Modeling Internet Topology

Kenneth L. Calvert, Georgia Tech
Matthew B. Doar, Ascom Nexion
Ellen W. Zegura, Georgia Tech[*]

Abstract

The topology of a network, or a group of networks such as the Internet, has a strong bearing on many management and performance issues. Good models of the topological structure of a network are essential for developing and analyzing internetworking technology. This article discusses how graph-based models can be used to represent the topology of large networks, particularly aspects of locality and hierarchy present in the Internet. Two implementations that generate networks whose topology resembles that of typical internetworks are described, together with publicly available source code.

1 Introduction

The explosive growth of networking, and particularly 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. The reason for this 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.

The topological structure of a network is typically modeled using a graph, with nodes representing switches or routers, and edges representing direct connections (transmission links or networks) between switches or routers. Thus, the graph models *paths*—sequences of nodes—along which information flows between nodes in an internetwork. For example, a FDDI ring to which four IP routers are connected would be represented as a completely connected graph of four nodes. Hosts can also be represented as nodes; the typical host will be represented as a leaf connected to a single router node.

Additional information about the network can be added to the topological structure by associating information with the nodes and edges. For example, nodes might be assigned numbers representing buffer capacity. An edge might have values of various types, including costs, such as the propagation delay on the link, and constraints, such as the bandwidth capacity of the link.

The purpose of this article is to review the basic topological structure of the Internet, then present a modeling method designed to produce graphs that reflect the locality and hierarchy present in the Internet. Two implementations of the method are available; each produces graphs according to the basic method. The differences between the implementations may be of importance in choosing an implementation for use.

1.1 Structure of the Internet

Historically, large networks such as the Public Switched Telephone Network have grown according to a topological design developed by some central authority or administration. In contrast, there is no central administration that controls —or even keeps track of— the detailed topology of the Internet. Although its general shape may be influenced to some

[*]This work was supported in part by NSF Careers Award MIP-9502669.

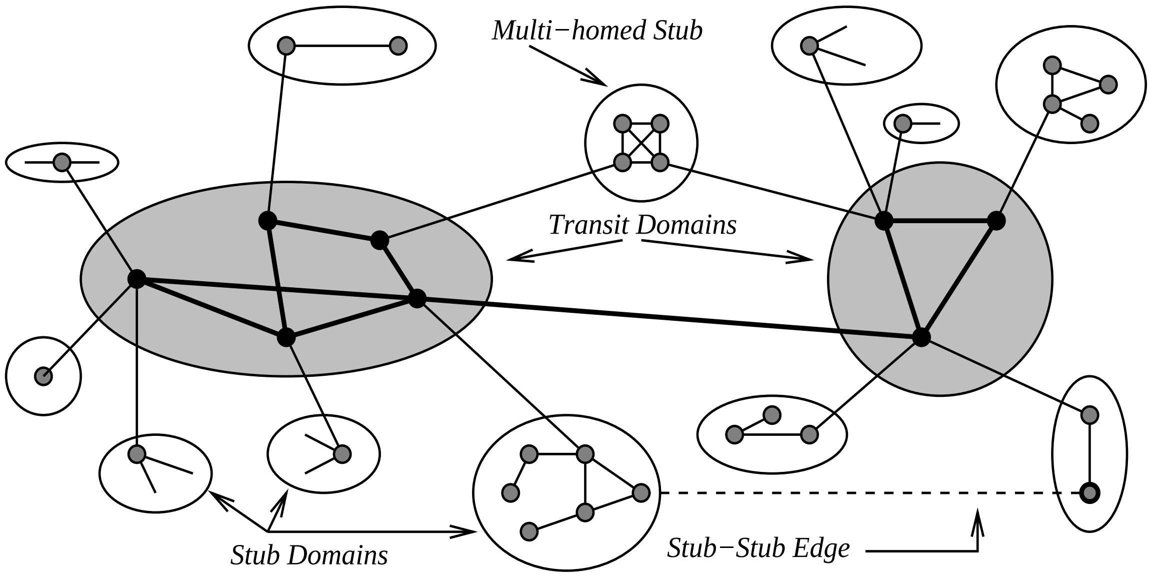

Figure 1: Example of Internet domain structure

small degree by policies for assignment of IP addresses and government funding of interdomain exchange points, the Internet, for the most part, just grows. The technology used to route and forward packets is explicitly designed to operate in such an environment. Today's Internet can be viewed as a collection of interconnected routing domains. Each routing domain is a group of nodes (routers, switches and hosts), under a single (technical) adminstration, that share routing information and policy.

Each routing domain in the Internet can be classified as either a *stub* domain or a *transit* domain. A stub domain carries only traffic that originates or terminates in the 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 1.) Stub domains generally correspond to campus networks or other collections of interconnected LANs, while transit domains are almost always wide- or metropolitan-area networks (WANs or MANs).

A transit domain consists of a set of *backbone* nodes. In a transit domain each backbone node may also connect 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 domains may have links to other stubs. Transit domains may themselves be organized in hierarchies, e.g. MANs connect mainly to stubs domains and WANs.

1.2 Existing Topology Models

One of the most commonly used models for generating random networks algorithmically is due to Waxman [3]. The nodes in the network are placed at random points in a two dimensional grid. Links (represented by edges between nodes) are added to the network by considering all possible pairs of nodes and then deciding whether a link should exist according to a probability function involving how far apart the two nodes are and how many links are expected to be in the whole network.

The original intention of this approach was to generate networks for comparing Minimum Steiner Tree algorithms. It has several serious drawbacks when used for generating typical internets.

First, the networks don't resemble the hand-drawn topological maps of real networks. There is no sense of a backbone or hierarchy, and the existence of links clear across the network is unrealistic.

Second, it does not guarantee a connected network. Each generated network must be checked for connectivity, then discarded or modified if the check fails. Either option involves extra work.

Third, as the number of nodes in the network grows, the number of links grows in a similar fashion. This is unlike real networks, where new links are added for redundancy, not just because more nodes joined the network.

Various modifications to Waxman's method have been proposed by the authors and others. Some of these attempt to restrict the longest links in the network, while others reduce the number of edges from any particular node. Still other modifications introduce a simple hierarchy to the network. None of them produce convincingly realistic networks.

2 A Better Method

Over the past few years a better method has been devised independently by the authors for generating graphs that reflect the hierarchical domain structure and locality that is present in the Internet [1, 4]. Three levels of hierarchy are modeled, corresponding to transit domains, stub domains, and LANs attached to stub nodes. The method constructs the graph piecewise, where the pieces correspond to domains at the different levels in the hierarchy. The connectivity within a domain (*intranetwork* connectivity) is dealt with separately from that between domains (*internetwork* connectivity).

2.1 Parameters

Two sets of parameters control the coarse properties of the networks generated. These parameters are chosen to provide reasonably simple control over the important structural characteristics of the graph. The parameters chosen also have obvious effects on the networks that are produced.

The first set of parameters governs the relative sizes of the three levels in the hierarchy:

- T, the total number of transit domains, and N_T, the average number of nodes per transit domain. Note that $T \geq 1$ and $N_T \geq 1$.
- S, the average number of stub domains per transit domain, and N_S, the average number of nodes per stub domain. Note that $S \geq 1$ and $N_S \geq 1$.
- L, the average number of LANs per stub node, and N_L, the average number of hosts per LAN. LANs are modeled as star topologies with a router node at the center of the star and the host nodes each connected to the center router. As compared to using a complete graph connecting all hosts in the LAN, this significantly reduces the number of edges in the graph and reflects the lack of physical redundancy in most LANs. Note that $L \geq 0$ and $N_L \geq 1$.

The total number of routing nodes, N_R, and the total number of hosts, N_H are given by:

$$\begin{aligned} N_R &= TN_T(1 + SN_S) \\ N_H &= TN_TSN_SLN_L \end{aligned}$$

Note that the parameter values are taken as the basis for distributions used to obtain the actual value for each run of the algorithm. Extra information can be associated with each parameter to describe the distribution of the parameter. For instance, an upper and lower bound on the number of nodes in a stub domain and the function for distributing the value between the bounds could be described.

The second set of parameters governs the connectivity within a domain (intranetwork connectivity) and the connectivity between domains at the same or higher and lower levels (internetwork connectivity).

- E_T, the average number of edges from a transit node to other transit nodes in the same domain. E_T must be large enough so that the graph corresponding to each transit domain can be connected. ($E_T \geq 2$.)

- E_S, the average number of edges from a stub node to other stub nodes in the same domain. E_S must be large enough so that the graph corresponding to each stub domain can be connected. ($E_S \geq 2$.)

- E_{TT}, the average number of edges from a transit domain to another transit domain. E_{TT} must be large enough so that the transit domains can be connected to one another. ($E_{TT} \geq 2$.)

- E_{ST}, the average number of edges from a stub domain to a transit domain. Every stub domain must be connected to at least one transit domain, thus $E_{ST} \geq 1$; multi-homed stub domains will have more than one edge to a transit domain.

- E_{LS}, the average number of edges from a LAN to a stub node. Every LAN must be connected to at least one stub node, thus $E_{LS} \geq 1$. LANs may be connected to more than one stub node in the assigned stub domain.

2.2 Generation Process

The generation process begins at the top (transit domain/WAN) level of the hierarchy, and proceeds down to the lowest (LAN) level of the hierarchy. The nodes within a level are generated in a rectangular sub-region of the overall space occupied by the graph. The scale of the sub-region in which the nodes are distributed is changed for each type of network, with LANs having the smallest scale.

In many of the steps, edges must be generated so that a sub-graph is connected and a specified edge count is met. There are many ways to achieve this; the implementations described later give two possible methods.

The generation process is roughly as follows, though the order of the steps may differ in a particular implementation:

Step 1. A location in the plane is chosen for each of the T transit domains from the overall space occupied by the graph. Edges are generated between transit domains such that the domains are connected and the E_{TT} specification is satisfied.

Step 2. For each transit domain, nodes are placed in a sub-region centered around the transit domain location, so that transit domains contain, on average, N_T nodes. Edges are generated within each transit domain such that the domain is connected and the E_T specification is satisfied.

Step 3. Particular transit nodes from Step 2 are selected as the endpoints of the inter-transit domain edges from Step 1.

Step 4. The locations for the TN_TS stub domains are chosen. For each stub domain, nodes are placed in a sub-region centered around the chosen location so that stub domains contain, on average, N_S nodes per domain. Edges are generated within each stub domain such that the domain is connected and the E_S specification is satisfied.

Step 5. Each stub domain is connected to a transit node by an edge, connected to a particular stub node and a particular transit node. If $E_{ST} > 1$, additional edges are added from stub domains to transit nodes.

Step 6. The locations for the TN_TSN_SL LANs are chosen. For each LAN, N_L nodes (representing host systems) are placed in a star around a center router.

Step 7. Each LAN is connected to one stub node by an edge from the center router to the stub node. If $E_{LN} > 1$, then additional edges are added from center routers to stub nodes.

In the next two sections, we describe two implementations of the basic generation method. Both implementations are available in the public domain; these sections describe differences that may be important in choosing an implementation for further development or use. For each implementation, we describe additions to and deviations from the basic model in three categories: graph generation, overlays of information and implementation details.

3 Transit-Stub

The first implementation is the Transit-Stub (TS) model [4] and is part of Georgia Tech Internetwork Topology Models (GT-ITM), a package for generating and analyzing graph models of internetworks.

The Transit-Stub model does not currently support representation of host systems. Thus, all nodes are of the same type, namely routers, and $N_L = E_{LN} = 0$.

The Transit-Stub model produces connected subgraphs by repeatedly generating a graph according to the edge count, and checking the graph for connectivity. Unconnected graphs are discarded. This method ensures that the resulting sub-graph is taken at random from all possible (connected) graphs; however, it may take a long time to generate a connected graph if the edge count is relatively small.

Extra edges from stub domains to transit nodes are added by random selection of the domains and nodes. The TS model also supports an additional parameter, E_{SS}, indicating the number of stub-to-stub edges. These edges are also added by random selection of the domains and nodes involved.

Several types of information are associated with nodes and edges to augment the basic topology. Each node has a string label that indicates properties of the node: an identifier indicating whether it is a transit or stub node, a global identifier for the domain to which it belongs, and a domain-local identifier. For stub nodes, the label also indicates the identifier for the primary transit node where the stub domain is attached. Each edge has a routing policy weight that can be used to find routes that follow the standard domain-based routing outlined earlier. That is, a shortest path found using the routing policy weights will traverse transit domain(s) if and only if the two endpoints are in different domains.

The Transit-Stub generation software is written in C, and uses the Stanford GraphBase [2] for representation and manipulation of graphs. The GT-ITM release includes the necessary libraries from the Stanford GraphBase (SGB), thus the user need not download or understand this code to generate and analyze graphs.

4 Tiers

The second implementation is called 'Tiers' [1]. In this model, the three levels of hierarchy, or tiers, are referred to as WAN, MAN and LAN levels, corresponding to the transit domains, stub domains and LANs of the basic method. The model does not currently support multiple WANs, thus $T = 1$. The parameter values for Tiers also differ slightly from that of the basic method in that "number of node" parameters are per higher-level domain, not per node in the higher-level domain.

The Tiers model produces connected sub-graphs by joining all the nodes in a single domain using a minimum spanning tree. The use of a minimum spanning tree is a crucial feature of Tiers and is particularly appropriate since it is sometimes used in reality as the basis for laying out large networks. When adding edges for intranetwork redundancy, edges are added to the closest nodes in the network, in increasing order of Euclidean distance. For internetwork redundancy, extra edges to the closest nodes in the next higher domain are added. This ensures some degree of local connectivity within geographic constraints.

Nodes are permitted to occupy the same points in the grid, since the purpose of the grid is to generate costs using an approximation of nodes' physical locations, and this can represent multiply connected hosts.

When a node is a gateway node, the node is represented as two interconnected nodes, e.g. a LAN node and a MAN node. The metrics of the edge between the two types of nodes reflect the processing delay and bandwidth constraints within the host. This approach permits better estimates of network performance than using strict approach that every edge is a physical link in a network.

If the redundancy parameters are small integers, the time complexity of Tiers is $O(N_H^2)$. For comparison, the complexity of Waxman's original algorithm is also $O(N_H^2)$, but with potentially longer generation times for some values of the probability function. Tiers is written in C++ and uses few

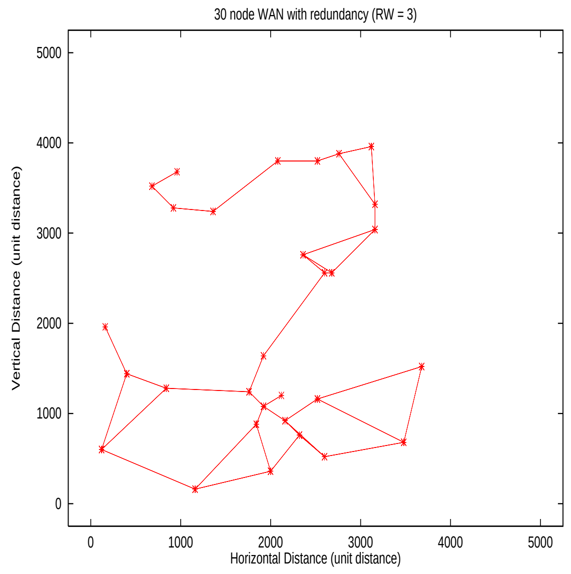

Figure 2: A Typical Tiers Network

floating point operations, in order to decrease simulation time when constructing thousands of network models with large numbers of nodes.

Figure 2 shows a single level of network as generated by Tiers. The basic spanning tree can be seen, together with the edges added for redundancy. The units of distance are the smallest unit in the network.

Figure 3 shows a typical full internetwork generated by Tiers. There is one WAN with eight nodes, three MANs with three nodes each and two LANs per MAN with three nodes per LAN.

Figure 4 shows a larger internetwork than Figure 3, also generated by Tiers. The end points of the links to LAN nodes have been omitted for clarity, so only the WAN nodes are seen clearly. Networks such as the one shown in Figure **??**, but with more than 30,000 nodes, have been generated in a few minutes using a typically configured Sparc 20.

5 Comparison of Implementations

Both implementations are based on the explicitly hierarchical modeling approach described in this article. Tiers introduces a different method for connecting the nodes in a network, by using a mini-

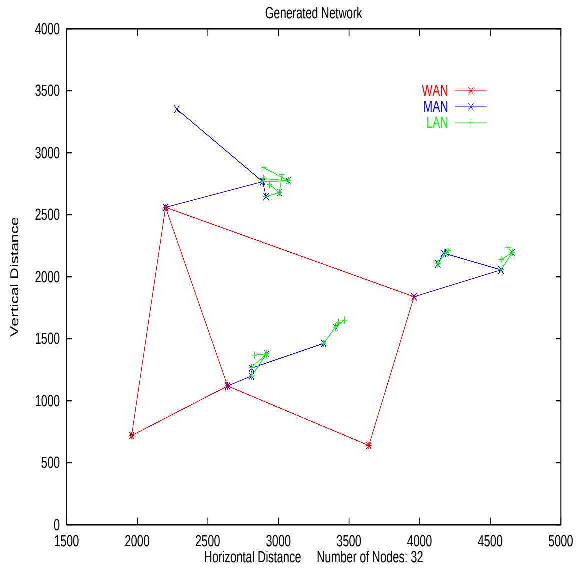

Figure 3: A Typical Tiers Internetwork

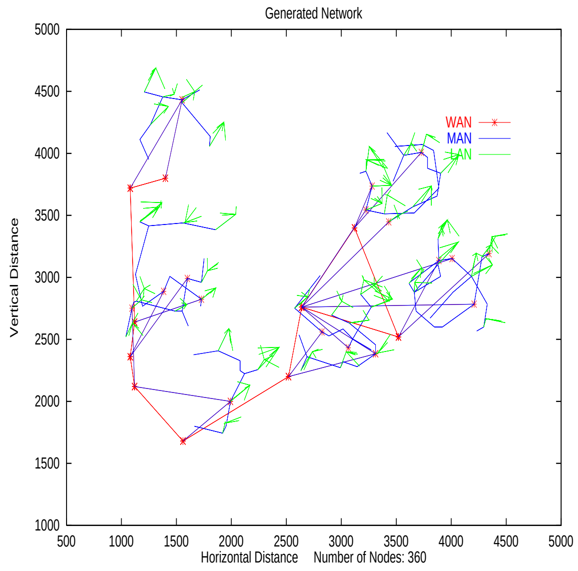

Figure 4: A Larger Tiers Network

mum spanning tree. The use of a minimum spanning tree guarantees connectivity, reduces the generation time of the network and produces more realistic networks at the WAN scale.

The Transit-Stub implementation generates graphs using a more probabilistic approach than that of Tiers, which in contrast uses a larger (though still small) set of parameters to control the different aspects of the networks produced. In both implementations, most of the parameters can be expected to remain constant between runs of generated networks.

A useful simplification introduced by Tiers to reduce number of edges, and hence improve simulation times, is to model LAN topologies as stars. Another useful idea is to treat a node which connects two types of networks as two nodes, with one node in each type of network. This permits modeling of the delay in transferring data from one network to another, which is often neglected in other network models.

Transit-Stub simplifies the network models by representing only routers and switches. In addition, weights are associated with edges to produce paths that follow standard Internet routing policy.

6 Conclusions

The issue of generating example networks is important for the testing of routing algorithms and generating likely network deployment scenarios. This article has described the difficulties of modeling the topology of typical communications networks and provided brief details for a modeling approach and two implementations.

In the future, once more data about the characteristics of the networks to be modeled has been gathered, better distributions for the values of the model parameters can be defined. Examining how the values of the parameters change with time may even help predict how networks change within organizations.

Acknowledgements

Many thanks are due to Bill Willcox and others at Ascom Nexion for helpful readings of the original Tiers work. Several students at Georgia Tech, including Megan Thomas, Samrat Bhattacharjee and Elizabeth Edwards, have been involved in portions of the Transit-Stub development.

References

[1] M. B. Doar. A Better Model for Generating Test Networks. In *Proceedings of Globecom '96. Also at* `ftp://ftp.nexen.com/pub/papers`, Global Internet '96, November 1996.

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

[3] B. M. Waxman. Routing of Multipoint Connections. *IEEE Journal on Selected Areas in Communications*, 6(9):1617–1622, 1988.

[4] E. Zegura, K. Calvert, and S. Bhattacharjee. How to Model an Internetwork. In *Proceedings of IEEE INFOCOM*, April 1996.

Source Code

Source code for the Transit-Stub implementation is available at:

`http://www.cc.gatech.edu/fac/Ellen.Zegura/graphs.html`

Source code for the Tiers implementation is available at `ftp://ftp.nexen.com/pub/papers`.

Biographies

KENNETH L. CALVERT (`calvert@cc.gatech.edu`) received the S.B. degree in Computer Science and Engineering from the Massachusetts Institute of Technology in 1979, the M.S. in Computer Science from Stanford University in 1980, and the Ph.D. in Computer Sciences from The University of Texas at Austin in 1991. From 1979 to 1984 he was a Member of Technical Staff at Bell Laboratories in Holmdel, New Jersey. He has been a faculty member of the College of Computing at Georgia Tech since 1991. His research deals with communication protocols and network architecture. A general theme of much of his work is the importance of interoperability, and of designing and extending protocols and implementations so they can be useful beyond their original contexts.

MATTHEW B. DOAR (`doar@pobox.com`) is Senior Member of Technical Staff (Internetworking) with Ascom Nexion. He has worked in the area of multicast and ATM switch design and implemented the ATM Forum LANE protocol. He is currently working on IP Autoconfiguration for the next generation of products. He obtained his B.A. and Ph.D. from Cambridge University in Computer Science, and his interests include high-speed network protocols, multicast routing algorithms and object-oriented analysis and design tools for real-time communications systems.

ELLEN W. ZEGURA (`ewz@cc.gatech.edu`) received the B.S. degree in Computer Science (1987), the B.S. degree in Electrical Engineering (1987), the M.S. degree in Computer Science (1990) and the D.Sc. in Computer Science (1993) all from Washington University, St. Louis, Missouri. She has been a faculty member of the College of Computing at Georgia Tech since 1993. Her current interest is in wide-area networking support for complex applications. In particular, the work focuses on dealing with fluctuations in wide-area performance that occur on a time scale that is slow enough to be tractable, yet fast enough to require dynamic, online solutions.