Bessie[1]: Portable Generation of Network Topologies for Simulation

Frank Adelstein[1]
Frederick Hosch[2]
Golden G. Richard III[2]
Loren Schwiebert[3]

[1]Xerox Design Research Institute
Cornell University
Ithaca, NY 14853
frank@dri.cornell.edu

[2]Dept. of Computer Science
University of New Orleans
New Orleans, LA 70148
{*fred, golden*}*@cs.uno.edu*

[3]Dept. of Electrical and Computer Eng.
Wayne State University
Detroit, MI 48202
loren@ece.eng.wayne.edu

Abstract

Widespread use of computer networking has resulted in considerable attention being paid to a variety of network-related problems, the generation of efficient multicast trees being one. While many algorithms for generation of multicast trees have been proposed, their relative effectiveness is difficult to assess. Some algorithms have never been implemented. Many have been simulated, but often using ad-hoc networking modeling and simulation tools, without consistent parameters, making direct comparisons difficult.

In this paper, we discuss a network topology generation tool named Bessie, written entirely in Java. Bessie generates descriptions of random point-to-point and hierarchical networks, based on user-specified statistical parameters. We introduce a modification to Waxman's [9] *parameters, commonly used in grid-based network topology generators, which eliminates undesirable increases in node degree as the number of nodes in a network increases. The modification improves on proposed fixed scale factors.*

Keywords: multicast network protocols, dynamic multicast groups, network generation tools, simulation.

1 Introduction

The widespread use of computer networking has resulted in a number of formidable networking-related problems. Consider just one of these problems: efficiently supporting popular multicast-based applications (e.g., video-on-demand and videoconferencing). Multicast transmission may use network bandwidth more efficiently than multiple point-to-point messages for such applications, but only if efficient multicast trees can be constructed and maintained. Generation of static optimal multicast trees can be modeled as the Minimum Steiner Tree problem: given a set of nodes in an undirected graph G, and a subset of nodes S, find the minimum cost tree which connects the set S. Nodes used in the minimum cost tree that are not in S are called *Steiner nodes*. This problem has been shown to be NP-Complete.

Since generation of optimal multicast trees is prohibitively expensive, there has been extensive research into development of heuristic methods for generating multicast trees [e.g., 1, 2, 4, 5, 7, 8, 9, 10]. Algorithms for generating and updating the multicast tree must typically balance "goodness" of the generated tree, execution time, and storage requirements. Recently, more attention has been focused on distributed solutions [1, 2] and support for dynamic multicast groups [1, 5, 8], where the set of nodes participating in a multicast session may change. Depending on the particular technique, changes are allowed during the initial construction of the multicast tree, during the lifetime of the multicast session, or both. While many algorithms have been proposed, their relative effectiveness is difficult to assess, since many have never been directly implemented. Of course, this is usually justifiable on practical grounds—large networks are rarely available as testbeds and experiments conducted on small networks may not yield relevant answers. Instead of implementation, potential techniques can be extensively simulated, using network descriptions which may (or may not) be similar to "real" networks. Unfortunately, many of the techniques to date have been simulated with ad-hoc tools and network models, making direct comparisons difficult.

One typical way of representing a network for simulation is as a graph [9]. Nodes in the graph correspond to routers (or switches) and edges correspond to direct connections. Costs are

[1] Bessie is named after the blues singer Bessie Smith.

associated with edges to model, for example, delay, or available bandwidth. These graphs can be created manually, which is extremely tedious when large numbers of network descriptions are created, or generated algorithmically with an appropriate tool. If a tool is used, it should be easy to control relevant characteristics of the generated networks, to visualize the resulting networks, and to generate networks quickly, since often large numbers of network descriptions must be created for simulation purposes.

In this paper, we discuss a network topology generation tool named Bessie. Bessie is written entirely in Java and generates random point-to-point and hierarchical networks, based on user-specified statistical parameters. Bessie operates in both GUI and command-line modes and interfaces with external software for generating multicast trees. The GUI allows users to easily see the effects that different parameters have on the structure of networks and resulting multicast trees. Command-line mode is appropriate for the generation of a large number of network descriptions. Network descriptions are saved in a format easily incorporated into other software, such as network simulators. Bessie forms part of our multicast simulation testbed, which we are developing to directly compare multicast tree generation techniques. While we are using the multicast tree problem as an example, we expect the situation to be similar in other areas of networking research and Bessie should be applicable (perhaps with some enhancement) in these areas as well.

The rest of the paper is organized as follows. Section 2 briefly describes related work. Bessie is discussed in Section 3. Finally, Section 4 presents conclusions and future research directions.

2 Related work

One of the techniques most widely used for generating random networks uses a grid of points with integer coordinates and a simple probability function for determining connectivity. This technique was discussed by Waxman [9]. Doar and Leslie [5] enhanced the Waxman model to use a scaling factor to offset growth in average node degree as the number of nodes in a network increases. Doar and Leslie's technique for non-hierarchical networks is discussed in Section 3.1. They also propose a technique for building hierarchical networks, which essentially builds smaller networks using the technique for non-hierarchical networks, and then connects these smaller "satellite" networks. Bessie's edge generation technique is related to that of Doar and Leslie, but achieves the same functionality with fewer parameters and does not require experimentation to fix desired mean node degree.

Recent work has focused on modeling internetworks, to arrive at network models that more closely model large-scale "real" networks. Calvert, Doar, and Zegura [3] build an internet-like topology piecewise from *transit domains*, *stub domains,* and LAN's connected to the stub nodes. The user can specify average number of transit domains, average number of stub domains per transit domain, average number of LAN's per stub node, and average number of hosts per LAN, in addition to parameters for connectivity between the various levels and between nodes in a particular level. Doar's [6] method introduces redundancy and models WAN's, MAN's, and LAN's. LAN's are modeled as star topologies. A grid is still used for network generation, but the scale is changed depending on the component being built (WAN, MAN, or LAN). Zegura, Calvert, and Bhattacharjee [11] compare various methods for random graph generation and propose a hybrid method called "Transit Stub", similar to Doar's [6] method, which allows creation of large internet-like networks from the composition of smaller random graphs.

3 Bessie: A portable tool for creating network topologies

Bessie allows both interactive and command-line generation of network topologies, with some nodes being designated as multicast nodes. Bessie generates networks by placing points randomly on a two-dimensional grid and then using probability functions to determine which nodes are connected by edges. Connectivity can be constrained by node degree and node distance. After each network is generated, a minimum spanning tree of the network is constructed, and, if necessary, the network is "repaired" by adding least cost edges to construct a single connected component. Currently Bessie generates edge weights based on distance or a uniform cost of 1. Other edge weight models are possible.

When used interactively, Bessie allows the user to view the current network and, optionally, superimposed minimum spanning and multicast trees. Bessie provides an interface to allow generation of multicast trees by external, user-specified programs, since there are a number of applicable techniques. Multicast nodes, non-multicast nodes, normal edges and edges which are part of the minimum spanning and multicast trees are color-coordinated for easy identification.

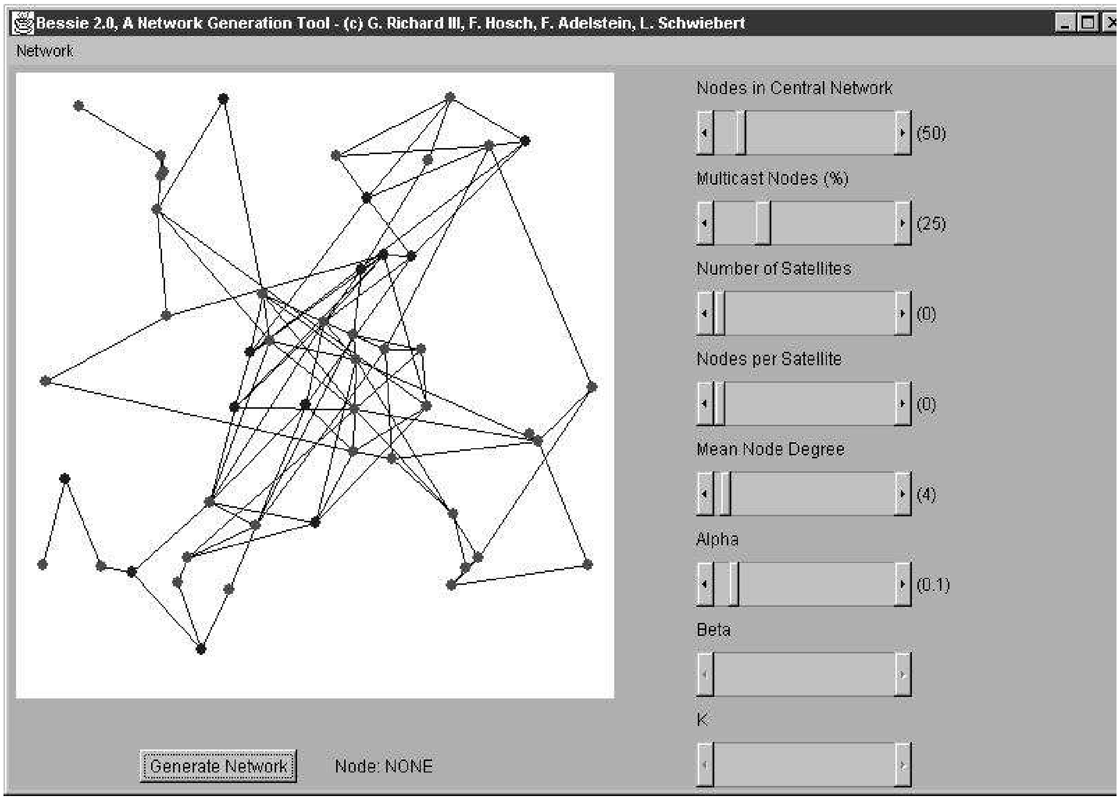

Figure 1. User interface for the current version of Bessie.

The user may also read and write descriptions of networks as Bessie files, and save a graphic image of the current network. Figure 1 illustrates the basic Bessie GUI. The parameters are explained in Section 3.1. While Bessie is being used primarily in evaluation of multicast tree generation algorithms, we expect the basic model to be useful in other areas of network research, particularly as additional network models are incorporated.

3.1 Parameters

A set of parameters controls network generation in Bessie. Among the parameters are the following:

- ✓ total size of the network,
- ✓ the number of satellite networks and the average size of a satellite network (for hierarchical networks only),
- ✓ the mean node degree,
- ✓ the percentage of nodes which will be multicast nodes, and
- ✓ a single parameter α which controls the probability of edges between distant nodes.

All of the parameters may be specified either interactively or on the command line. When Bessie is used interactively, the user can immediately see the effects that changes in parameters have on network topology and the corresponding multicast tree.

The probability function used to determine whether a pair of nodes (u, v) should be connected is computed as follows. Let the "raw probability" that nodes u and v should be connected be given by

$$P_r(u, v) = e^{-d(u, v)/\alpha L} \qquad \text{(Equation 1)}$$

where L is the maximum possible distance between two nodes, $d(u, v)$ is the distance between u and v, and α is a parameter in the range $0 < \alpha \leq 1$. This follows Waxman [9] and Doar and Leslie [5]. The larger the value of α, the higher the probability that distant nodes will be connected.

To simplify the subsequent computation, $d(u, v)$ is taken to be $\max(\,|x_u\text{-}x_v|,\ |y_u\text{-}y_v|\,)$, where (x_u, y_u) and (x_v, y_v) are the Cartesian coordinates of nodes u and v respectively. Thus L is effectively the diameter of the grid from which nodes are selected.

Now let P be the mean raw probability for all pairs of grid points. If $num(x)$ is the number of pairs of grid points x units apart and N the total number of grid point pairs, then P can be approximated by

$$P = \frac{\int_0^L num(x) e^{-x/\alpha L} dx}{N} \qquad \text{(Equation 2)}$$

By counting, $num(x)$ and N can be determined to be:

$$num(x) = 2x^3 - (6L + 6)x^2 + (4L^2 + 8L + 4)x \qquad \text{(Equation 3)}$$

$$N = (L^4 + 4L^3 + 5L^2 + 2L) / 2 \qquad \text{(Equation 4)}$$

Finally, given P, the probability function used to determine if nodes (u, v) are connected is

$$P_e(u, v) = (\bar{e} / nP)\, P_r(u, v) \qquad \text{(Equation 5)}$$

where n is the number of nodes in the graph, and $\bar{e}$ is the desired mean degree. The advantage of this approach over Doar and Leslie [5] is simplicity. Their edge probability function is

$$P_e(u, v) = ((k\bar{e} / n)\beta)\, P_r(u, v) \qquad \text{(Equation 6)}$$

where k and β are additional parameters. Thus one must specify four values *a priori*: α, β, k, and $\bar{e}$ (the parameter β is from Waxman's formulation). When building graphs, for a given α and $\bar{e}$, one must experimentally determine an appropriate $k\beta$ so that the graphs generated will in fact exhibit the desired mean node degree. With our approach, α and $\bar{e}$ are the only two values that need to be specified, and can be specified independently. The graphs produced exhibit the desired mean node degree without the need for providing additional, experimentally determined, parameters.

Bessie implements both the Doar and Leslie [5] edge model and the improved model described above. To illustrate the difference, consider the set of parameter values discussed in Section 2.1 of Doar and Leslie, namely, 150 node networks, with $\alpha = 0.25$, $\beta = 0.2$, a desired mean node degree of 3, and k experimentally determined to be 25. These settings consistently give actual mean node degrees of 3-4 under our implementation of Doar and Leslie's technique. Increasing α to 0.35 results in actual mean node degrees of 5-6, and increasing α to 0.65 results in actual mean node degrees of 7-9. To test our edge model, we varied α by 0.1 in the range 0.1 to 0.9, randomly generating 100 networks for each value of α. The actual node degree for every network generated was in the range 3.0-3.1. Similarly, holding α constant while varying desired mean node degree requires experimentally determining a new value for $k\beta$ under Doar and Leslie's model. In contrast, our model consistently achieves desired node degrees within $\pm$ 1 while varying either desired mean node degree or α, for all network sizes. These results are not reported in tabular form for obvious reasons—desired node degree under our edge model is virtually identical to actual node degree in all cases.

3.2 Bessie file format

Bessie reads and writes portable files which contain network descriptions. Since a primary design goal of Bessie is portability, Bessie files are stored in ASCII format. We did not take advantage of Java's ability to write portable binary files, which would substantially reduce storage space, because we want Bessie files to be easily readable by network simulation software written in other languages. In the current version, a Bessie file contains version information, network topology, edge weights, a complete set of parameters used to generate the network (so networks with similar characteristics can easily be generated), a snapshot of the minimum spanning tree for the network, and optionally, a multicast tree specification created by an external application.

Information after the optional multicast tree specification is ignored. This is convenient because it allows network simulators and other software to write interesting statistics directly to the same file that contains the network topology, without preventing the use of Bessie to visualize the network.

4 Conclusions and future work

Widespread use of computer networking has resulted in a wealth of networking-related problems and generally, potential solutions are simulated rather than directly implemented. We have described Bessie, a tool for generating random network topologies for simulation purposes. Bessie allows the user either to work with network models visually and immediately see the effects of modifying network parameters, or to generate many network descriptions in command-line mode. Bessie's improved edge generation model is intuitive and eliminates the need for experimentation with additional parameters to achieve desired mean node degrees. Bessie provides an interface to external multicast tree generation code, so networks can be generated, multicast trees automatically generated using a particular technique, and the multicast tree visualized, all without leaving the application. Finally, Bessie is written in Java and generates completely portable network descriptions so it is usable on a variety of architectures.

We are currently developing Bessie primarily to generate network models as part of our testbed for head-to-head evaluation of multicast tree generation algorithms. Future work will include incorporation of new network models [e.g., 3, 6, 11] into Bessie to broaden its applicability in other areas of networking research. The current version of Bessie will be available via a link on *http://www.cs.uno.edu/~golden/research.html*.

References

[1] F. Adelstein, G. G. Richard III, L. Schwiebert, "Distributed Multicast Tree Generation with Dynamic Group Membership", University of New Orleans, Computer Science Technical Report UNOCS-TR97-01.

[2] F. Bauer, A. Varma, "Distributed Algorithms for Multicast Path Setup in Data Networks," Proceedings of GLOBECOM '95, pp. 1374-1378, Nov. 1995. Also as University of California, Santa Cruz, Computer Engineering Department Technical Report UCSC-CRL-95-10, August 1995.

[3] K. Calvert, M. Doar, E. Zegura, "Modeling Internet Topology," IEEE Communications Magazine, June 1997.

[4] C. Diot, W. Dabbous, J. Crowcroft, "Multipoint Communication: A Survey of Protocols, Functions, and Mechanisms," IEEE Journal on Selected Areas in Communications, 15(3), pp. 277-290, April 1997.

[5] M. Doar, I. Leslie, "How Bad is Naïve Multicast Routing?" IEEE INFOCOM, pp. 82-89, San Francisco, California, March 1993.

[6] M. Doar, "A Better Model for Generating Test Networks," IEEE Global Telecommunications Conference/GLOBECOM'96, London, November 1996.

[7] Y. Im, Y. Lee, S. Wi, Y. Choi, "Delay Constrained Distributed Multicast Routing Algorithm," Computer Communications, 20(1), pp. 60-66, January 1997.

[8] B. H. Ryu, M. Murata, H. Miyahara, "A Dynamic Application-Oriented Multicast Routing for Virtual-Path Based ATM Networks," IEICE Transactions on Communications, E80-B(11), pp. 1654-1663, November 1997.

[9] B. Waxman, "Routing of Multipoint Connections," IEEE Journal on Selected Areas in Communications, 6(9) pp. 1617-1622, December 1988.

[10] P. Winter, "Steiner Problem in Networks: A Survey," Networks, 17(2) , pp. 129-167, 1987.

[11] E. Zegura, K. Calvert, S. Bhattacharjee, "How to Model an Internetwork," Proceedings of IEEE Infocom '96, San Francisco, CA.