Models

Five Flavors of Flat Random Graphs

The networking literature contains a variety of flat (i.e., non-hierarchical) random graphs used to model internetworks. All are variations on the standard random graph model that distributes vertices at random locations in a plane and then considers each pair of vertices; an edge is added between a pair of vertices with probability α. We call this the Pure Random model or simply the Random model. While it 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.

The other models also distribute the vertices randomly in the plane, however they alter the function used for the probability of an edge, in an attempt to better reflect real network structure. After the Pure Random Model, perhaps the most common random graph model is one proposed by Waxman [2], 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 $L = \sqrt{2} \times scale$ is the maximum distance between any two nodes. An increase in α will increase the number of edges in the graph, while an increase in β will increase the ratio of long edges relative to shorter edges. We call this the Waxman 1 model.

Several variations on the Waxman model have been proposed. They include:

- Replacing d by a random number between 0 and L [2]. We call this the Waxman 2 model.
- Scaling $P(u, v)$ by a factor $k\epsilon/n$, where ϵ is the desired average node degree, n is the number of nodes and k is a constant that depends on α and β [1].
- Allowing $\alpha > 1.0$ [3].

The second model, while not fundamentally different than the Waxman model, is interesting because the addition of the factor $radius = k\epsilon$ gives more direct control over the number of edges in the graphs that are generated, provided k is known. Clearly the α parameter of the Waxman model can be chosen to be equivalent to any particular setting of the parameters k, ϵ, n and α in the Doar-Leslie model.

We also propose two new models, intended to relate edge probability to distance between vertices (as in the Waxman model), but with more straightforward probability functions. Our Exponential model uses:

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

The probability of an edge in this model decreases exponentially with the distance between the two vertices. Our Locality model partitions the edges into discrete categories based length, and assigns a different edge probability for each category. With two categories we use one parameter $radius$ to define the boundary:

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

Model	Edge Probability
Pure Random	α
Waxman 1	$\alpha e^{-d/(\beta L)}$
Waxman 2	$\alpha e^{-rand(0,L)/(\beta L)}$
DoarLeslie	$\alpha (radius)/ne^{-d/(\beta L)}$
Exponential	$\alpha e^{-d/(L-d)}$
Locality	α if $d < L \times radius$ β if $d \geq L \times radius$

Table 1: Flat random graph models

One nice feature of the Locality model is that we have been able to extend many of the analytic results for (pure) random graphs to this model. Any finite number of categories are allowed, giving a reasonable amount of control over the topologies generated.

To summarize, Table 1 indicates the probability of an edge between two vertices at Euclidean distance d for each model described above.

Hierarchical Models

The random and regular models represent extremes in the sense that random models offer very little control over the structure of the resulting topologies, while regular models are extremely rigid in their structure. Neither type of model 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 topologies by connecting smaller components together according to a particular structure.

N-Level

The N-level hierarchical model constructs a topology recursively. Beginning with a connected graph, at each step in the recursion, each node in the current topology is replaced by a connected graph. The edges are "resolved" in various ways (e.g., by selecting nodes at random in the replacement graphs).

More precisely, in constructing the flat random graphs, we divide 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. Euclidean coordinates of all nodes are adjusted to the new scale, in which n squares of the original top-level plane are now $n \times S_2$ squares. This process continues recursively. 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

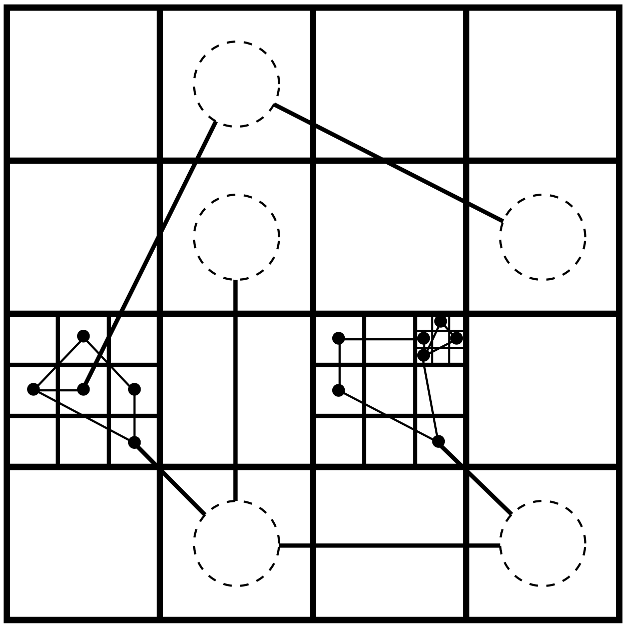

Figure 1: N-Level Hierarchical Layout

graph, most "top-level" edges (i.e. one that was part of the original graph) are typically longer than the second-level edges, while the second-level edges are typically longer than third-level edges. Figure 1 depicts the layout of a pure hierarchical graph.

In the hierarchical models, it is possible to define a routing policy based on the principle that routes between nodes within the same domain should stay entirely within that domain. We augment the edges with a routing policy weight, in addition to the Euclidean edge length, to use in constructing policy-based shortest paths. Because of the way "domains" are laid out in this method, there is a close relationship between the policy weights and the edge lengths.

Transit-Stub

Our Transit-Stub model produces hierarchical graphs in a different way, by composing interconnected transit and stub domains. (See Figure 2.) We first construct 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. Finally, we add some number of additional edges between pairs of nodes, one from a transit domain and one from a stub, or one from each of two different stub domains. Clearly, if the random graphs generated are all connected, this construction results in a connected graph. We note that Doar and Leslie proposed a graph model that has a similar hierarchical structure to our model, but is more limited in scope [1].

The size of the graph (number of nodes) and distribution of nodes between transit and

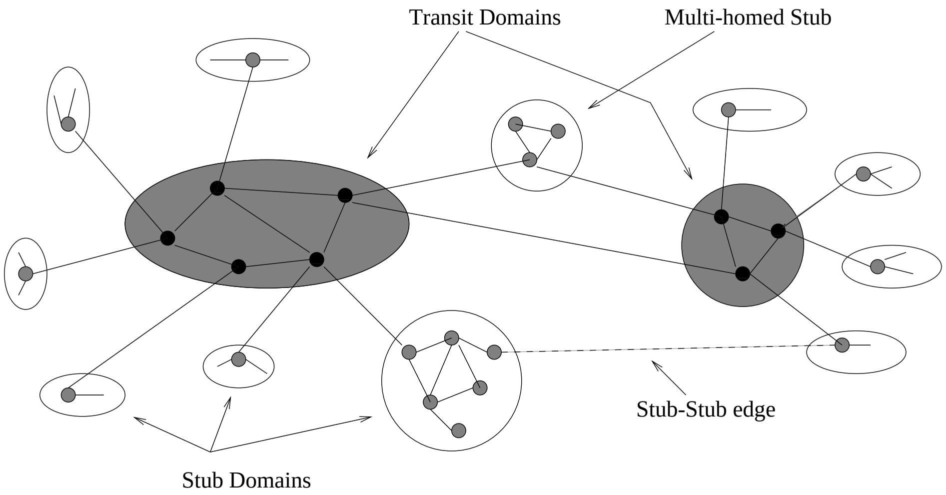

Figure 2: Transit-Stub Domain Structure

stub domains is controlled by the following parameters:

Parameter	**Meaning**	**Example**
T	# 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.

Note that the parameters given are averages. In our implementation, we "randomize" the numbers of nodes in the transit domains while preserving the average.

The values of K and N_s for each transit node and stub domain are similarly randomized. Our implementation also allows different random graph methods and parameters to be plugged in to construct transit and stub domain subgraphs with different characteristics.

Once the complete graph is constructed, integer edge weights are assigned in such a way that the following routing characteristics are exhibited when the edge weights are used by a shortest path algorithm: The routing characteristics of the current Internet can be summarized by the following general principles:

- The path connecting two nodes in the same domain stays entirely within that domain.
- The shortest path connecting node u in stub domain U to node v in another stub domain V goes from U through one or more transit domains to V, and does not pass through any other stub domains.
- In case two stub domains are connected directly via a stub-stub edge, the path between two nodes on the two domains *may* (but need not) go along that edge and avoid any transit domains.

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:

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

The domains in transit-stub graphs are placed in the plane in such a way that they may overlap one another. The same scale is used throughout the construction process; it is computed from the parameters given to the graph construction routine. The process is depicted schematically in Figure 3. A top-level graph determines the location and connectivity of the transit domains; that is, each node of the top-level graph corresponds to a transit domain. Each node of the top-level graph is then replaced by a complete transit domain graph, whose scale (as a fraction of the total length of a side of the unit square) is determined by the parameter *transfrac* of the program. Stub domains are created with a scale that is some fraction of the global scale; this fraction is again a program (compile-time) parameter *stubfrac*. Each stub domain is placed within a certain distance of the transit node to which it is attached; no stub node is more than the scale of the stub domain away from its transit node. Thus, in transit-stub graphs, policy and length-based edge weights of paths may have little to do with each other.

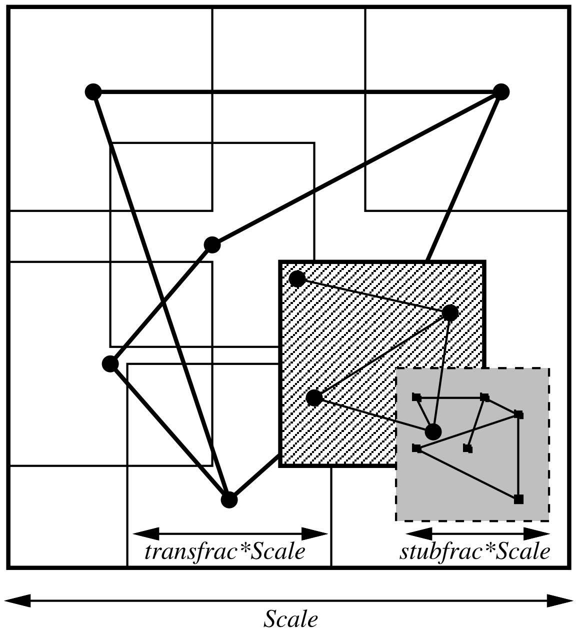

Figure 3: Transit Stub Layout

References

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

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

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