–62, 1993.

[56] M. Steenstrup. IDPR: An approach to policy routing in large diverse internetworks. *Journal of High Speed Networks*, 3(1), 1994.

[57] M. Thomas and E. W. Zegura. Generation and analysis of random graphs to model internetworks. Technical report, Georgia Tech, College of Computing, GIT-94-46.

[58] Dinesh C. Verma and P.M. Gopal. Routing reserved bandwidth multi-point connections. In *Proceedings of ACM SIGCOMM '93*, 1993.

[59] D. W. Wall. *Mechanisms for Broadcast and Selective Broadcast.* PhD thesis, Stanford University, California, USA, 1980.

[60] David Wall. Selective broadcast in packet-switched networks. In *Proceedings of the Sixth Berkeley Workshop on Distributed Data Management and Computer Networks*, pages 239–258, 1982.

[61] Bernard M. Waxman. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*, 6(9):1617–1622, 1988.

[62] Bernard M. Waxman. Performance evaluation of multipoint routing algorithms. In *Proceedings of IEEE INFOCOM '93*, pages 980–986, 1993.

[63] Liming Wei and Deborah Estrin. The trade-offs of multicast trees and algorithms. In *International Conference on Computer Communications and Networks*, August 1994.

[64] C. Williamson. Optimizing file transfer response time using the loss-load curve congestion control mechanism. In *Proceedings of ACM SIGCOMM '93*, pages 117–126, 1993.

[65] William T. Zaumen and J.J. Garcia-Luna Aceves. Dynamics of distributed shortest-path routing algorithms. In *Proceedings of ACM SIGCOMM '91*, 1991.

Appendix: Transit-Stub Edge Weight Calculation

The edge weights to be calculated and assigned are:

W_{tt}	Weight of a transit-transit interdomain edge
W_{ts}	Weight of a transit-stub interdomain edge
W_{ss}	Weight of a stub-stub interdomain edge

Note that all edges of the same type are given the same weight. Also, all *intra*domain edges are given unit edge weight. By making the weights of the *inter*domain edges sufficiently large, we guarantee that the path between any two nodes in the same domain will remain within that domain.

In calculating the weights of the interdomain edges, the following quantities, taken from the constructed graph, are used:

Parameter	Meaning
D_{top}	Diameter of transit-domain connectivity graph
D_t	Maximum diameter of any transit domain graph
D_s	Maximum diameter of any stub domain graph

The following constraints guarantee the desired locality characteristics:

Constraint	Result
$2W_{ss} > D_s$	Intra-stub preferred over any Inter-stub path
$2W_{tt} > D_t$	Intra-transit preferred over any Inter-transit path
$2W_{ts} > D_{top}W_{tt} + (D_{top}+1)D_t$	No shortcut via a stub node connected to any pair of transit nodes
$2W_{ss} > 2D_s + 2W_{ts} + D_{top}W_{tt} + D_{top}D_t$	Every all-transit path preferred over any path with three stubs
$W_{ss} \approx 2W_{ts} + \epsilon$	Direct connection between 2 stubs may be preferred sometimes.

By assigning the following values, the above constraints are satisfied:

$$
\begin{aligned}
W_{tt} &:= \lceil D_t/2 \rceil \\
W_{ts} &:= \lceil D_{top}W_{tt}/2 \rceil + \lceil (D_{top}+1)D_t \rceil \\
W_{ss} &:= D_s + 2W_{ts}
\end{aligned}
$$