Extending and Enhancing GT-ITM

K. Calvert*, J. Eagan†, S. Merugu†, A. Namjoshi*, J. Stasko†, E. Zegura†
*University of Kentucky †Georgia Institute of Technology
{calvert, aditya}@netlab.uky.edu {eaganj,merugu,stasko,ewz}@cc.gatech.edu

ABSTRACT

GT-ITM is a collection of software tools for creation, manipulation, and analysis of graph models of internet topology. It has been used by networking researchers in a variety of ways, most often to create topologies for use in simulation studies. This paper describes the features of a new release of GT-ITM, including enhanced visualization capabilities; a routing and forwarding module for use with large graphs; and support for modeling of interdomain routing policies.

1. INTRODUCTION

GT-ITM is a widely-used facility for creation and analysis of graph models of network topology. Such models are widely used in simulation based evaluation and comparison of new protocols and algorithms, and also as a tool to understand the topological characteristics of the Internet. We have developed a new release of GT-ITM, which includes a number of new features and enhancements, including: a visualization capability; support for routing/forwarding in very large graphs (up to hundreds of thousands of nodes); and the ability to specify and use interdomain routing policies on a per-domain basis. The next section presents a brief overview of GT-ITM, followed by a section describing each of the major enhancements. Section 6 describes some other features being considered for inclusion, on which we would like to get feedback at the workshop.

2. OVERVIEW OF GT-ITM

The original GT-ITM [2] comprises:

- A command-line program that controls the creation of random graphs according to various models (including the *transit-stub* model [9]) and parameters.
- A command-line program that controls the evaluation of various characteristics of graphs, e.g. diameter.
- Various example graphs and parameter files for creating them.

ACM SIGCOMM 2003 Workshops, August 25&27, 2003, Karlsruhe, Germany.

All tools are built on the Stanford GraphBase (SGB) [5] platform for general graph representation and manipulation. Nodes in a GT-ITM SGB graph represent routers; bidirectional links are represented by edges (pairs of arcs). SGB was designed to be completely portable and very efficient; this gives the GT-ITM tools an advantage in that area as well.

GT-ITM has been used by many researchers in comparisons of routing, caching, peer-to-peer and other algorithms and services. It has been included as part of the "ns2" network simulator package for several years.

However, the state of the art in topology modeling has advanced since the initial release of GT-ITM. The topology of the Internet has been studied a good deal over the last five years, and more is known about its structure. For example, BGP data has provided a more accurate picture of the way domains are interconnected [1]. Other researchers have developed topology-generation tools that attempt to mimic this domain-level structure [7, 6, 8]. While GT-ITM can produce graphs with a discernible domain structure (and edge weights ensuring that shortest paths will respect that structure), the domain-interconnection structure of those graphs is not designed to reflect the "power-law-like" distributions seen in the high-level structure of the Internet.

In addition, the limits of what is feasible in topology-related simulation studies have increased dramatically; it is now possible to deal with much larger graphs for high-level simulations. However, the some of the tools in GT-ITM and the Stanford GraphBase are inadequate for graphs of more than a few thousand nodes. As an example, consider the problem of comparing the paths followed by packets, say, for different multicast routing algorithms. For moderate-sized graphs, it suffices to run an all-nodes shortest paths algorithm and build an $O(n^2)$-sized next-hop table so the next hop from any node to any other node can be determined in constant time. However, for a graph with 10^5 nodes, the size of the table alone is prohibitive (never mind the time required to compute it) for current platforms. For graphs of this size, different route-computation methods are needed.

Another change related to routing is the recognition of *policy* as an important factor in understanding interdomain route selection [4, 3]. Packets in the Internet do not always follow "shortest" path between two domains in terms of distance or number of hops; rather, their paths are determined by business or other relationships. For simulations involving interdomain traffic, we would like to be able to compute and use routes that reflect this fact. Given this capability, we would like to use GT-ITM to study the effects of such policies on routing protocols. Finally, the ability to visualize topology models can be very useful, both as a sanity-check and as a way to detect features that might be relevant to the topic under study.

With these and other motivations, we have extended and en-

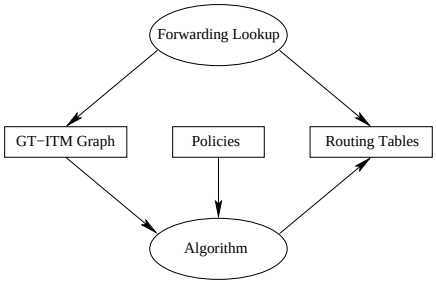


Figure 1: Routing/Forwarding tool structure

hanced GT-ITM. The next few sections describe the enhancements motivated by the above observations.

3. ROUTING AND FORWARDING CAPABILITIES

GT-ITM's graph-construction tool can produce graphs with random interconnections exhibiting various types of high-level structure. One type, the *transit-stub* graph, imposes a domain structure, distinguishes between transit and stub domains, and assigns edge weights in such a way that traditional shortest-path routing produces paths that respect the domain structure of the graph: Once a path leaves a domain, it does not return to that domain. Also, paths do not traverse stub domains except those containing the source or destination. However, GT-ITM does not provide any special tools for construction and use of routing/forwarding tables for constructed graphs.[1] The absence of such support can be a significant drawback for graph models with tens or hundreds of thousands of nodes. Because path computing can be rather expensive for large graphs, it is important to support offline computation and storage of the results for later re-use.

In particular, we want to:

- Support construction of forwarding tables for very large transit-stub graphs.
- Be able to store computed forwarding information in a format that is independent of the particular routing algorithm and separate from the graph itself, so that the same graph can be used with different forwarding tables, and the effects of different routing algorithms and policies can be compared.
- Provide a way to use these precomputed forwarding tables efficiently to determine the next hop to a given destination.

Figure 1 shows the general structure of this approach, which has two software components. We provide a run-time API which, given a graph, and forwarding tables, encapsulate the "forwarding lookup" function, which returns the next hop from a given node to a given destination, according to the given tables. The other component runs off-line, constructing forwarding tables according to selected routing algorithms and storing them in a standard format. To implement these components in a manner suitable for use with large graphs, we resort to the same method used in the Internet: we split the problem into two parts, namely computation of *intradomain* routing information for each domain, and computation of *inter-domain paths* between domains.

Associated with each domain in a transit-stub graph are two routing tables, an intradomain table and an interdomain table. The intradomain table is a simple two-dimensional array shared by all nodes in the domain; entry i, j in the array contains the index (in the domain graph) of the first hop on the path from i to j. This array is constructed by running the Floyd-Warshall all-nodes shortest path algorithm on the domain graph with edges to other domains removed. Each interdomain routing table is an array of entries, each of which consists of a string denoting a destination and forwarding information for that destination. The destinations represented in this table are domains. For each destination domain, we store the *egress* node in the same domain, that is, the node from which an edge connects to the next domain in the domain-level path to that destination. In addition, we store *domain-level path* (corresponding to the "AS path" BGP attribute) for that destination. The table contains one entry for each transit domain, plus one entry for each stub domain to which the domain is directly connected. The way nodes are identified in transit-stub graphs[2] makes it possible to do a kind of longest-matching-prefix lookup on this table to find the destination domain.

The forwarding lookup function computes next-hop information for a destination in a different domain as follows. First the interdomain table for the origin domain is consulted to determine the egress node to get to the destination domain; the intradomain table for that domain is then consulted to determined the next hop to the egress node. At the egress node a different procedure must be used: first the next domain along the path to the destination is determined, using the AS path attribute from the interdomain table; then the neighbors of the egress node are examined until one is found that belongs to that domain; it is returned as the next hop.

Although this approach results in a more complex forwarding computation, it reduces the size of the forwarding table from $O(n^2)$ to $O(DT + DM^2)$, where D is the total number of domains, T is the number of multihomed domains, and M is the size of each domain. As an example, we have constructed a transit-stub graph with 15,000 domains (approximately 40 transit domains) and over 300,000 nodes total. The routing information for this graph occupies about 18 Megabytes (without compression); the simple all-to-all (n^2) table would provide constant-time lookup, but would require 90 Gigabytes of space.

We have created a `traceroute`-style program to demonstrate the use of the routing utilities. Figure 2 shows sample output on the 300K-node graph mentioned above.

We use an algorithm that simulates BGP operation to generate the interdomain forwarding tables. Our implementation algorithm supports the specification and enforcement of path-selection (import and export) policies for each multihomed domain, as described in the next section. (The interdomain tables of single-homed domains are trivial.) Once the routing tables have been computed, the

[1]The Stanford GraphBase includes an implementation of Dijkstra's algorithm, but it finds the shortest path between *two* nodes.

[2]Each node in a stub domain is identified by a string that encodes a transit domain, transit node relative to that transit domain, stub domain relative to that transit node, and node in that stub domain. Names of nodes in a transit domain only have the first two components.

```
laurel> gtitmtr ts300504-0.gb ts300504-0-3.rt 100000 300000
Source = S:8.8/13.2 Destination = S:24.20/5.12
Intradomain routing table loaded
Interdomain table loaded
Routing tables loaded in memory
1: S:8.8/13.4
2: S:8.8/13.14
3: S:8.8/13.17
4: T:8.8
5: T:8.11
6: T:8.9
7: T:16.13
8: T:16.8
9: T:16.18
10: T:24.0
11: T:24.2
12: T:24.20
13: S:24.20/5.19
14: S:24.20/5.12
Lookup time: 90077 microseconds
```

Figure 2: Output of GT-ITM "traceroute" command

routing information is stored in a file (whose name is derived from that of the input graph file). To ensure that the routing information is used with the same graph for which it was generated, the checksum from the graph file is stored along with the table information.

4. INTERDOMAIN POLICY

Routing based strictly on shortest paths is not "realistic" in the sense that the paths followed by packets in the real Internet are determined as much by *policy* as by distance, hops, or other metrics. For example, given a choice between a route advertised by a service provider and another route to the same destination advertised by a provider who is a customer, in general the customer route will be preferred (because it generates income). The interdomain protocol BGP is designed to support selection of paths based on domain-level policies of this kind. This means that the path followed by packets between two nodes may not be the "shortest" one by any of the usual metrics. To achieve realistic routing in simulations, it is therefore desirable that the GT-ITM routing/forwarding facility support specification and enforcement of routing policies.

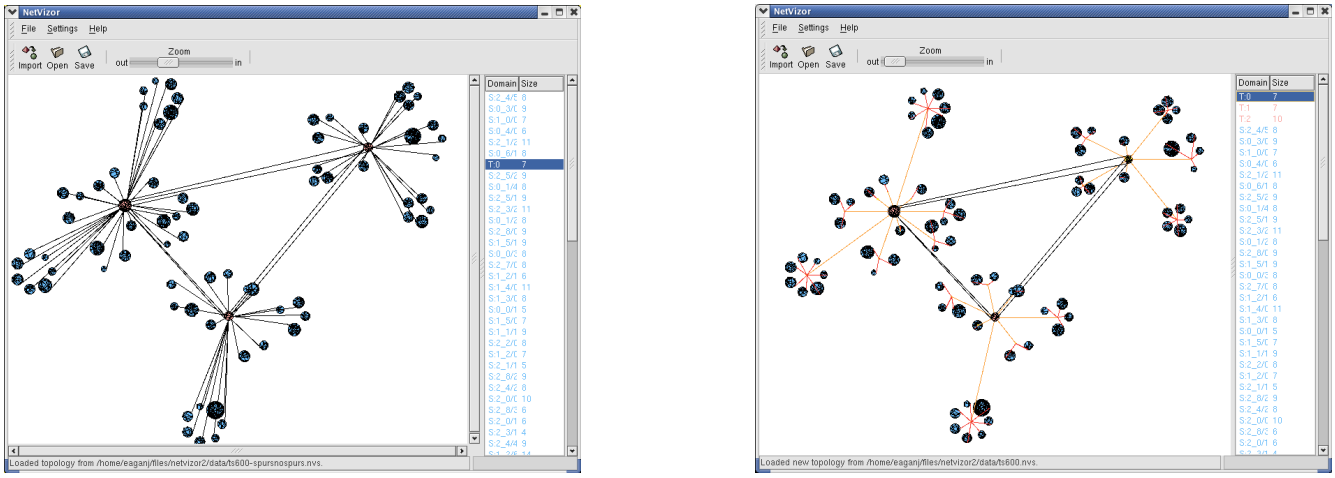
The new release supports a simple language for specifying policies, and allows policies to be specified for particular domains. If no policy is explicitly stated for a domain the default policy is used. Policies are of two kinds: *Import* policies assign local preference values to routes based on the domain-level path associated with the route. Such policies can be used to favor routes through particular domain-level neighbors (e.g. the above-mentioned preference for customer routes over provider routes). *Export* policies allow the user to define rules for exporting the route information for a particular domain to its neighbor. A general example of the need for such policies is that routes learned from *peers* (i.e. neighbor domains that are not customers and not providers) may or may not be advertised to other domains.

Clearly it would be cumbersome to specify policies for graphs with thousands of domains. We therefore provide a mechanism for automatically generating policies that reflect the customer-provider and peer-peer relationships found in the Internet [3]. This mechanism is based on the observation that provider domains in the Internet can viewed as organized in *tiers*, with a fully-connected core of domains that are not customers of any other domain forming "Tier 1", "Tier 2" domains being the customers of Tier 1 providers, etc. For any given transit-stub graph it is possible to assign customer-provider relations automatically by considering the domain-level graph. First a clique is found and designated as the core (Tier 1). Then a breadth-first search is performed, with neighbors of Tier 1 domains becoming their customers or Tier 2 domains, neighbors of Tier 2 become Tier 3, and so on.

Once customer-provider relationships are assigned in this way, it is straightforward to generate import (local route preference) and export (which routes are advertised) policies consistent with those relationships. We have created program `policytool` to automate the process. It takes a graph file as input, generates the policy information as described above, and then stores the information as a policy file. (In assigning domains to tiers, the desired size of the core is given as input; a fast constraint-satisfaction system is used to find a clique of the given size.) The resulting policies are stored in a file, which can be provided as input to the routing table computation described in the previous section. Policy files can of course also be modified or created by hand. Thus the user can generate different policies for the same graph and see the effect on forwarding.

5. VISUALIZATION FEATURES

One of the most commonly-requested additional features is support for visualization of the graphs generated by GT-ITM. Based on our own experience and interactions with other researchers, we find that visualizations are desired for several different purposes. First, a pictorial representation of a topology can provide some high-level intuition about the structure of the graph. Such intuition may allow a coarse-grain sanity check on the correctness of simulation code and/or the ability to trace "by hand" the intermediate expected results as a form of debugging. Second, a visualization may help identify locations in the topology that have particular properties, so they can be used as sites for particular entities (e.g., servers, clients, content distribution caches), in simulation runs. Third, as researchers have attempted to evaluate topology generators, they have generally applied standard graph theory metrics. Visualization may help shed light on alternative graph features that are particularly discriminating in a generator. Finally, for documentation



(a) Traditional Graph View

(b) Spurred Graph View

Figure 3: A comparison of `NetVizor` output using (a) the traditional graph view and (b) the spurred graph view.

purposes, it is useful to be able to include pictures of topologies in papers describing research results.

Working with colleagues in information visualization, we have build an interactive tool—`NetVizor`—for the layout of transit-stub graphs. The tool provides several options for layout, including two options that involve the user in the layout process. By involving the user, the intent is to develop and strengthen the mental model of the structure. Once an initial layout is complete, the tool allows interactivity of several types, intended to enhance the mental model and support more scalable versions of visualization. We describe the layout and interactivity features in more detail below.

5.1 Layout and Interactivity

Several layout features are common to all views. A domain is represented by a circle, with nodes on the circumference. The size of the circle is automatically chosen based on the number of nodes in the domain. In all cases, the intra-domain structure is automatically generated by the tool.

Three options for initial layout are provided. The **fully-automatic** mode will construct the entire topology automatically, placing transit domains on a circle and fairly far apart from one another, then surrounding a transit domain with its associated stub domains[3]. The **partially automatic** or **manu-matic** mode combines manual layout with automatic layout. The goal is to involve the user in the high-level details of the layout, while automating the tedious aspects (e.g., of intra-domain connectivity). The user is responsible for the placement of transit domains; the system automatically places the associated stub domains in a circle around each transit domain. In the **fully-manual** mode, the user places all domains.

After initial layout, the user may interact with the graph. This interactivity can be used to improve the layout to be more visually pleasing or to increase intuitive understanding of structure. Nodes and domains support selection and dragging, with some understanding of domain-level semantics. For example, moving a transit domain causes all of its stub domains to move as well.

[3]Multihoming of stub domains complicates the task of layout. If many domains are multi-homed, layout with some user involvement is probably appropriate.

The user can also alternate between two different views of the graph. The **traditional** view uses one node per router, with an edge between each pair of connected routers. The **spur** view aggregates the edges from multiple stub domains to the same transit node by providing a single "spur edge" from the transit node with branches to each stub domain. This view is intended to reduce visual clutter, while preserving visualization of domain-level connectivity. Figure 3 shows an example graph in traditional and spur views. This is a 600 node topology with three transit domains.

5.2 Implementation and Extensions

`NetVizor` is implemented in C++ using the GNOME libraries. It runs on any system supporting GNOME, including Linux, Mac OS X, and Solaris. The basic implementation is extensible. Support exists for additional views (e.g., bar charts, histograms), though these views are not currently implemented. The data format allows arbitrary nominal or quantitative values to be added to the graph topology to encode additional information.

6. OTHER ENHANCEMENTS

In addition to the enhancements described above, we have made some other changes to the tools that make up GT-ITM. Included below are possible additional enhancements that we could make if desired by the research community.

Alternative specification of domain-level graph. It is known that the domain-level structure of transit-stub graphs produced by GT-ITM does not reflect some of the structural characteristics of the "real" AS-graph as seen by interdomain routing protocols [1, 7]. Other topology generators (e.g. Brite [6] and Inet [8]) have focused on constructing models that more accurately reflect the Internet's AS-level structure. In order to provide this capability in GT-ITM, we have separated construction of the domain-level graph from the rest of the process of generating a transit-stub graph; the domain graph is now given as an input to the latter part of the process. This makes it possible to construct the domain-level graph using

any method desired—including other topology generators—and expand it into a transit-stub graph.

More efficient route computation. We have implemented a piecewise route computation that greatly improves the efficiency of all-to-all route computation for graphs in which many stub domains are single-homed. The scheme computes routes independently within each stub domain and in a graph consisting of the border router for each stub and all the transit nodes. A complete end-to-end route is then constructed by piecing together routes in each stub domain with the route from origin domain border router to destination domain border router. The efficiency gains are considerable; for example, an all-pairs computation which took hours to run now takes 10s of minutes. An additional enhancement would involve correct operation in graphs with multi-homed stub domains.

Bug Fixes. We have fixed some errors in the program, including one that caused input parameters to be interpreted differently than the specification indicated. A "bug-compatibility mode" is included for those who need to preserve past behavior.

7. CONCLUSION

We believe the improvements we have made to GT-ITM will enhance its utility for the research community. The authors welcome feedback on these features and suggestions for others to be added.

Acknowledgment

The authors are grateful for the support of the US National Science Foundation under grants ANI-0081557 and ANI-0082318.

8. REFERENCES

[1] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On Power-law Relationships of the Internet Topology. In *SIGCOMM*, 1999.

[2] http://www.cc.gatech.edu/projects/gtitm/.

[3] L. Gao and J. Rexford. Stable Internet Routing Without Global Coordination. *IEEE/ACM Transactions on Networking*, 2001.

[4] T. Griffin, F. B. Shepherd, and G. T. Wilfong. Policy Disputes in Path-Vector Protocols. In *ICNP*, 1999.

[5] D. E. Knuth. *The Stanford GraphBase: A Platform for Combinatorial Computing*. ACM Press, 1993.

[6] A. Medina, A. Lakhina, I. Matta, and J. Byers. BRITE: An Approach to Universal Topology Generation. In *International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems- MASCOTS*, 2001.

[7] H. Tangmunarunkit, R. Govindan, S. Jamin, S. Shenker, and W. Willinger. Network Topology Generators: Degree-Based vs. Structural. In *ACM SIGCOMM*, 2002.

[8] J. Winich and S. Jamin. Inet-3.0:internet topology generator. Technical Report CSE-TR-456-02, University of Michigan, 2002.

[9] E. W. Zegura, K. Calvert, and S. Bhattacharjee. How to Model an Internetwork. In *IEEE INFOCOM*, 1996.