2006

EXTENDING AND ENHANCING GT-ITM

Aditya Namjoshi
University of Kentucky, aditya@qualcomm.com

ABSTRACT OF THESIS

EXTENDING AND ENHANCING GT-ITM

GT-ITM is a topology generation tool. Since its release GTITM is widely used in the scientific community for network simulations. GTITM is extended to support routing on its topology. The routing algorithm used for interdomain routing attempts to emulate the BGP routing protocol seen on the Internet. It uses a policy file if supplied to make routing decisions. An additional functionality provided with the tool is the ability to automatically generate policy file for large graphs.

KEYWORDS: GT-ITM, Topology, Routing, Internet, BGP

Aditya Namjoshi

Aug 29, 2006

EXTENDING AND ENHANCING GT-ITM

By

Aditya Namjoshi

Dr. Kenneth L Calvert, Computer Science
(Director of Thesis)

Dr. Grzegorz W. Wasilkowski
(Director of Graduate Studies)

Aug 29, 2006
(Date)

RULES FOR THE USE OF THESIS

Unpublished THESISs submitted for the Doctor's degree and deposited in the University of Kentucky Library are as a rule open for inspection, but are to be used only with due regard to the rights of the authors. Bibliographical references may be noted, but quotations or summaries of parts may be published only with the permission of the author, and with the usual scholarly acknowledgments.

Extensive copying or publication of the THESIS in whole or in part requires also the consent of the Dean of The Graduate School of the University of Kentucky.

A library which borrows this THESIS for use by its patrons is expected to secure the signature of each user.

Name	Date

THESIS

Aditya Namjoshi

The Graduate School
University of Kentucky
2006

EXTENDING AND ENHANCING GT-ITM

THESIS

A thesis submitted in partial fulfillment of the
requirements for the degree of Masters in Computer Science
in the College of Engineering at the University of Kentucky

By

Aditya Namjoshi

Lexington, Kentucky

Director: Dr. Kenneth L Calvert, Computer Science

Lexington, Kentucky

2006

Table of Contents

List of Figures

List of Files

File Name	File Size
extendingandenhancinggtitm.pdf	24MB

Chapter 1

Introduction

1.1 Background

Researchers seeking to improve some aspect of the functioning of the Internet often test their hypotheses using some form of simulation. In many cases the network topology plays an important role in such simulations. Network topology refers to the relationships among the elements (channels and switches/routers) that make up the network. In the recent years, a number of tools have been developed to produce models of internet topology that are in some sense realistic. These tools try to emulate the essential characteristics of the Internet topology in order to provide a realistic test bench for researchers. One of the most important aspects of a network topology model is the way it determines the path or paths followed by a packet as they travel through a network. This routing aspect affects the performance of many algorithms.

A network essentially has two important attributes: Topology and Routing. A network may be represented as a collection of nodes connected to each other through links. Topology refers to this interconnection of the nodes with each other, whereas routing refers to the paths taken by packets from source nodes to destination nodes. A good network topology generator should be able to represent these two features, viz. topology and routing, in such a way that the topology is similar to the Internet topology and the routes are similar in nature to the routes taken on the Internet.

In this chapter, we first describe the topological structure of the Internet. Later we describe the intradomain and interdomain routing protocols. Then we shift our attention to Internet topology model generators, specifically GT-ITM, and understand the current limitations in the tool. We then outline the improvements to GT-ITM to be discussed in the remainder of the thesis.

1.2 Structure of the Internet

The Internet is divided into Autonomous Systems (AS's). Each AS is a unit of router policy: either a single network, or a group of networks that is controlled by a common network administrator (or group of administrators) on behalf of a single administrative entity (such as a university, a

business enterprise, or a business division). Each autonomous system in the greater Internet is also sometimes referred to as a routing domain. An autonomous system is assigned a globally unique number, sometimes called an Autonomous System Number (ASN). In this document we will use a more generic term *domain*, rather than AS. [1]

On the Internet different routing algorithms are used within a domain and outside a domain. RIP (Routing Information Protocol) and OSPF (Open Shortest Path First) are common intradomain routing protocols while BGP (Border Gateway Protocol) is the defacto interdomain routing protocol for the Internet. Networks within an AS use an intradomain routing protocol for message exchanges within the AS and use an interdomain routing protocol for message exchanges outside the AS.

Generally [1, 2], each domain on the Internet can be classified as a *transit* domain or a *stub* domain. Transit domains provide *transit* connectivity for other domains: that is, they carry packets whose source and destination are both outside the domain. Stub domains do not provide such transit services; only packets whose source or destination lies within the domain can be found in a stub domain. Stub domains contain most of the *end systems* in the Internet, and most traffic travels between the stub domains. This distinction is the basis for the *transit-stub* model used in GT-ITM (Georgia Tech Internet Topology Models). The figure 1.1 illustrates the different types of domains.

- *Transit Domains* correspond to service providers on the Internet. They offer connectivity for the stub domains to rest of the Internet. Transit domains are by definition *multi-homed*: a transit domain may be connected to multiple stub domains as well as other transit domains. Transit domains correspond to Internet Service Providers which provide Internet connectivity to smaller domains.

- *Single-Homed Stub Domains* have a connection to just one neighbor domain, i.e. they are connected to only one transit domain. Single-homed stub domains correspond to the "leaves" of the AS-level topology graph - for example, University campus networks with just one service provider.

- *Multi-homed Stub Domains* are connected to two or more domains. Such a domain may be connected to two or more transit domains, in which case it has multiple entry/exit points,

[1] Technically an AS is not quite equivalent to a routing domain, and there can be several routing domains within the same AS in which case each domain within that AS can independently speak a interdomain routing protocol with its neighboring domains. But for the sake of simplicity, and the way GT-ITM tool is designed, we consider a domain equivalent to an AS. As we will see in later sections, transit domains and stub domains, behave as AS's on the Internet, exchanging routing information.

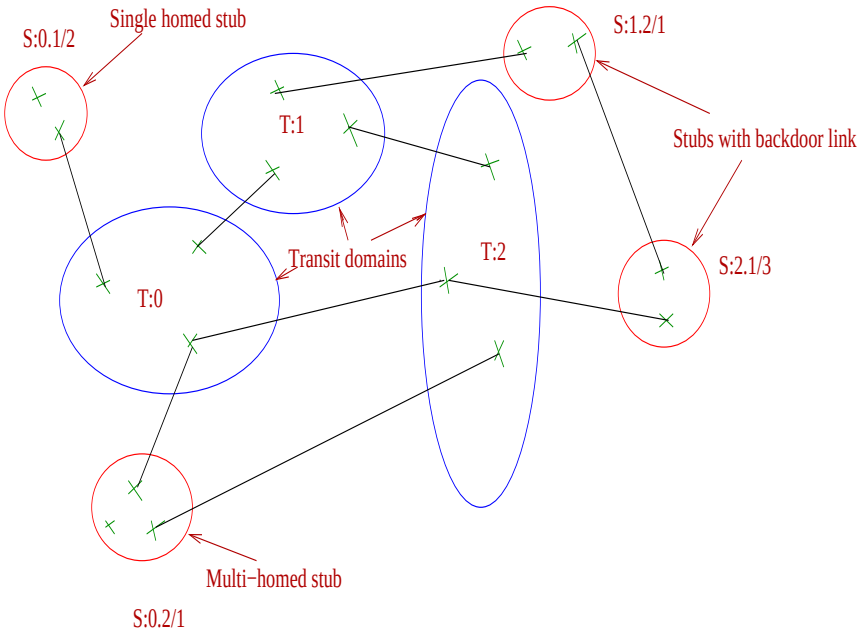


Figure 1.1: Hierarchy on the Internet

and packets may be routed to different exits for different destinations. Multi-homed stubs correspond to large campuses or companies having multiple providers.

Alternatively, a multi-homed stub may connect to only one transit domain, but have link(s) to other stub domain(s). The stub domains involved form a peering relationship and agree to exchange traffic among themselves over the backdoor link without going through their service providers. The main objective in setting up a backdoor link is to reduce costs by not transiting packets through a provider.

Domains on the Internet may form a provider-customer relationship or a peer-peer relationship based on the characteristics described above. These relationships are generally governed by the cash flow between domains. If a domain *X* pays domain *Y* for connectivity to the Internet, domain *X* is a customer of domain *Y*. For example, stub domains are customers of transit domains and stub domains connected by a backdoor link form a peering relationship. Similarly a transit domain *x* can be a customer of some transit domain *y*, and a peer of some other transit domain *z*. Research on the Internet topology [9] has confirmed that the domain structure of the Internet exhibits a hierarchy of at least 4 to 5 *tiers*. A tier on the Internet is a logical collection of routing domains such that all

domains in a tier form a provider or a customer relationship with domains in lower and higher tiers respectively. The top tier consists of the core, which is a dense interconnection of AS's. That is, in the core, every AS has a direct link to every other AS in the core and thus is a source of reachability for all other core AS's. The AS's in the bottom tiers typically have a lower edge degree[2], and hence have to depend on the core to route packets to some far off destination. The bottom tier consists of stub domains.

1.2.1 Internet Topology generators

In the last few years a good deal of study has been focused on Internet topology. Simulations and other experiments need models of topology of the Internet. These simulation models are called as Internet Topology generators. Such models are usually represented as graphs, in which the nodes represent routers and edges represent the channels. It is quite common for network simulators to generate large graphs to test the wide area performance of new protocols or to measure the characteristics of the network or to compare different protocols with each other. Because of the scale of the Internet and other considerations[3], nobody knows the actual topology of the Internet. To make up for the lack of knowledge about the actual graph, the topology generators generate the models stochastically. The challenge is to produce random graphs that have structural characteristics similar to those of the Internet. A lot of research has been aimed at determining what those characteristics are. Different researchers have come up with different graph generation methods, based on which characteristics they believe are important. Some focus on hierarchy, others on degree of nodes[4]. Some try to model the router level topology while others focus on mainly domain level topology. PLNG (Power Law Network Generator) is an example of degree based network generator, where the node in-degree and out-degree follows a power law. Topology generators like Inet, Tiers, GT-ITM [4, 1] take the hierarchy into consideration for modelling the Internet topology. Both types of topology generators try to match the essential characteristics of the Internet with some kind of tradeoff caused due to the preference given to either the hierarchy or the degree aspect of the topology.

[2]Here edge degree refers to the number of edges a domain has with its neighbor domains. For example if a domain A has 5 neighbors, but is connected to just three out of those, then the domain has an edge degree of three.

[3]Internet is huge and is constantly changing. Internet topology is in a state of constant flux because new domains are added to the Internet and new links are established between neighbor domains.

[4]Internet hierarchy refers to the provider-customer relationships between domains and the node degree refers to the number of edges a node has with its neighbors.

1.3 Routing in the Internet

As mentioned in previous section RIP and OSPF are mainly used for intradomain routing, i.e routing within an AS, while BGP is the main interdomain routing protocol on the Internet. Intradomain routing involves finding the shortest path from the source to the destination according to a common metric—mostly hop count or the distance between nodes. However, Interdomain routing is based on policies and the path followed from source to the destination makes use of domain-level polices to decide its route. In particular, the Internet's interdomain routing protocol (BGP) is designed to support selection of interdomain paths based on domain-level policies. There policies reflect, for example the customer-provider and peering relationships.

Routing within a domain is less complex than interdomain routing. This is mainly because of the use of a common metric to select paths. Hop count is typically used as a metric to decide which route is to be used; the route with the minimum hop count is chosen. There are two classes of intradomain routing protocol: Distance Vector and Link State. With a distance vector protocol, e.g. RIP, each node advertises distance information to its neighbors and with every subsequent advertisement a node gains information about the network and starts building its routing tables. On the other hand, with link state routing protocols, e.g. OSPF, each node advertises the status of its attached links to its neighbors; once all the information is gathered, each node runs a shortest path algorithm like Dijkstra's to compute the shortest path to each node and to populate its routing tables. The next subsection focuses on interdomain routing.

1.3.1 Border Gateway Protocol (BGP)

BGP, the current Internet standard for interdomain routing between the AS's, allows each AS to set its own policies for route selection. Policies are a set of rules, that help the BGP border router to select routes to a destination. Some of the things which policies can achieve are:

- If an AS has two neighbors A and B, a policy may give more preference to routes through neighbor A for some destination prefixes and more preference to neighbor B for other destination prefixes.
- A policy may reject a route if a route violates some defined condition. For example, a route may be rejected if it has been received via an AS which is deemed untrustworthy.

BGP, being an interdomain routing protocol, deals with routes at the AS level. Thus BGP treats the Internet as an AS graph with each AS labeled with some set of addresses (prefixes) that

are reachable from the border router in that AS. These border routers (entry-exit nodes in the AS) exchange information about the routes they know with other border routers in neighbor AS's. On receiving a packet, a BGP speaker decides to forward the packet to an appropriate neighbor based on the prefix information it has received from all neighbors and the packet destination.

When connection between BGP neighbors is first established, each BGP node advertises its presence to its neighbors. As information about nodes is propagated through the network, each BGP node starts building a routing table, which it can consult to find a path to a particular address. When changes to the routing tables are encountered, BGP routers send to their neighbors only those routes which have changed. BGP routers do not send periodic routing updates and do not advertise routes that are not installed in the local routing table (i.e. that are not being used to route packets).

Routes learned via BGP have *attributes* associated with them, which are used in choosing the route that will actually be used from among multiple paths to the destination. Following is a (greatly abbreviated) discussion of the BGP route attributes and the selection process.

BGP Attributes

- **Network Layer Reachability Information (NLRI):** (address prefix) defines the set of destination addresses of the route being advertised. For example, the network with network number 202.54.10.* will advertise a route with NLRI 202.54.10.*. NLRI is used for comparing destinations.

- **Local Preference:** (number) Local preference is the basic mechanism for implementing import policies and is not advertised to other domains but is used within the domain to assign preferences to routes. This attribute is set by the domain administration and the routes with a higher local preference are preferred over others. As an example, a provider will prefer a route through a customer to one through a peer.

- **AS Path:** (list of ASN's) When a route transits an AS, the AS Number is added to an ordered list of identifiers that records the sequence of AS's through which the route has passed. This serves two purposes. First, it is used in the route selection process as described below. Second, it is used to detect cycles: if adding the domain's ASN to the AS Path forms a cycle then the route is discarded. For example if a BGP node in AS 8 receives a route with AS Path (1,4,8,7,6), then it will discard this route as the AS has already processed this route before, and adding 8 again to this AS path will form a cycle (8,7,6,8)

- **Nexthop:** (address) This is the IP address of the border router in the neighbor domain. The routers in a multi-homed domain which are connected to routers in other domains are the border routers in that domain. Every time a route advertisement leaves a domain the border router in that domain attaches its address to the *nexthop* attribute. For a particular destination, the nexthop attribute of the route indicates the neighbor domain through which the packet has to be forwarded to reach to that destination.

There are several other attributes, but the ones mentioned above are most important for our purposes.

BGP Route Selection Process

The BGP route selection process determines which routes will be used for interdomain routing and advertised to other AS's. Routes on the border router are maintained in RIB's (Routing Information Bases). A BGP speaking router maintains three RIB's viz. RIB_in, RIB_local, RIB_out for storing the route information it has received. RIB_in contains routes received from its neighbors (minus any routes with cycles in their AS path). The *local preference* for each route is then determined from the policy information and the route with the highest local preference is installed in the RIB_local and used to route packets. If two routes have the same local preference value, the route with the shortest AS path is selected. Export policies of an AS, then further select a subset of routes from RIB_local to be placed in RIB_out for advertisement to neighbors. Note that the nexthop attribute is not used in the selection process, it just determines where to forward the packet once the route has been selected.

The routes stored in RIB_local are aggregated to reduce the size of the RIB's. For example, if routes to IP prefixes 172.168.224.00/24 and 172.168.224.00/20 both use 10.1.0.2 as next hop, the destinations can be aggregated into a single destination prefix 172.168.224.0/20 if no other *.*.*.0/24 prefix matches 172.168.224.0/20.

1.4 Introduction to GT-ITM

GT-ITM (Georgia Tech Internet topology models) as the name depicts, is an Internet topology generator [2]. Since its release GT-ITM has been widely used in the scientific community for network simulations. It is implemented on top of SGB (Stanford Graph Base) [5], a flexible collection of data structures and algorithms for creating, storing, and manipulating abstract graphs. GT-ITM

supports the creation of random graphs that have a variety of structures, as well as storage of such graphs in a portable file format. GT-ITM's *transit-stub* model attempts to create realistic topology with a two level hierarchy[5] and appropriate edge weights to implement a default routing policy between domains[6].

GT-ITM lets graph generation parameters be specified in a configuration file. Using the configuration file, the size of the graph as well as various parameters that control graph properties, such as edge probability factor, number of extra edges etc. can be specified. Below are the list of parameters user can specify in the configuration file.

- *Method of generating routing tables:* User can specify whether he wants to generate a random, hierarchical or a transit stub graph.
- *Number of graphs:* Number of graphs to be generated.
- *Initial Seed:* Seed to generate random numbers for graph creation.
- *Number of stub domains per transit domain:* Average number of stub domains connected to a single transit domain.
- *Random transit-stub edges:* Extra edges to be placed between transit domains and stub domains
- *Random stub-stub edges:* Extra edges to be placed between stub domains.
- *Probability of double edges between transit domains:* If this parameter is 1, there will be double edges between the transit domains which are connected. If this parameter is 0, there will be a single edge between connected transit domains. Any number between 0 and 1 would indicate the probability of having double edges between connected transit domains.
- *Number of transit domains:* Number of transit domains.
- *Edge method:* This parameter specifies the method of placing edges between transit domains.
- *Edge density between transit domains:* The edge density indicates how densely the transit domains are connected with each other. Edge density of 1 indicates that all transit domains have an edge with all other transit domains.

[5]Transit-Stub model forms a two level hierarchy with transit domains in the top tier and stub domains in the bottom tier

[6]The default policy takes care of the fact that multi-homed stub domains do not provide transit service between two domains.

- *Average number of nodes in the transit domains.*
- *Edge method.* Method of placing edges between the nodes of transit domains.
- *Edge density between the nodes in the transit domains.*
- *Average number of nodes in the stub domains.*
- *Edge method.* Method of placing edges between the nodes of stub domains.
- *Edge density between the nodes in the stub domains.*

The generated graph is stored in the Stanford Graph Base's file format, which can later be read in for simulation and other purposes. Along with this basic tool to generate graphs, GT-ITM offers other tools for evaluating some of the important graph properties and for converting the graph to a human-readable format to get some better understanding of the graph. The original GT-ITM comprises

- A command-line program that controls the creation of random graphs according to various models (including the transit-stub model) 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.

Limitations of GT-ITM

Since its release GT-ITM has not been modified except for few bug fixes and other minor changes. Below are some of the known limitations of the GT-ITM tool.

- GT-ITM does not provide a mechanism to do routing on its topology and the user has to supply an implementation.
- GT-ITM generated topology is essentially a two level hierarchy (i.e., Transit-Stub), whereas the real Internet seems to have more hierarchy in its structure [9].
- The Degree distribution of *domains(ASs)* in GT-ITM does not look like that in the Internet. It is rather uniform, where the Internet's AS-level node degree looks at least something like a power-law distribution.

Taking these three limitations into consideration, the problem statement of the thesis is defined in the next section.

1.5 Problem Statement

The problem we are trying to solve is two-fold: routing and topology. The idea is to generate routing tables for the GT-ITM topology, such that the routes are realistic in nature. Also the topology on which the routes are generated should be a reasonably measurable representation of the Internet.

1.5.1 Scalable Routing

GT-ITM by itself is a topology generation tool. This tool is used by researchers to generate graph models for simulation purposes. Most of the simulation experiments are based on measuring or evaluating some of the important Internet characteristics: latency, bandwidth etc. It is quite common to generate large graphs to test the wide area performance of new protocols, or to measure characteristic of the network, or the interaction of different protocols with each other, etc. In order to do these simulations its important for the simulator to have some means of mapping a destination to a next hop from a particular node, so as to be able to route the traffic generated in the simulation in a realistic way. In other words, some means for creating routing tables is required for routing. GT-ITM does not presently include such capability.

Current Techniques

Various solutions are possible in order to generate these routing tables. One simple but non-optimal way is to construct a big 2-D matrix where entry *i,j* contains the next hop on the shortest path from node *i* to node *j* as is done in Floyd Warshall algorithm. However, as stated earlier this is a non-optimal solution both in terms of space and time and the solution to use Floyd-Warshall is not a scalable one. The Floyd-Warshall all pairs shortest path algorithm has a complexity of the order of n^3, where n is the number of nodes in the graph[7] . Thus, with the increasing size of the graph the time needed to compute the all node shortest path increases rapidly, soon reaching limits of practicality. The space requirement for this solution too is very high as it requires storing a matrix of size n^2. This demands a lot of memory, and certainly this is not a viable option for large graphs.

[7]Using Floyd-Warshall, each node in a graph of n nodes, can compute a path to all other nodes in n^2 time. So the time required for building routing tables for the entire graph is n^2 x n = n^3.

However, there is another observation to be made in this context. The approach of using Floyd-Warshall forsakes all the advantage that is offered by a GTITM transit-stub graph. Floyd Warshall regards all the nodes in the graph as part of a flat graph, ignoring the hierarchy among the nodes in the graph. This is not the case with the Internet as we have already observed, different routing algorithms are used within a domain and outside a domain. This differentiation in intradomain and interdomain routing protocols is the first source of scalability in the Internet. This layering provides two advantages for routing information to any destination. First it prevents huge amount of information from being exchanged between nodes and second it reduces the information storage at each node. Thus using Floyd Warshall algorithm to compute routes on the GT-ITM topology, turns out to be an inefficient solution.

An alternative solution to the Routing problem is to use the divide and conquer approach:

1. Run Floyd Warshall within individual domains

2. Run Floyd Warshall over a graph in which each node represents a single domain

Merging information gathered from the first and the second step, routing tables for the whole graph can be generated. This solution is less expensive in terms of both space and time, as here we run Floyd Warshall on small blocks rather than one large block. Similarly, the memory required to store these small blocks of information is much less than that required for one large block. Consider a graph with n nodes, x domains, X_t transit domains, X_s stub domains and on average y nodes per transit domain and z nodes per stub domain. The complexity of calculating routing tables for the whole graph is $O(x^3 + X_t y^3 + X_s z^3)$. Complexity of calculating all pairs shortest path on a domain level graph of x nodes is $O(x^3)$. Similarly complexity of calculating all pairs shortest path with X_t transit domains with an average of y nodes and X_s stub domains with an average of z nodes is $O(X_t y^3 + X_s z^3)$. As achieved in the previous implementation of GT-ITM, we can guarantee that stub domains don't transit packets between domains. Though we will have some control over the path packets will follow using this approach, internet-like policy-based routing is difficult to achieve by assigning definite values to edge weights between the domains. (It is worth noting here that edge weights are assigned in GT-ITM transit-stub graphs in a manner that ensures that shortest-path routing always produces a path between two nodes in different domains that has the correct form, i.e. which passes through the first stub domain, followed by zero or more transit domains, followed by the other stub domain. However, the sequence of transit domains selected is always the one that yields the shortest sequence of edges.)

Need for Policy-Based Routing

Paths generated by Floyd Warshall are shortest path routes based on the edge weights defined. Routes obtained on the Internet are governed by domain level policies and may not be the shortest. On the Internet every AS needs to have some control over how routing information flows in and out of their network, which they achieve using domain level polices. So we need a mechanism to do policy-based routing on the GT-ITM topology. Some of the challenges in doing policy-based routing are:

- **BGP Simulation**: Since BGP protocol used on the Internet uses domain-level policies to make routing decisions we decided to do something similar to BGP in order to simulate Internet-like routing on GT-ITM graph. We decided to simulate the BGP protocol (to its bare minimum) ignoring the unessential details and focusing on the prime aspects of the protocol which are relevant and beneficial in our context.

- **Generation of policies for simulation**: We need to generate policies which can be used during BGP simulation. On the Internet an administrator of a domain specifies policies for his domain. The user needs to have the knowledge of the topology in order to write down policies and to study their effect on the routes taken. Since in our simulation, a user can generate multiple domains in the GT-ITM topology, he may need to specify policies for every such domain. If number of domains is small, then specifying per-domain polcies can be a simple task once the basic connectivity between domains is known. But for a graph with a large number of domains, specifying policies can be time-consuming, and hence we need a mechanism to automatically generate policies for large graphs with minimum user input. This user input should define some generic rules which every domain should follow, and we should be able to generate polices for every domain based on these rules.

One of the important advantages of simulating BGP-style policy-based routing is that it enables us to study issues related to policy and convergence of the protocol. Currently a lot of research is being done on BGP convergence [8, 10, 11, 12]. Number of researchers have suggested conditions on policies which ensure that the protocol converges and we get stable routing tables at each node. BGP divergence due to incorrect policies may cause unnecessary flooding of routing information and create unstable oscillations in the BGP protocol. The solution described in the next chapter will help the researcher to see the effect of policies on BGP protocol convergence.

1.5.2 Realistic Topology

In the GT-ITM transit-stub model, by definition transit domains transit the information sent by the stub domains. Stub domains form the terminal end points of the topology and use the transit domains for communicating information to other stub domains. Stub domains do not transit any information. Looking at the nature of route flow in the GT-ITM transit-stub model based on this definition, it may appear to be a 2 tier topology composed of transit domains forming the first tier and stub domains forming the second. Conceptually the transit level graph in the GT-ITM topology is treated as a flat graph and path taken by the route between transit domain is essentially shortest path based on some metric like hop count or edge weight. Whereas if we reduce the Internet into a GT-ITM like transit-stub graph such that domains which do not transit information become stub domains and the rest get classified as transit domains, we would observe that the routes taken between transit domains on the Internet may not be shortest path routes. This is because, the routes taken on the Internet are influenced by the domain policies which in turn are governed by the provider-customer-peer relationship existing between connected domains. So for GT-ITM topology to appear realistic, we need to do Internet-like routing on top of the GT-ITM topology. We try to solve this issue by providing an efficient routing solution on top of the GT-ITM transit-stub topology, such that the GT-ITM topology appears to be like the Internet topology.

1.5.3 Relation between Routing and Topology

Though we have described routing and topology as two separate problems, it is observed that solving one problem complements the other. One of the solutions is to increase the number of tiers in the GT-ITM topology. We tag each transit domain by a logical tier number such that the transit domains with lower tier numbers are the providers of transit domains with higher tier numbers if there exists a link between the two domains. Domains with the same tier number become peers if there exists a link between the domains. Thus by assigning tags to transit domains we define provider-customer-peer relationship in the transit level topology. We take these tags into account to construct routing tables for the GT-ITM topology. The method used to construct routing tables based on these tags, is explained in detail in the next chapter.

Looking at the Routing problem, it has been described that GT-ITM lacks routing support. Solutions like Floyd-Warshall result in shortest path routes. We wish to provide Internet-like routes which are based on some domain policy. Domain administrators select routes which are more commercially profitable and thus may end up selecting longer routes than the actual shortest path.

The commercial profitability here means that the domain has to pay less for the packet flow through other domains. This in turn is directly related to the commercial relationship between domains. Thus, solving the topology problem helps in solving the Routing problem. In the next chapter, we explain in detail the process of route selection based on domain relationships.

1.5.4 Conclusion

In this chapter we established two main objectives: need for policy-based routing on the GT-ITM topology and need of Internet-like topology. In the next chapter we describe our proposed solution to achieve the above mentioned objectives.

Chapter 2

Generation of Routing Tables

In this chapter we discuss our approach to generation of routing tables for the GT-ITM topology. This would extend the usability of the tool for more complex simulations, providing more control over the routing aspects of the network.

2.1 General Approach

It was discussed previously that routing on the Internet does not always follow the shortest path paradigm; instead, routing domains select routes based on interdomain policies. Our solution takes into consideration this policy-based routing paradigm in providing routing support for GT-ITM topology.

If we attempt to control the paths between transit domains, this would require the user to have complete knowledge of the transit level topology to specify policies to govern the paths. To make it easier to apply our methods to very large graphs, it was necessary to come up with a more generic way to define policies, with minimum user input and minimum knowledge of the generated topology. We used the idea presented in [10] to solve the problem of generating policies with minimum user input; a nice byproduct of this approach is that it guarantees protocol convergence and stable routes at each node [8].

In our implementation we first define transit domains as being customers, peers or providers of other transit domains (by assigning logical tier numbers) thus forming a n tier topology. The number of tiers n depends on the clique size in the topology and the edge density between the nodes and its computation is explained later in this chapter. Then, making use of the provider-customer-peer relationship existing in the n-tier topology[1], routing policies are generated. These routing policies are used to generate interdomain routing tables. The intradomain routing tables for each domain are generated by running Floyd Warshall all pairs shortest path algorithm for each domain. Once we have the routing tables, path from any source node to any destination node can be determined, by consulting the intradomain and interdomain tables.

[1] Domain in tier n is a provider of domain in tier n-1, if there exists a link between the two domains. Similarly domain in tier n-1 becomes a customer of that domain in tier n. Domains in the same tier, who share a link become peers

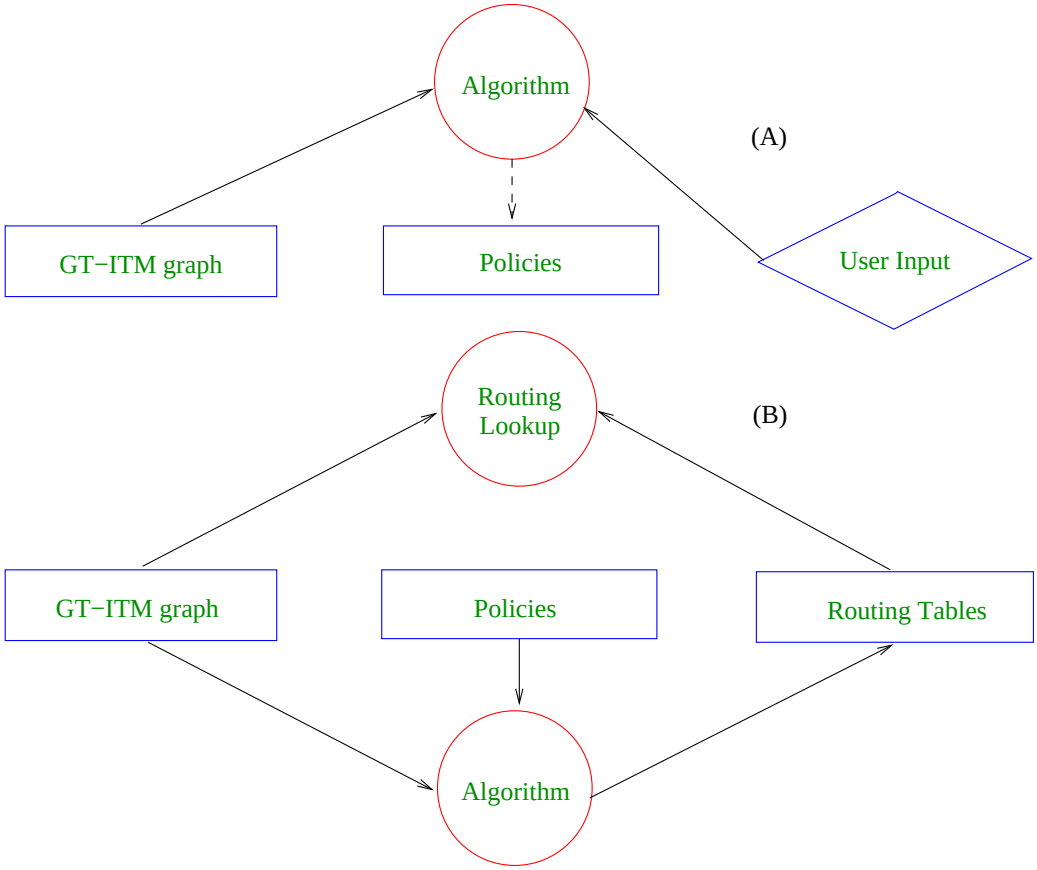


Figure 2.1: Routing Model

Policies are a set of rules specified in a file which is read by the routing file generation tool, to decide the paths. In Figure 2.1 (A) we see that the policies can be generated using an algorithm which reads the graph, or can be manually written by the user. As shown in Figure 2.1 (B), using an appropriate algorithm the GT-ITM graph is read and routing tables are generated. Policies may be provided as an input to the algorithm, and the generated routing tables will obey the policies. The routing lookup API reads the routing tables to give the next hop node from the source to the destination. In the following sections we describe in detail how each block in Figure 2.1 is designed and implemented.

There are two components to the problem of providing scalable routing services to simulations that use graph models:

- Computing and storing next hop routing information
- Using the next hop routing information to enumerate the path from the given source to the given destination at simulation time

We wish to separate these two components. That is, we wish to provide an ability to construct and store routing tables separately from the graph itself so that simulations can be run on the same graph with different routing tables.

2.1.1 Simplifications in design

We have made the following simplifying assumptions in our design to help us generate routing tables for the GT-ITM topology. These simplifications do not affect or violate the core working of the BGP protocol. Instead they ignore some of the more complex aspects of the protocol which may not be very important for the user during simulation.

- The (unique) border router in a single-homed stub domain always uses a default route for destinations outside the domain. This is usually the case in the Internet, as just one path is available for a packet to exit a domain.
- We assume that all border routers in a domain are always synchronized, by modeling each domain as a single node in a graph. In other words, we ignore the effect of the Internal BGP[2]

[2]On the Internet every AS can have multiple border routers. Each border router runs BGP protocol and exchanges routing information with its neighbors. This routing information is synchronized between all other border routers using the Internal BGP (IBGP) so that each border router has the knowledge of routes exchanged by other border routers in the AS.

- Routing information is not aggregated as it travels through the network. This is an artifact of the way GT-ITM identifies networks; it is never possible to replace a set of prefixes with a (shorter) prefix without some possibility of losing information.

Before we plunge into the specifics of our solution, we briefly describe the GT-ITM naming convention. This naming convention is important to understanding the partial aggregation performed in the route exchange process.

2.1.2 Naming conventions in GT-ITM

In GT-ITM, each transit-stub graph is represented as a graph structure. The nodes of the graph are stored in an array; edges are stored in linked lists associated with nodes. Thus nodes and edges can be accessed directly, via the data structures in the array. In addition, each node in the graph is assigned a name that encodes its position in the graph structure. These names have a well-defined syntax of the form

```
 <type
indicator>":" <transit domain id> "." <transit node id> ["/" <stub
domain id> "." <stub node id>]
```

where the type indicator indicates whether the node is a transit domain node or a stub domain node. For example, S:1.2/3.4 refers to node 4 in stub domain 3 connected to transit node 2 in transit domain 1. All numbering begins at 0. Similarly T:1.2 refers to node 2 in transit domain 1. In our implementation we use the $*$ symbol as a wildcard character; thus T:1.* refers to all nodes in transit domain 1 and S:1.2/3.* refers to all nodes in stub domain 3 attached to node 2 in transit domain 1. This naming convention is useful for advertising destination networks in routing; it corresponds to the use of prefixes to denote parts of IP address space. For example, the NLRI for a route originating from transit domain 1 can be represented as T:1.*. Similarly the NLRI for a route originating from a stub domain 2 connected to transit node 1 in transit domain 2 can be represented as S:2.1/2.* or T:1.* as the node belongs to a stub domain which is connected to transit domain T:1.*. As shown in Figure 2.2 this type of aggregation can be done for single-homed stub domains as they are connected to just one transit domain.

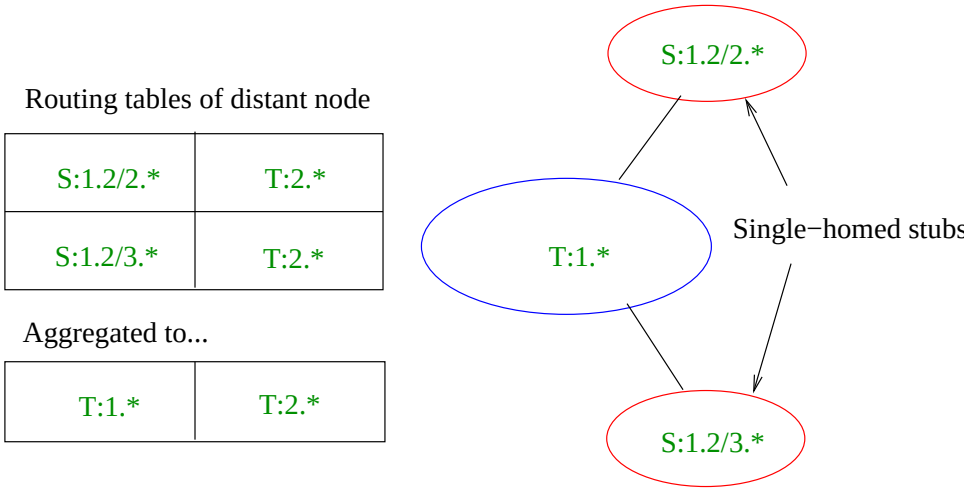


Figure 2.2: Naming and aggregation in GT-ITM

2.2 GT-ITM:Routing and Topology

As was noted in the introduction, the naive approach of computing the shortest path from every node to every other node does not scale very well. We therefore split the problem into two parts, namely computation of intradomain routing information for each domain, and computation of a domain level path connecting each pair of stub domains. The most straight-forward approach to more scalable routing is to run Floyd-Warshall within every domain to produce an intradomain routing table, and then to run Floyd-Warshall again on the domain level graph (i.e. an ”abstract” graph that has domains as nodes and an edge between two domains if there is an edge that connects a node in one domain to a node in the other). However, we wish to accommodate policies that restrict the selection of domain-level paths. The process by which this is achieved is described below; first we describe the representation of the routing information.

2.2.1 Routing table structures

Our scheme for scalable routing associates with each node in a transit-stub graph two routing tables: an intradomain table and an interdomain table. As shown in Figure 2.3, the intradomain table is a two-dimensional matrix shared by all nodes in the domain; entry i, j in the matrix contains the index of the node which is the first hop on the path from i to j. This matrix is populated by running the Floyd Warshall all-nodes-shortest path algorithm on the domain graph without edges to other

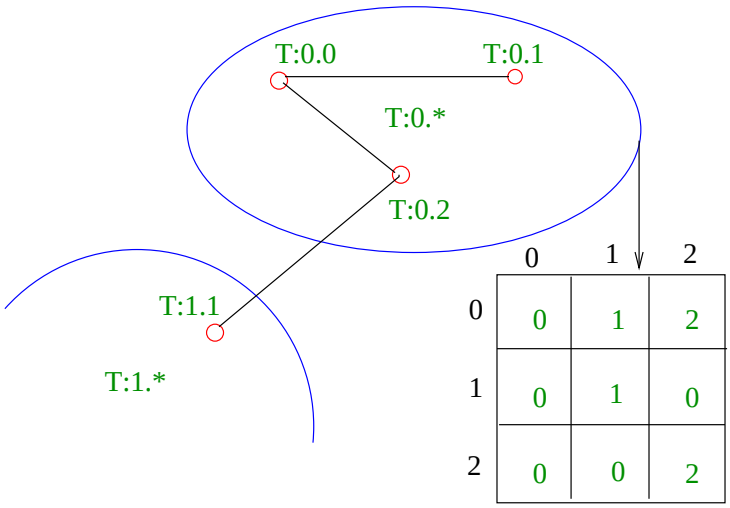


Figure 2.3: Intradomain routing tables

domains. (Because of the way GT-ITM graphs are constructed, the nodes belonging to a domain are contiguous in the vertex array of the overall transit-stub graph.) Intradomain routing lookup can be done by simply indexing into an array, using the node number of the destination as offset. Another way to view this information is that each node i has its own row of the table, and the next hop to node j in its domain is contained in column j of that row i.

The interdomain routing table is an array of entries, each containing a string representing the destination , the next hop information for that destination, and the AS path attribute associated with the route. Because strings are used to identify destination domains, interdomain lookup requires a longest prefix match similar to that used in IP forwarding. To simplify the implementation of the routing lookup API, the next hop information stored for an interdomain route is actually the border router in that domain, i.e. the exit node from the domain to reach that destination.

As shown in Figure 2.4, the interdomain table for a stub domain consists of a single entry representing the default route. There is just one copy of the interdomain routing table for the entire domain. There is one problem with this approach. The interdomain next hop information for the *exit router*, i.e. the last router encountered by a packet before it leaves the domain, needs to be different for that for other nodes, lest it forward interdomain packets to itself. As can be seen from the

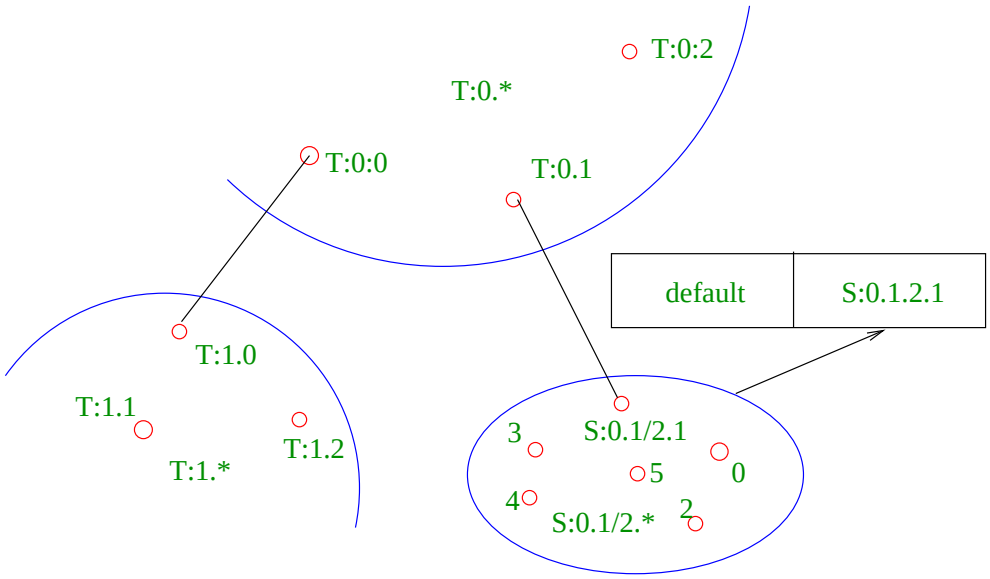


Figure 2.4: routing tables for single-homed stub domain

figure, the next hop for the border router S:0.1/2.1 is S:0.1/2.1. However, the interdomain routing lookup for the node S:0.1/2.1 needs to return T:0.1 instead of S:0.1/2.1. This could be handled by having a separate copy of the interdomain table for each border router. However, we chose to trade computation for space, and instead recognize this special case in the routing lookup code, so that the correct next hop for the border routers is returned. We explain this in detail in the next chapter where we discuss the routing lookup algorithm.

The situation becomes trickier in the case of multi-homed domains, but it is all handled in the lookup code. Figure 2.5 shows the interdomain routing table for transit domain T:0.*. If the destination is some node in T:1.* then the routing lookup for the border router should return the border router in domain T:1.* and not the border router in domain T:2.*. To determine the correct next hop in such cases, the lookup code consults some attribute information stored with the route. (The routing lookup algorithm is discussed in in the next chapter.)

2.2.2 Generation of routing tables

Generation of intradomain routing tables is fairly simple. Every domain is represented in terms of a two dimensional matrix M[i,j] where M[i][j] = 1 if there exists a direct link from node i to node j. If there is no direct link from node i to node j then M[i][j] = ∞. Here we are using the convention

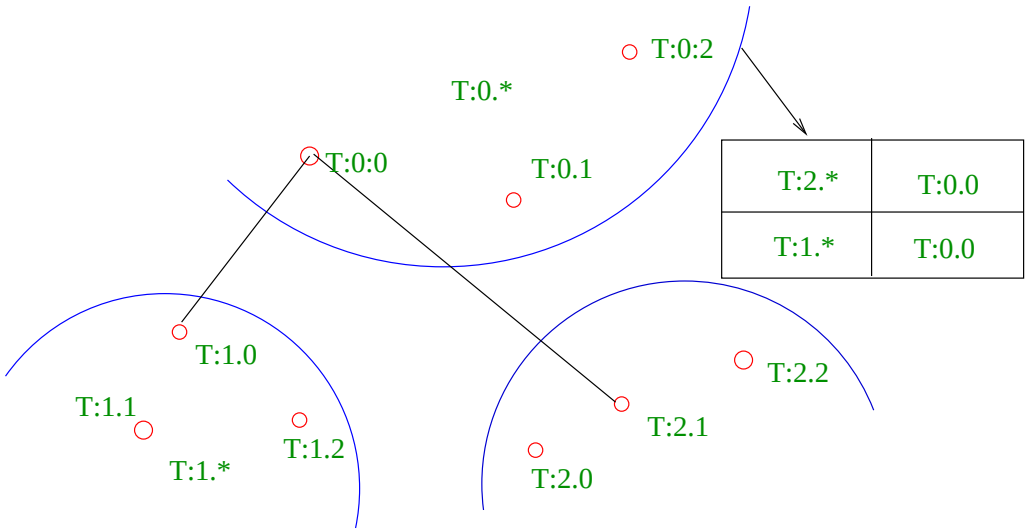


Figure 2.5: routing tables for multi-homed domains

of 1 or ∞ because we are considering hop count as a metric to determine shortest path and not the actual distance between the nodes. This matrix is given as an input to the Floyd Warshall algorithm which finds the shortest path from every node to every other node in the domain. Thus if there are t transit domains with x nodes per domain and s stub domains with y nodes per domain the complexity of computing the intradomain routing tables becomes ($tx^3 + sy^3$).

Generation of interdomain routing tables uses a much more complex approach than the generation of intradomain routing tables. Firstly, we need to get a domain level picture of the graph, as interdomain routing takes place between individual domains. Secondly, as described before we need to run a BGP-like routing protocol on this domain level graph. The interdomain routing policies may also be supplied as an input for deciding the routes. After running both the intradomain and interdomain routing protocols on the graph, we store the generated routing tables to a file, which then can be used later for simulation.

As described before we use a process similar to BGP to generate interdomain routing tables. When BGP first starts, the domains exchange routing information advertising their presence on the Internet. Similarly, we make each domain in the GT-ITM graph advertise its reachability information to other domains. As each domain gathers information about other domains, it decides which route it should store in its interdomain routing tables using its local policies.

Below we enumerate in detail the steps taken to generate routing tables.

- **Identify individual domains in the graph:** GT-ITM graph is a flat graph and vertices of the graph are stored in a linear array. Contiguous parts of this linear array correspond to the nodes of individual domains. The nodes of the first transit domain are stored first followed by nodes of each stub domain to which it is connected, followed by the next transit domain and so on. To generate intradomain routing tables, we mentioned that we need a two dimensional matrix representing the vertices and their edge interconnections. Similarly for interdomain routing, the domains need to exchange routing information. Thus it becomes necessary to identify the nodes of every domain, for running the intradomain and interdomain routing algorithms. To achieve that, for every transit and stub domain, we store the size (number of nodes) of each domain in a single array, and, using cumulative additions, a particular domains nodes can be accessed.

- **Run Floyd-Warshall all pairs shortest path algorithm on each domain:** Using the previous step, a two dimensional matrix is generated to represent each domain. This matrix is given as an input to Floyd Warshall's all pairs shortest path algorithm and the next hop matrix (intradomain routing tables) is generated for each domain.

- **Build a SGB graph with number of nodes equal to number of multi-homed domains:** Domains with two or more border routers are called multi-homed domains; they need to run an interdomain routing protocol to decide the border router to which the traffic has to be forwarded for destinations outside the domain. Multi-homed domains include all transit domains (assuming there is more than one) and multi-homed stub domains. Single-homed stub domains have just one border router connecting the domain to a transit domain. Ideally it is not required to run any interdomain routing protocol like BGP for such domains, because for any destination outside the domain the traffic can be forwarded via border router.

  The multi-homed domains in the transit-stub graph are identified and a new SGB graph with nodes equal to the multi-homed domains is created, so we have one node for every multi-homed domain. We call this graph the *interdomain graph* and the nodes in the graph are connected in a pattern identical to the way multi-homed domains are connected to each other in the main graph-that is if a link exists between a node in transit domain 1 and a node in transit domain 2 then there is a link between the node representing transit domain 1 and the node representing transit domain 2 in the interdomain graph. We use the node index in this linear graph array as the ASN (Autonomous System Number) of the domain for the purposes of constructing the AS path.

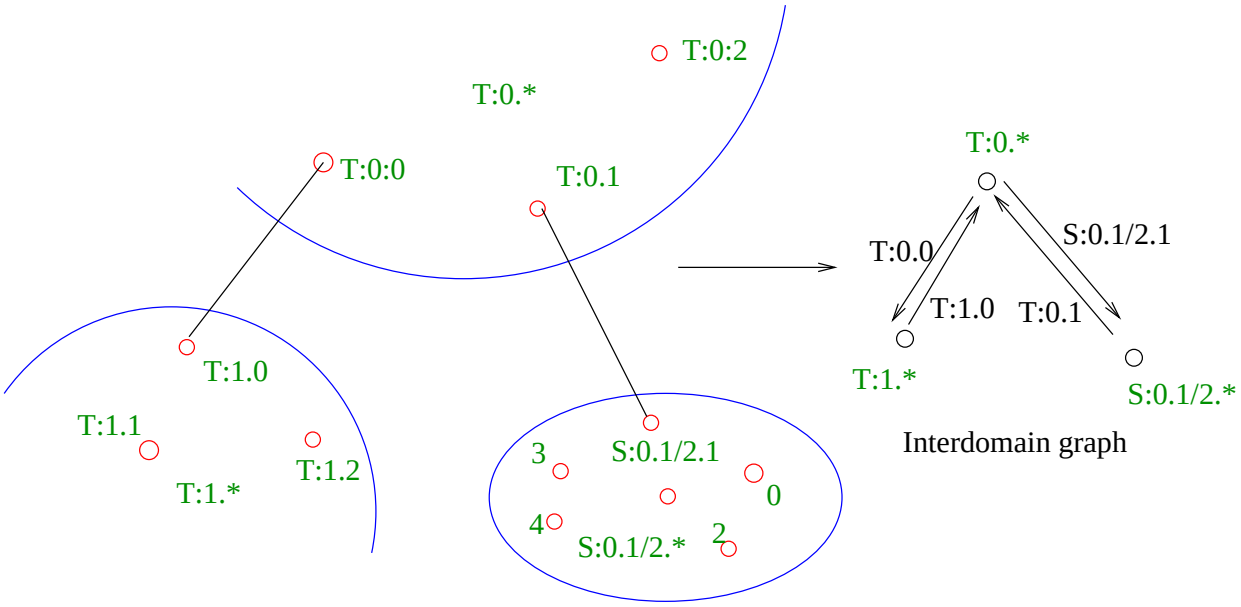


Figure 2.6: Store border routers at edges.

- **Identify the border routers in each domain:** A node in the interdomain graph represents a multi-homed domain in the main graph. During the route exchange process, when a node in the interdomain graph receives a route advertisement from some neighbor node, the next hop attribute of that route is set to the neighbor domain who forwarded this route. But here we have no information about the router inside that domain who forwarded this route as we are dealing with the interdomain graph where each node represents an entire domain. So we find the border routers in each domain by parsing the main graph, and store the name of the border routers connecting two domains in the main graph along with the edge connecting these two domains in the interdomain graph[3]. See Figure 2.6

- **Initialize the Routing Information Bases(RIBs):** Three RIBs (RIB_in, RIB_local and RIB_out) are defined for each node in the interdomain graph. Each domain starts off with just one route- the route to itself. During bootstrapping this route is advertised to the neighbors. For example, a transit domain T:1.* will initially have a route with NLRI T:1.* and its own AS number as the only entry in its AS path. This route is stored in the RIB_out of this domain to be exported

[3]For doing route traversal on the main graph, we need this border router information. Stanford Graph Base tool allows the storage of two auxiliary fields with an edge. We use one of these to store this border router information

to its neighbors.

- **Propagate routes:** Routes stored in RIB_out of a node are propagated to its neighbors. A queue is maintained to monitor the process of this route exchange. Each item in the queue carries the following information.

  – The neighbor node which should copy the routes from the current node's RIB_out

  – The current node

  To start with, the first node in the interdomain graph fills the queue with information about its neighbors. For example, if node 0 in the interdomain graph is connected to node 6, 10 and 12, then it would fill the queue with information $(6,0),(10,0),(12,0)$.Till the queue becomes empty, each entry in the queue is fetched and processed. The neighbor route is copied from its RIB_out and checked for cycles. If the route contains the current node's ASN in its AS path, that route is discarded, as it has already been processed by this node. This also prevents flooding of routing information. If the route is valid and contains no cycles, then it is copied into the RIB_in of the current node. If a route is new to the receiving AS(node), (i.e. a route with similar NLRI information is not present in RIB_local for that AS(node)), then the route's local preference for that AS is computed after referring to the policy file. (The generation of the policy file is discussed in greater detail in section 2.3.) If no policy file is specified, or the local preference for that route is not listed in the policy file, a default local preference of 80 is assumed. If a route with similar NLRI information is present in RIB_local of the current AS (node), then the local preference of the two routes is compared, and the route with a lower local preference is discarded. In case of a tie between the local preference values of the two routes, the route with a longer AS path is discarded and the one with the shorter AS path is stored in the RIB_local of the current node. If the receiving node is a transit domain, then it refers to its export policies to decide whether it can export the route to its neighbors, if so, it places the route in its RIB_out[4]. Once a node has copied the route to its RIB_out, the next step is to advertise this route to its neighbor. This is done by adding the appropriate entry to the queue.

- *Generate interdomain routing table for single-homed stub domains:* Single-homed stub domains are not a part of the route exchange process described above. Single-homed stubs have

[4]For stub domains the only route which is placed in RIB_out is the route to itself. This prevents other domains from transiting the traffic through a stub domain thus satisfying the basic property of the transit- stub model

only one route (the *default* route, which connects to one transit domain) to reach other domains. A route that has NLRI as '*' and next hop as the address of the border router in that single-homed stub domain is placed in the interdomain routing table for such domains (as discussed earlier, and shown in Figure 2.4).

The Queue of route exchanges becomes empty when no changes occur to any routing tables and no new route is learned by any of the vertex. Once the above computation is complete, the routing information is stored in a file for later access. The filename is the same as that of the graph, with the extension '.rt' (so if the graph file is ts600-0.gb then the routing information file is named as ts600-0.rt). The routing file contains the intradomain routing tables (next hop matrix for each domain) followed by the interdomain routing tables (RIB_local for each domain). To make sure that the routing file is being used for the same graph file for which it was generated, the checksum from the SGB '.gb' file is stored with the routing. The above mentioned file naming is slightly different if the policies are auto-generated, which is discussed in the section 2.3.

2.3 Policies

One of the important features of the tool developed for this thesis is the use of policies for making routing decisions. User is able to specify policies for route selection, which are read by the routing table generation program. Policies are of two types: import and export policies. Import policies assign a local preference to routes and are used within a domain to select among different routes to the same destination. Export policies filter out domains to which a route should not be forwarded. Policies are important because they determine the flow of traffic between transit domains.

Our objective is to generate a policy file with minimum user input. Internet domain-level policies are a result of the commercial relationships among domains. We use this commercial relationship model in generating a policy file that specifies policy for every domain in the Internet. In the existing GT-ITM tool, there is no information other than distance to consider in choosing routers. Policies provide the additional information for routing algorithms to use in selecting routes, by assigning roles of **provider**, **customer** and **peer** to domains. It is observed that route flows are in opposite direction to cash flow [10, 13, 14, 15]. In particular a provider will always prefer to route traffic through its customers because the customer pays for carrying that traffic. Its next preference is to route traffic via a peer. Figure 2.7 shows the general provider-customer hierarchy seen on the Internet.

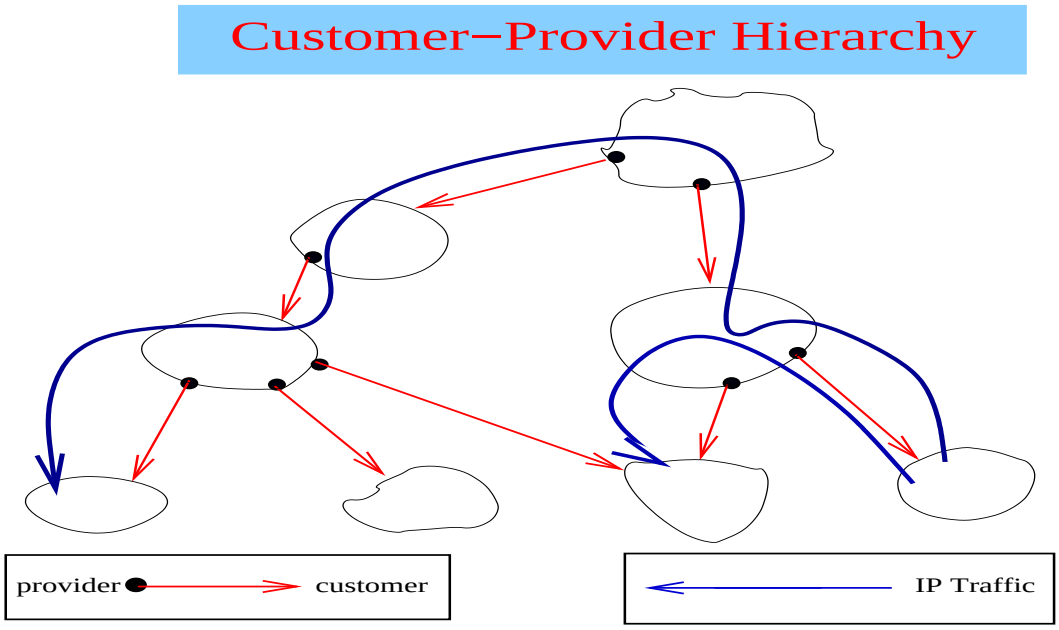


Figure 2.7: Hierarchy on the Internet

As seen in figure 2.8, domains in lower tiers route information through domains in higher tiers. We see that due to peering, domains can exchange routing information without going through their providers. Generally peers do not export routing information about their peers as show in Figure 2.9. Thus the route showed in dotted line is not allowed in such a scenario. Domains which belong to the same tier but are not directly connected need to go through higher tiers in order to be reachable. Policy information generated in GT-ITM guarantees such a behavior. A transit domain does not export the route information it received from a domain in its own tier to another domain in the same tier.

Our current implementation supports a simple language for specifying import and export policies. An input file defines the policies used by specific domains; if no policy is explicitly specified for a domain, the default policy is used. Import policies assign local preference values to routes based on their attributes. An example of an import policy is 'If Transit domain 0 receives a route to Transit domain 2 from Transit domain 1 then assign it a local preference of 100'. In the policy file this rule is stated as

```
0 1 2 100
```

This policy can be used to favor a route to Transit domain 2 through neighbor Transit domain 1. As another example, a transit domain x may not trust some other transit domain y, hence may prefer not to route through that transit domain. In that case a policy may specify ”If x receives a route to a destination transit domain ’*’ and transit domain y is in its AS path then assign it a local preference 50”. This filters out routes through transit domain y to any destination whenever there is a better (say, with default local preference) alternative. The wildcard character * denotes *any ASN*.

Export policies allow the user to define rules for exporting the route information for a particular domain to its neighbor. An example of an export policy is ’Transit domain 0 should not advertise a route received from Transit domain 2 for any destination ’*’ to neighbor Transit domain 1’. In the policy file this rule is stated as

```
0 * 2 1
```

This policy information enables the user to specify realistic policies and study the behavior of BGP for that set of policies.

The user can specify this policy file manually and provide it as an input to the routing file generation program. The route file generation program reads this policy file to determine which routes to store for a particular domain. It can become cumbersome for the user to write huge policy files for large graphs. Thus we need to devise a mechanism to automatically generate a policy file for a given graph with minimum user input. Below we describe the process of generating policy files for large graphs.

In the GT-ITM graph the stub domains form the leaves of the graph. Since stub domains do not provide transit service, they cannot be providers to other domains. But it is possible for a transit domain to act as a provider to some other transit domain. From [9] we know that the top tier of the Internet has an edge degree of 1 with every domain having an edge to every other domain in that tier. The bottom tiers have lesser edge degree and use the top tiers to transit traffic to the destination. So we try to label the nodes of the transit domain with their tier information (essentially a tier number) such that highest tier number consists of densely connected network of domains. To achieve this, first we find a clique of domains. Clique is a group of nodes such that every node in that group has an edge to every other node in that group. Then we run Breadth First Search (BFS) algorithm starting with domains in the clique. The detailed approach used to generate the hierarchical transit level graph is as follows.

The user is asked to enter the number of transit domains which should form the core of the graph. As mentioned previously, the core has domains which are fully connected, having an edge

from every domain to every other domain. In graph terminology this core is commonly referred as a clique. If a clique (of domains) of a specified size cannot be found then the user is prompted to enter a smaller size. If a clique of the specified size exists, we run breadth-first-search starting with nodes (domains) in the clique. The nodes which are at the same depth in the breadth first search tree are at the same tier in the graph. The depth of the BFS tree is the total number of tiers in the graph, with an extra tier comprising the stub domains. The number of tiers formed depends upon the size of the core and the edge density of the transit level graph. Below we describe the detailed process of generating tiers and creating a policy file for use.

- **Generate a transit level graph from the GT-ITM graph:** We need to identify all the transit domains in the main graph, and create a graph with nodes representing the transit domains in the main GT-ITM graph.. This graph will be used (instead of the main graph) in the next two steps.

- **Find a clique of the required size:** There is no efficient algorithm to find a clique of a given size (Finding a clique is a NP complete problem). A brute force algorithm for large graphs would be extremely time consuming, so we need to use some heuristics to determine a clique of a given size. We decided to use the stable model semantics to find a clique. This was developed in the Laboratory for Theoretical Computer Science at the Helsinki University of Technology[16].

  Stable model semantics is a tool for constraint programming, where we define a set of rules and run the stable model (smodels) program to find a solution for the rules we defined. In our case we defined certain rules which would result in finding a clique of particular size in a graph. The rules are defined by us and smodels program finds a solution for the defined set of rules.

- **Run breadth first search (BFS) on the reduced graph:** If the size of the transit level graph is *n* and the size of the core is *c* then create a graph of size *(n-c+1)*. This way we represent the whole core with one node. That node has an edge for every edge that connects a core node to a non-core node. Then, starting at this node we run the breadth first search algorithm. As we run the BFS algorithm at each step we store the tier number at each node.

- **Generate the policy file:** The information generated from breadth first search is used to generate the policy file, which contains import and export policies. Import policies are specified by assigning a local preference to routes from neighbors. Based on the hierarchical

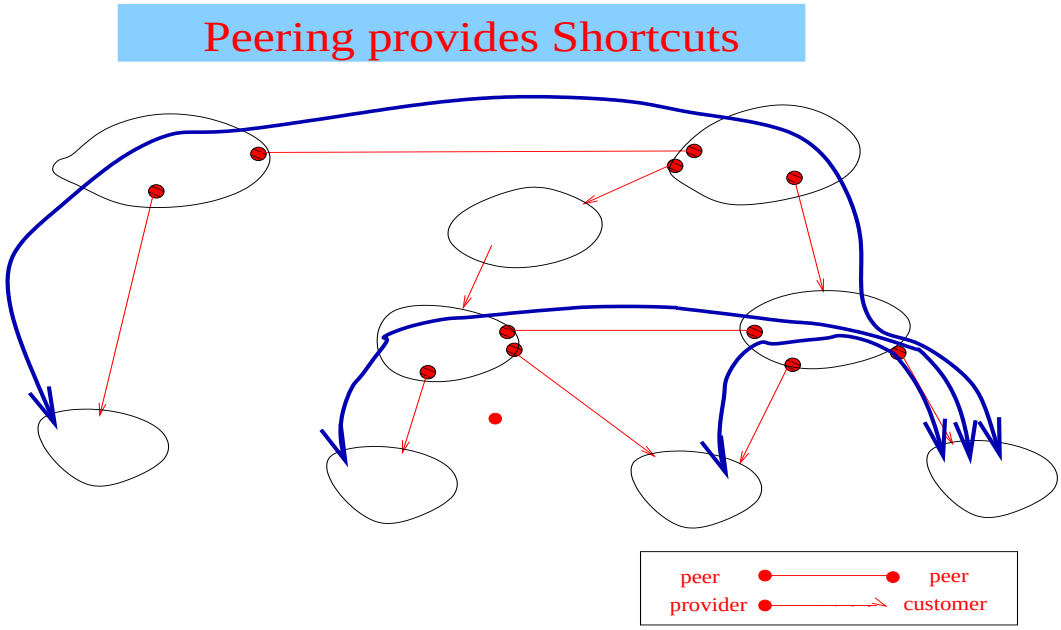


Figure 2.8: Peering

relationship between two adjacent nodes, each node gets an appropriate local preference values for routes received from the other. At every node in the transit level graph, the following algorithm is used to determine the local preference of routes from its neighbors.

1. If neighbor is a peer then assign local preference of x.
2. If neighbor is a customer then assign local preference of $x + 1$.
3. If neighbor is a provider then assign local preference of $x - 1$.

The pseudo code for generating export policies is as follows.

1. If a route is received from a node in the same tier, then don't send the route to neighbors within that tier or the tiers above.
2. If a route is received from a node in a higher tier, don't send the route to nodes in the current tier

- Store this information to a policy file, using the simple language described earlier.

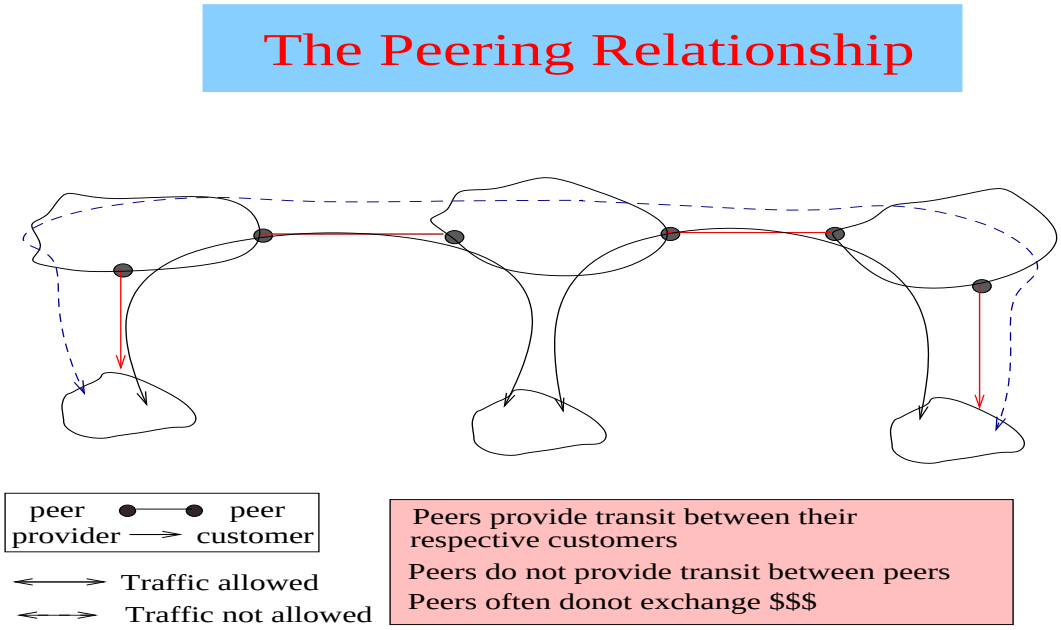


Figure 2.9: Routing flows

2.4 Realistic topology using real Internet data

The current topology generation algorithm in GT-ITM does not use real Internet data. Instead it uses user input to decide number of nodes and for placing edges between router nodes. User can change the input parameters to generate a topology with different densities and attributes. But if we can generate a topology using the real Internet data as the input, then that topology might be a better approximation of the Internet topology. In this section we describe the generation of GT-ITM domain-level topology using real Internet data.

Skitter[15], developed by CAIDA (Co-operative association for Internet Data Analysis), is a tool for actively probing the Internet in order to analyze topology and performance. Skitter consists of a set of monitors which are continuously sending ICMP (traceroute) requests to multiple destinations on the Internet. The traceroute information gathered at each monitor is aggregated to generate Internet IP route information. The IP routes are then mapped to their corresponding AS using the Border Gateway Protocol (BGP) routing tables collected by the University of Oregon's RouteViews project. Thus we can generate AS path data of the Internet, which can be effectively used to generate the domain-level GT-ITM topology.

We first generate an AS graph from the AS path data using the Stanford Graph Base. Then we identify the transit domains and the stub domains in the AS graph. The leaf nodes in the graph are

identified as the stub domains and all other nodes become the transit domains.

Once we have generated the transit-stub graph of the Internet AS graph, this graph can be expanded to include nodes within the AS or transit and stub domains. Later this expanded graph can be treated the same as other GT-ITM topologies to generate routing tables. This latter feature (i.e. expanding the AS graph to include nodes) is not incorporated in the current implementation of GT-ITM and is a subject of further research. However, a tool to convert the skitter AS path data into a transit-stub SGB graph is provided.

2.5 Conclusion

In this chapter we discussed our solution for the routing and topology problem. We divide the GT-ITM topology into multiple tiers, which aided us in defining policies based on the provider-customer-peer model. Then we generate routing tables based on these policies. We also briefly discussed how we can achieve realistic domain interconnection by building the GT-ITM topology of the real Internet data acquired from skitter.

Chapter 3

Routing Lookup and GTITM Software

3.1 Introduction

In this chapter, first we explain the routing lookup process to find a path from a source to a destination and then we list the API used to access the routing tables and return the correct next hop. Towards the end of the chapter, we demonstrate the effective use of policies to return different paths for the same source-destination pair by giving some examples.

Our implementation provides a nexthop function, which returns the nexthop node along the path from a given source to a given destination. Because we save just one copy of the interdomain routing table for the whole domain, the routing lookup is a little bit complicated. We explain the working of routing lookup for the example topology shown in Figure 3.1.

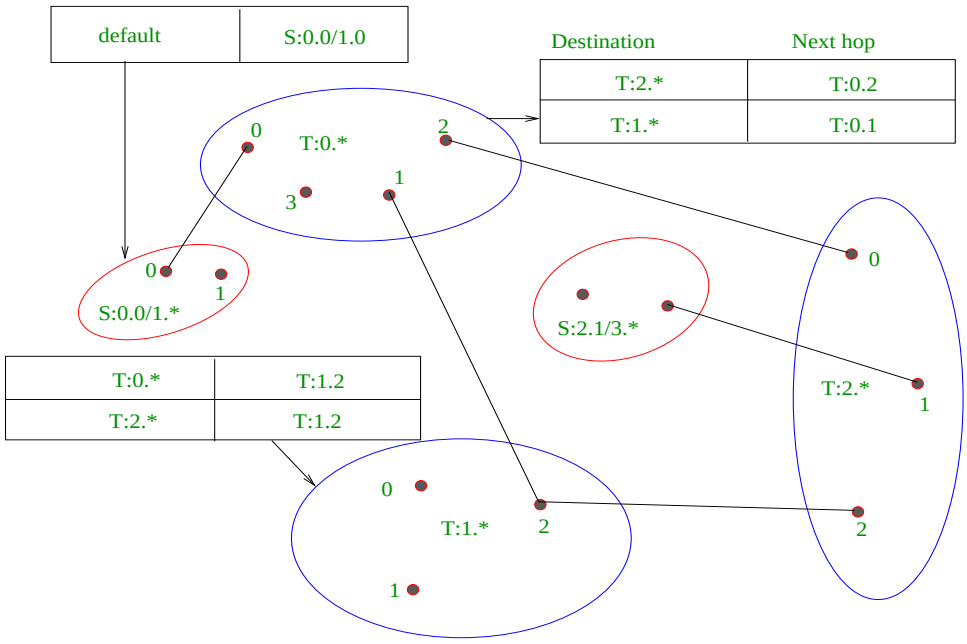


Figure 3.1: Sample routing topology

Suppose we want to find a route from node S:0.0/1.1 to S:2.1/3.2. The route can be traced by calling the nexthop function iteratively until the destination is reached[1]. As can be seen from the figure the source stub domain is single-homed and has just one link, to transit domain T:0.*. So there is just one entry in its interdomain routing table: the default route. When the nexthop function is called for the given source and destination, the first step is to find whether the destination node is in the same domain or a different domain. If the node is in the same domain then the intradomain table is consulted and we are done. There is just a single intradomain table for every domain which stores the index of the nexthop node from every source to every destination in that domain. This table can be read to figure out the path from a given source and destination within that domain. Otherwise (as in this case), the interdomain table is consulted. The interdomain table returns the address of the exit border router in the current domain as the next hop. The border router is temporarily made the destination and the next hop address is determined using the intradomain routing table. When the nexthop function is called at the border router, instead of returning the same node (as indicated by the interdomain routing table) the function returns the address of the transit domain node T:0.0 to which the border router is connected. This process of finding the correct neighboring border router is fairly simple in the case of a single-homed domain, because there is just one interdomain link. Now on calling the nexthop function for node T:0.0 and destination S:2.1/3.2 the nexthop function fails to find a match for S:2.1/3.2 in the interdomain table. So it looks for T:2.* (knowing that stub domains matching S:2.1/* are connected to node 1 in transit domain 2) and discovers T:0.2 as next hop. The route to T:0.2 is determined by consulting the intradomain routing table and calling the *nexthop* function at T:0.2, route to T:2.0 is determined. On calling the nexthop function at node T:2.0, again no entry for S:2.1/3.2 is found in the interdomain table; however, the nexthop function recognizes that the destination is connected to a node in the same transit domain, and so looks up T:2.1 in the intradomain table. At T:2.1 the nexthop function returns the stub node in the domain S:2.1/3.*; once inside that domain the intradomain table is used for the rest of the route. Finally, we explain the case mentioned earlier in the context of Figure 3.1, where a border transit node has neighbors in two different transit domains. Consider a case where the nexthop function is asked for the next hop for a destination in transit domain T:0.* from node T:1.2. The border router T:1.2 has two outgoing links: one to T:0.* and one to T:2.*. To find the correct nexthop, the AS path of the route is consulted. The AS path for the route to destination T:0.* is [0 2] which means that the route

[1]On calling the nexthop function with a certain source and destination, the function returns the next hop node on the path from the source to the destination. This nexthop function is called again with the same destination but now the source as the node returned from the previous call to nexthop. This process is continued till the nexthop function returns the destination node itself, which confirms that complete path from source to the destination has been traced.

traverses domain 2 and domain 0. (Assume that 0, 1 and 2 are the ASNs of domains T:0.*, T:1.* and T:2.*.) This means that the next hop for destination T:2.* is in domain with ASN 2. In this way, the correct next hop for border routers can be determined from the AS path.

3.2 API

The routing support being described will be available as a separate library along with GT-ITM. Here we briefly describe the API to be exported by this library. Each function returns a non zero value to indicate successful operation else it returns 0 or NULL to indicate failure.

- **int itmrt_generate_routing_tables(Graph *g):** Given a GT-ITM graph, this function runs the intradomain and the interdomain routing algorithms to generate the interdomain and intradomain routing tables.

- **int itmrt_free_routing_tables(Graph *g):** Given a GT-ITM graph, this function frees the memory allocated for both the routing tables. The call is generally made before calling gb recycle.

- **int itmrt_read_tables_from_file(Graph *g, char *rt file name):** Given a GT-ITM graph and the name of the .rt routing file, the function reads the routing tables from the file into the memory.

- **int itmrt_write_tables_to_file(Graph *g, char *sgb file name):** Given a GT-ITM graph and the sgb filename, the function writes the routing tables to a file with the same name but with extension rt instead of gb.

- **Vertex *next_hop(Graph *g, Vertex *source, Vertex *destination):** This is the main call which gives access to routing tables. The source and destination are vertex pointers in the Graph g. This call returns the Vertex pointer of the nexthop node towards the destination. If lookup fails it returns NULL indicating the reason for lookup failure to standard output. We need to call itmrt_read_tables_from_file or itmrt_generate_routing_tables before calling this function.

3.3 Using the tool

The routing information generation feature in GT-ITM is made of two important programs *policytool* and *genrtb*. genrtb invoked the API routings described above to create routing table information and save it to a file.

3.3.1 Generation of policy file

policytool is used to generate a policy file. The input to the program is the .gb file.

policytool .gbfile

A sample run of the tool is shown below.

```
-----------------------------------------------------------------
ash:~/gt-itm/bin> policytool ts300504-0.gb
Number of transit domains 25
Enter the desired size of the core: 7
Policies generated in file ts300504-0-7.po
Tier info stored in file ts300504-0.tr
ash:~/gt-itm/bin> more ts300504-0.tr
Total tiers 2
Core: 4 7 9 11 12 18 20
Tier 1: 0 1 2 3 5 6 8 10 13 14 15 16 17 19 21 22 23 24
-----------------------------------------------------------------
```

3.3.2 Generation of routing tables

Once the policy file is generated we can use the genrtb program to generate the routing files. The genrtb program takes the .gb file and the .po policy file as an input. The policy file input to the genrtb program is optional.

genrtb -g .gbfile -p .pofile

The policy file is either written by the user or it is automatically generated by the policytool program. A sample run of the genrtb program is shown below.

```
-----------------------------------------------------------------
ash:~/gt-itm/bin> genrtb -g ts300504-0.gb -p ts300504-0-7.po
```

```
Generating the intradomain routing tables
number of domains = 15025
no. of multi-homed domains = 233
Reading the policy file ts300504-0-7.po...
Generating the interdomain routing tables
0 percent completed
1 percent completed
    |
99 percent completed
100 percent completed
Adding the default routes to the single-homed domains
Writing the routing tables to the .rt file
Routing tables generated and stored in ts300504-0-7.rt
-----------------------------------------------------------------
```

3.3.3 Traceroute from source to destination

The program *gtitmtr* traces a route from a given source to a given destination. The program internally makes the above described API calls to read the graph and calls the next hop function to compute the path from the source to the destination.

gtitmtr .gbfile .rtfile start node index end node index

A sample run of this program is shown below.

```
-----------------------------------------------------------------
ash:~/gt-itm/bin> gtitmtr ts300504-0.gb ts300504-0-7.rt 10000 20000
Source = S:0.16/25.16 Destination = S:1.10/4.16
Intradomain routing table loaded
Interdomain table loaded Routing tables loaded in memory

1: S:0.16/25.6
2: S:0.16/25.1
3: S:0.16/25.7
4: S:0.16/25.14
5: S:0.16/25.5
```

```
6: S:0.16/25.8
7: T:0.16
8: T:0.1
9: T:0.11
10: T:20.14
11: T:20.9
12: T:1.19
13: T:1.8
14: T:1.10
15: S:1.10/4.18
16: S:1.10/4.1
17: S:1.10/4.21
18: S:1.10/4.16

Lookup time: 2226 microseconds
ash:~/gt-itm/bin>
----------------------------------------------------------------------
```

3.4 Effect of Policies

3.4.1 Effect of import policies

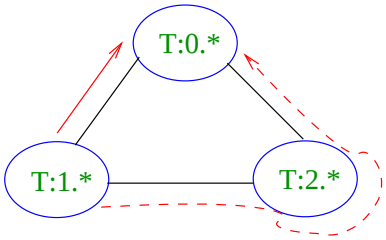
Import policies are mainly defined using the local preference attribute. To see the result of import policies, we will create a simple topology consisting of 3 transit domains, fully connected to each other as shown in Figure 4.1.

Then we manually write a policy file which defines an import policy for T:1.* stating:

(Assign a local preference of 100 for any destination route received from T:2.*.) Since the default local preference is 80, for destination T:0.*, T:1.* should prefer a longer route through T:2.*, instead going though the direct link to T:0.* as the former has a higher local preference. Below we enumerate the path from source to destination, with and without the defined import policy.

Path from source to destination without policies:

```
----------------------------------------------------------------------
Source = S:1.0/2.2 Destination = S:0.0/1.1
```



Which route should T:1.* take ? What do the policies say ?

Figure 3.2: Import policies example

```
Intradomain routing table loaded
Interdomain table loaded Routing tables loaded in memory
1: T:1.0
2: T:1.1
3: T:0.1
4: T:0.2
5: T:0.0
6: S:0.0/1.1
Lookup time: 134 microseconds
-----------------------------------------------------------------------
```

Path from source to destination with the import policy:

```
-----------------------------------------------------------------------
Source = S:1.0/2.2 Destination = S:0.0/1.1
Intradomain routing table loaded
Interdomain table loaded
Routing tables loaded in memory
1: T:1.0
2: T:1.1
3: T:2.0
4: T:0.1
5: T:0.2
6: T:0.0
```

```
7: S:0.0/1.1
Lookup time: 150 microseconds
-----------------------------------------------------------------
```

We see from above, that when the policy file is used in creating the routing tables, a longer path is taken to the destination.

3.4.2 Effect of export policies

Each node in the domain level graph has been tagged by a tier number, which enables us to identify a particular node as either a customer, provider or a peer of its connected neighbor. In the tool described in the previous section, our automatically generated export policies make sure that a domain does not export routes learnt from its peer and provider to its other peers and providers. We now present an example demonstrating how export policies affect the route taken.

Consider a GT-ITM topology with 300504 nodes, 25 transit domains and 233 multi-homed domains (transit + multi-homed stubs). To see the effect of policies we will show the traceroute from a source to destination with and without policies. Firstly to use policies, we run the *policytool* program on the .gb file, to generate a policy file for use. The tier structure of the transit level topology is shown below.

```
-----------------------------------------------------------------
Total tiers 2
Core: 4 7 9 11 12 18 20
Tier 1: 0 1 2 3 5 6 8 10 13 14 15 16 17 19 21 22 23 24
-----------------------------------------------------------------
```

Then we generate two routing tables, one with this policy file as an input to the *genrtb* program and one without. Below is a traceroute from a source to a destination without the use of policies.

```
-----------------------------------------------------------------
Source = S:2.6/22.12 Destination = S:6.11/9.1
Intradomain routing table loaded
Interdomain table loaded Routing tables loaded in memory
1: S:2.6/22.17
2: S:2.6/22.13
```

```
3: S:2.6/22.1
4: S:2.6/22.9
5: T:2.6
6: T:2.8
7: T:2.16
8: T:23.17
9: T:23.11
10: T:23.3
11: T:6.11
12: S:6.11/9.9
13: S:6.11/9.19
14: S:6.11/9.1
Lookup time: 1189 microseconds
-----------------------------------------------------------------
```

As we see in the above traceroute, the path follows transit domain $T:2.*--T:23.*--T:6.*$ and the hop count is 14. Looking at tiers information shown above, T:2.*, T:23.* and T:6.* belong to the same tier. If policies were used, such a behavior is not acceptable, as a pier is not supposed to exchange routes learnt from its peers to other peers. Below is a traceroute for the same source and destination, but with policies imposed to decide the route.

```
-----------------------------------------------------------------
Source = S:2.6/22.12 Destination = S:6.11/9.1
Intradomain routing table loaded
Interdomain table loaded
Routing tables loaded in memory
1: S:2.6/22.17
2: S:2.6/22.13
3: S:2.6/22.1
4: S:2.6/22.9
5: T:2.6
6: T:2.2
7: T:2.1
8: T:18.17
```

```
9: T:18.2
10: T:18.7
11: T:6.10
12: T:6.5
13: T:6.11
14: S:6.11/9.9
15: S:6.11/9.19
16: S:6.11/9.1
Lookup time: 1504 microseconds
--------------------------------------------------------------------
```

In this route the path followed is T:2.* – T:18.* – T:6.* and the hop count is 16. Looking at the tiers information shown above, T:18.* belongs to a higher tier than T:2.* and T:6.* making it a provider to both T:2.* and T:6.*. As we see here, though the hop count has increased to 16 (as compared to 14 without the use of policies), the path taken obeys the export policies as shown in Figure 4.2.

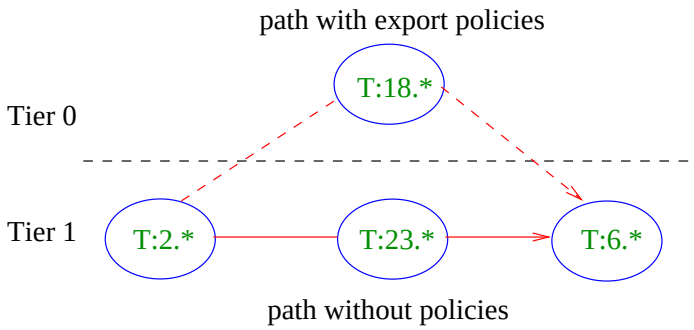


Figure 3.3: Export policies example

3.4.3 Effect of different policies on the same graph

In this section we demonstrate the use of different policy files with the same graph to generate different routing tables. We will see that these two routing tables result in different paths for the same pair of source and destination. We vary the core size in order to generate two different policy files for the same graph.

For this example we select a graph of 80800 nodes. First we generate a policy file based on a

core size of 5. Observe that the total number of tiers in this case is 3 (Core, Tier 1 and Stub domains)

```
ash:~/gt-itm/freshcopy/bin> policytool ts80800-0.gb

Number of transit domains 20
Enter the desired size of the core: 5
Policies generated in file ts80800-0-5.po
Tier info stored in file ts80800-0.tr

ash:~/gt-itm/freshcopy/bin> more ts80800-0.tr

Total tiers 2
Core: 5 6 13 16 17
Tier 1: 0 1 2 3 4 7 8 9 10 11 12 14 15 18 19
ash:~/gt-itm/freshcopy/bin>
```

Then we generate a policy file based on a core size of 2. Observe that the total number of tiers in this case is 4 (Core, Tier1, Tier2 and Stub domains)

```
ash:~/gt-itm/freshcopy/bin> policytool ts80800-0.gb
Number of transit domains 20
Enter the desired size of the core: 2
Policies generated in file ts80800-0-2.po
Tier info stored in file ts80800-0.tr
ash:~/gt-itm/freshcopy/bin> more ts80800-0.tr
Total tiers 3
Core: 11 14
Tier 1: 0 1 2 5 6 8 9 10 13 16 18 19
Tier 2: 3 4 7 12 15 17
ash:~/gt-itm/freshcopy/bin>
```

For each policy file generated, we use the genrtb program to generate routing tables. The routing tables are stored by the name ts80800-0-5.rt and ts80800-0-2.rt respectively. Below is the traceroute for some source and destination by using ts80800-0-5.rt as the routing table file.

```
ash:~/gt-itm/freshcopy/bin> gtitmtr ts80800-0.gb ts80800-0-5.rt 10 20000
Graph Restored
Source = T:0.10 Destination = T:5.2
Intradomain routing table loaded
Interdomain table loaded
Routing tables loaded in memory
1: T:0.0
2: T:0.19
3: T:13.19
4: T:13.20
5: T:5.28
6: T:5.2
Lookup time: 558 microseconds
```

In the above trace we see that the route passes through transit domain 13 to reach transit domain 5. Below is a traceroute for the same source and destination but using ts80800-0-2.rt as the routing table file.

```
ash:~/gt-itm/freshcopy/bin> gtitmtr ts80800-0.gb ts80800-0-2.rt 10 20000
Graph Restored
Source = T:0.10 Destination = T:5.2
Intradomain routing table loaded
Interdomain table loaded
Routing tables loaded in memory
1: T:0.2
2: T:14.22
3: T:14.27
4: T:11.27
5: T:11.0
6: T:11.31
7: T:5.34
8: T:5.0
9: T:5.2
Lookup time: 848 microseconds
```

In the above trace we see that the route passes through transit domain 14 and 11 to reach transit domain 5.

3.5 Conclusion

In this chapter, we showed the effective use of policies to generate different paths from a source to a destination. By changing the core size, we can manipulate the hierarchy in the graph. This in turn results in different policy files for the same graph. Each policy file when used with a given GT-ITM graph, will generate different set of routing tables and each routing table may result in different paths for the same source-destination pair. This feature of GT-ITM can be effectively used to simulate Internet routing, by dynamically changing the routing files to get different paths at different times. In the next chapter we show some results we obtain by running some experiments of different graphs.

Chapter 4

GT-ITM Software

4.1 Introduction

In this chapter, first we list the API used to access the routing tables and return the correct next hop. In the previous chapter, we discuss the routing tables lookup algorithm. Towards the end of the chapter, we demonstrate the effective use of policies to return different paths for the same source-destination pair by giving some examples.

4.2 API

The routing support being described will be available as a separate library along with GT-ITM. Here we briefly describe the API to be exported by this library.

- **int itmrt_generate_routing_tables(Graph *g):** Given a GT-ITM graph this function runs the intradomain and the interdomain routing algorithms to generate the interdomain domain and intradomain routing tables.
- **int itmrt_free_routing_tables(Graph *g):** Given a GT-ITM graph this function frees the memory allocated for both the routing tables. The call is generally made before calling gb recycle.
- **int itmrt_read_tables_from_file(Graph *g, char *rt file name):** Given a GT-ITM graph and the name of the .rt routing file the function reads the routing tables from the file into the memory. This is an alternative to the generate API call as it is faster.
- **int itmrt_write_tables_to_file(Graph *g, char *sgb file name):** Given a GT-ITM graph and the sgb file name the function writes the routing tables to a file with the same sgb filename but with extension rt instead of gb. This call can be made only if generate call is successful.
- **Vertex *next_hop(Graph *g, Vertex *source, Vertex *destination):** This is the main call which gives access to routing tables. The source and destination are vertex pointers in the

Graph g. This call returns the Vertex pointer of the nexthop node towards the destination. If lookup fails it returns NULL indicating the reason for lookup failure to standard output. We need to call itmrt_read_tables_from_file before calling this function.

4.3 Using the tool

The routing information generation feature in GT-ITM is made of two important programs *policytool* and *genrtb*.

4.3.1 Generation of policy file

policytool is used to generate a policy file. The input to the program is the .gb file.

policytool .gbfile

A sample run of the tool is shown below.

————————————————————————- ash: /gt-itm/bin¿ policytool ts300504-0.gb Number of transit domains 25 Enter the desired size of the core: 7 Policies generated in filets300504-0-7.po Tier info stored in file ts300504-0.tr ash: /gt-itm/bin¿ more ts300504-0.tr Total tiers 2 Core: 4 7 9 11 12 18 20 Tier 1: 0 1 2 3 5 6 8 10 13 14 15 16 17 19 21 22 23 24 ————————————————————————————-

4.3.2 Generation of routing tables

Once the policy file is generated we can use the genrtb program to generate the routing files. The genrtb program takes the .gb file and the .po policy file as an input. The policy file input to the genrtb program is optional.

genrtb -g .gbfile -p .pofile

The policy file is either written by the user or it is automatically generated by the policytool program. A sample run of the genrtb program is shown below.

```
------------------------------------------------------------------------
ash:~/gt-itm/bin> genrtb -g ts300504-0.gb -p ts300504-0-7.po
Generating the intradomain routing tables
number of domains = 15025
no. of multi-homed domains = 233
Reading the policy filets300504-0-7.po...
```

```
Generating the interdomain routing tables
0 percent over
1 percent over
    |
99 percent over 100 percent over Adding the default routes to the
single-homed domains Writing the routing tables to the .rt file
Routing tables generated and stored in ts300504-0-7.rt Finishing
the final tasks 66545 66545
-----------------------------------------------------------------
```

4.3.3 Traceroute from source to destination

The program *gtitmtr* traces a route from a given source to a given destination. The program internally makes the above mentioned API calls to read the graph and calls the next hop function to compute the path from the source to the destination.

gtitmtr .gbfile .rtfile start node index end node index

A sample run of this program is shown below.

```
-----------------------------------------------------------------
ash:~/gt-itm/bin> gtitmtr ts300504-0.gb ts300504-0-7.rt 10000 20000
Source = S:0.16/25.16 Destination = S:1.10/4.16
Intradomain routing table loaded
Interdomain table loaded
Routing tables loaded in memory

1: S:0.16/25.6
2: S:0.16/25.1
3: S:0.16/25.7
4: S:0.16/25.14
5: S:0.16/25.5
6: S:0.16/25.8
7: T:0.16
8: T:0.1
9: T:0.11
```

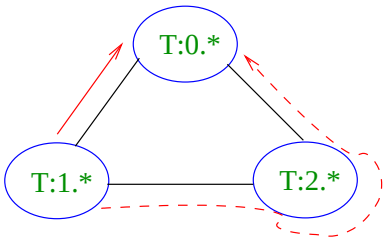
```
10: T:20.14
11: T:20.9
12: T:1.19
13: T:1.8
14: T:1.10
15: S:1.10/4.18
16: S:1.10/4.1
17: S:1.10/4.21
18: S:1.10/4.16

Lookup time: 2226 microseconds
ash:~/gt-itm/bin>
-----------------------------------------------------------------------
```

4.4 Effect of Policies

4.4.1 Effect of import policies

Import policies are mainly defined using the local preference attribute. To see the result of import policies, we will create a simple topology consisting of 3 transit domains, fully connected to each other as shown in Figure 4.1.



Which route should T:1.* take ? What do the policies say ?

Figure 4.1: Import policies example

Then we manually write a policy file which defines a import policy for T:1.* stating: Assign a local preference of 100 for any destination route received from T:2.*. Since the default local pref-

erence is 80, for destination T:0.*, T:1.* should prefer a longer route through T:2.*, instead going though the direct link to T:0.* as the former has a higher local preference. Below we enumerate the path from source to destination, with and without the defined import policy.

Path from source to destination without policies.

```
------------------------------------------------------------------
Source = S:1.0/2.2 Destination = S:0.0/1.1
Intradomain routing table loaded
Interdomain table loaded
Routing tables loaded in memory
1: T:1.0
2: T:1.1
3: T:0.1
4: T:0.2
5: T:0.0
6: S:0.0/1.1
Lookup time: 134 microseconds
------------------------------------------------------------------
```

Path from source to destination with the import policy.

```
------------------------------------------------------------------
Source = S:1.0/2.2 Destination = S:0.0/1.1
Intradomain routing table loaded
Interdomain table loaded
Routing tables loaded in memory
1: T:1.0
2: T:1.1
3: T:2.0
4: T:0.1
5: T:0.2
6: T:0.0
7: S:0.0/1.1
Lookup time: 150 microseconds
------------------------------------------------------------------
```

We see from above, that when the policy file is used, a longer path is taken to the destination.

4.4.2 Effect of export policies

Each node in the domain level graph has been tagged by a tier number, which enables us to identify a particular node as either a customer, provider or a peer of its connected neighbor. In the tool described in the previous section, our automatically generated export policies make sure that a domain does not export routes learnt from its peer and provider to its other peers and providers. Giving a suitable example we will demonstrate how the export policies affect the route taken.

Consider a GT-ITM topology with 300504 nodes, 25 transit domains, 233 multi-homed domains (transit + multi-homed stubs). To see the effect of policies we will show the traceroute from a source to destination with and without policies. Firstly to use policies, we run the *policytool* program on the .gb file, to generate a policy file for use. The tier structure of the transit level topology is shown below.

```
----------------------------------------------------------------
Total tiers 2
Core: 4 7 9 11 12 18 20
Tier 1: 0 1 2 3 5 6 8 10 13 14 15 16 17 19 21 22 23 24
----------------------------------------------------------------
```

Then we generate two routing tables, once with this policy file as an input to the *genrtb* program and once without. Below is a traceroute from a source to the destination without the use of policies.

```
----------------------------------------------------------------
Source = S:2.6/22.12 Destination = S:6.11/9.1
Intradomain routing table loaded
Interdomain table loaded
Routing tables loaded in memory
1: S:2.6/22.17
2: S:2.6/22.13
3: S:2.6/22.1
4: S:2.6/22.9
5: T:2.6
6: T:2.8
```

```
7: T:2.16
8: T:23.17
9: T:23.11
10: T:23.3
11: T:6.11
12: S:6.11/9.9
13: S:6.11/9.19
14: S:6.11/9.1
Lookup time: 1189 microseconds
-----------------------------------------------------------------
```

As we see in the above traceroute, the path follows transit domain $T:2.* - - T:23.* - - T:6.*$ and the hop count is 14. Looking at tiers information shown above, T:2.*, T:23.* and T:6.* belong to the same tier. If policies were used, such a behavior is not acceptable, as a pier is not supposed to exchange routes learnt from its peers to other peers. Below is a traceroute for the same source and destination, but with policies imposed to decide the route.

```
-----------------------------------------------------------------
Source = S:2.6/22.12 Destination = S:6.11/9.1
Intradomain routing table loaded
Interdomain table loaded
Routing tables loaded in memory
1: S:2.6/22.17
2: S:2.6/22.13
3: S:2.6/22.1
4: S:2.6/22.9
5: T:2.6
6: T:2.2
7: T:2.1
8: T:18.17
9: T:18.2
10: T:18.7
11: T:6.10
12: T:6.5
```

```
13: T:6.11
14: S:6.11/9.9
15: S:6.11/9.19
16: S:6.11/9.1
Lookup time: 1504 microseconds
--------------------------------------------------------------------
```

In this route the path followed is T:2.* – T:18.* – T:6.* and the hop count is 16. Looking at the tiers information shown above, T:18.* belongs to a higher tier than T:2.* and T:6.* making it a provider to both T:2.* and T:6.*. As we see here, though the hop count has increased to 16 (as compared to 14 without the use of policies), the path taken obeys the export policies as shown in Figure 4.2.

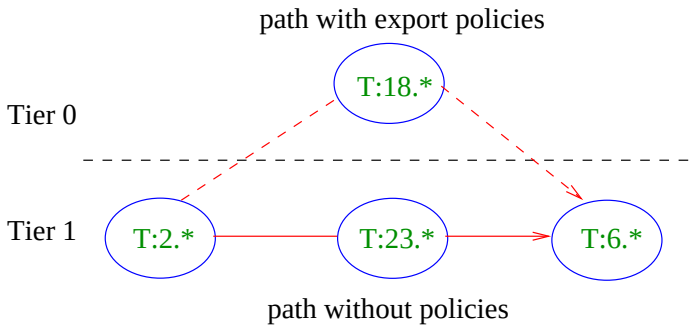


Figure 4.2: Export policies example

4.5 Conclusion

In this chapter, we showed the effective use of policies to generate different paths from a source to a destination. By changing the core size, we can manipulate the hierarchy in the graph. This in turn results in different policy files for the same graph. Each policy file when used with a given GT-ITM graph, will generate different set of routing tables and each routing table may result in different paths for the same source-destination pair. This feature of GT-ITM can be effectively used to simulate Internet routing, by dynamically changing the routing files to get different paths at different times. In the next chapter we show some results we obtain by running some experiments of different graphs.

Chapter 5

Simulation graphs

In the first chapter, we discussed how the traditional approaches to generating routing paths were ineffective, as they demanded more memory space and computational time. Hence we proposed a solution which is less complex in space and time, and which also gives better control over generation of routing tables. In this chapter we will study the performance of the tool in terms of the time it takes to generating intradomain and interdomain routing tables and memory space required to store the routing tables.

The time complexity of generating the intradomain routing tables is a function of the number of domains in the graph and the number of nodes in each domain, whereas the time taken for the generation of interdomain routing tables depends on the number of multi-homed domains and the number of edges between the multi-homed domains in the graph. Since transit domains are essentially multi-homed, increasing the number of transit domains automatically increases the time taken to generate interdomain routing tables. Hence we take different measurements by varying the number of nodes, the number of multi-homed domains, and the number of transit domains, and study the time taken for routing table generation and its space utilization for storing the routing tables.

Based on the parameter (number of nodes, number of domains, number of multi-homed domains) we are changing, we generate different graphs by changing that parameter. For each graph, we generate the routing tables using *genrtb*. We record the total time taken for the generation of the routing tables. Once the routing tables are stored in a file, the file size is recorded for different graphs.

Below we show the graphs, which plot the results for our different set of experimental runs. These simulation runs were made on a 2 GHz machine with 1GB RAM.

On an average the lookup time for a single hop is 100 microseconds, which means that if a route has 10 hops from a source to a destination, it would take the route around 1ms. This time excludes the time taken for reading the graph into the memory.

In the graphs 5.1 and 5.2 the number of transit domains was kept constant at 25, and number of nodes in the transit domains was varied to increase the total number of nodes in the graph.

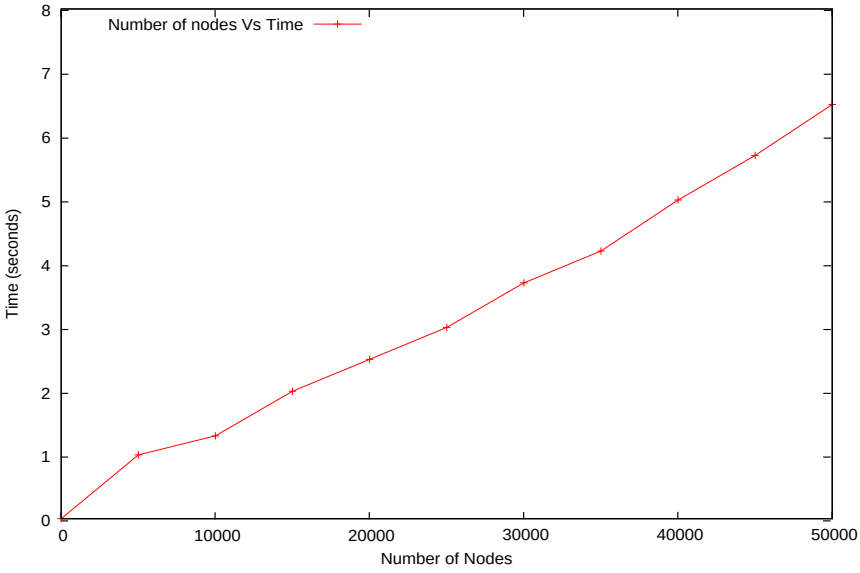


Figure 5.1: Number of nodes Vs Time

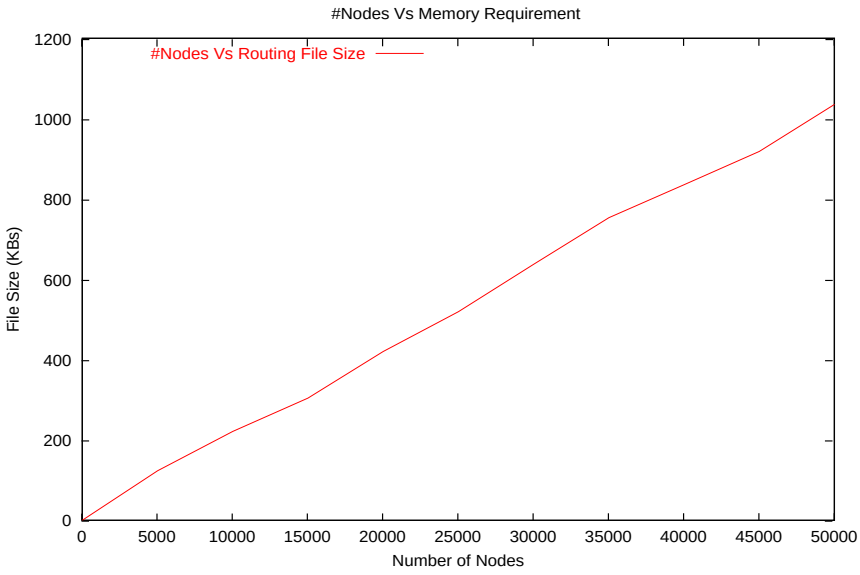


Figure 5.2: Number of nodes Vs Size

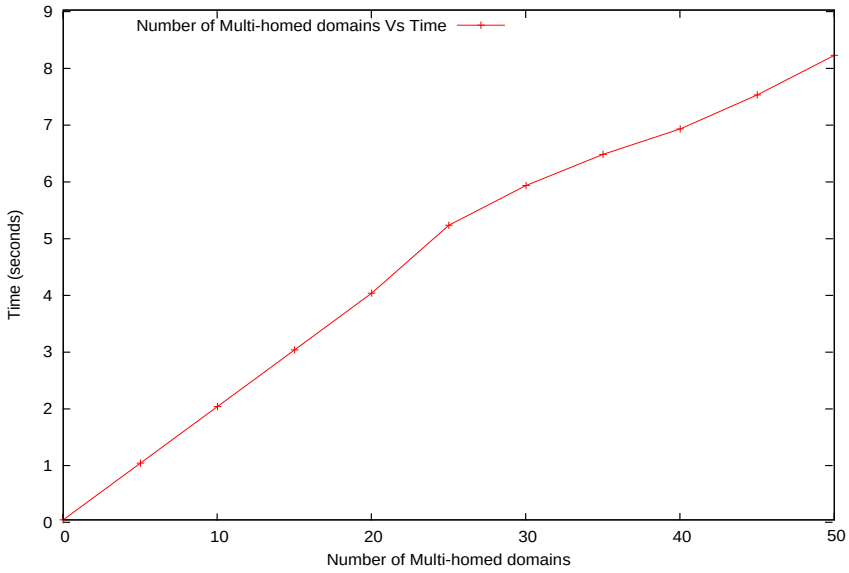


Figure 5.3: Number of multi-homed domains Vs Time

In the graph 5.3 the number of multi-homed domains was varied but by keeping the number of transit domains constant. This was done by adding extra transit-stub and stub-stub edges thus increasing the number of multi-homed stub domains.

In the graph 5.4 and 5.5, the number of transit domains were varied.

In graphs 5.1 and 5.2 we increase the nodes in the transit domains but keep the number of transit domains and stub domains constant in order to increase the total number of nodes within the graph. This keeps the size of the interdomain graph (graph constructed to run the BGP-like interdomain algorithm to find the interdomain routes) constant, but increases the matrix size for computing the intradomain routing tables. Let us assume that we have t transit domains, s stub domains per transit node, x nodes per transit domains and y nodes per stub domain, the time complexity of computing the intradomain routing tables would be O($tx^3 + txsy^3$). The total number of nodes in the graph would be $tx(1 + sy)$. Let us assume the time taken to generate the interdomain routing tables as I_t. So the total time to generate the routing tables would be O($tx^3 + txsy^3$)) + I_t. When we increase x by keeping all other parameters constant, we observe from graph 5.1 that we get a linear increase in time when referenced against the total number of nodes in the graph.

Let us analyze the effect of increasing the number of nodes on the space requirements. As mentioned in the previous chapters, the routing table file consists of intradomain routing tables

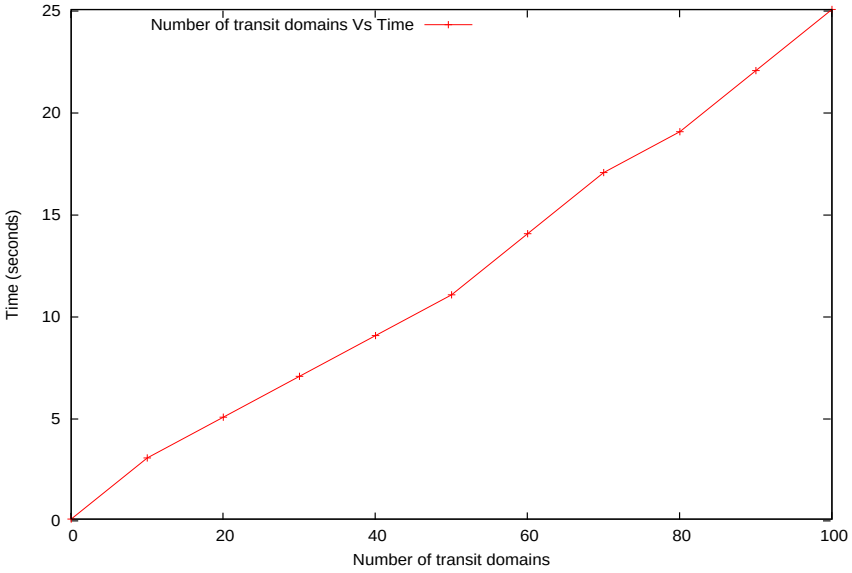


Figure 5.4: Number of transit domains Vs Time

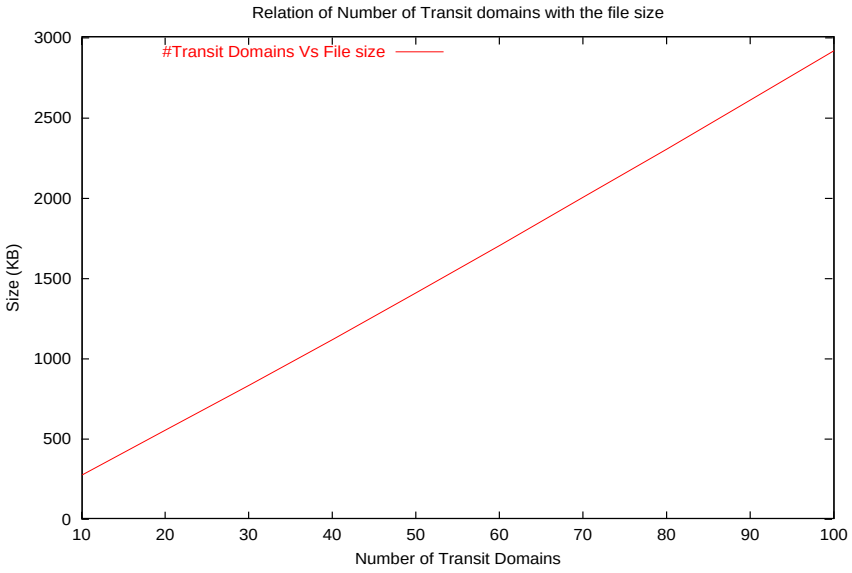


Figure 5.5: Number of transit domains Vs Space

followed by the interdomain routing tables. Intradomain routing tables consist of two dimensional matrices for each domain (transit and stub) giving all pairs shortest paths between the nodes of that domain. In the interdomain routing tables, for each multi-homed domain we list a routing path to every other multi-homed domain. Say for example we have m multi-homed domains, each multi-homed domain would have a path to $m - 1$ multi-homed domain. So the size of interdomain routing tables is a function of the number of multi-homed domains in the graph whereas the size of the intradomain routing tables is a function of the number of nodes within the transit domains and stub domains. In the graph 5.2 we observe that the size of the routing tables file increases linearly with the increase in the transit domain nodes. In this case the size of the intradomain tables increases and the size of the interdomain tables remains constant. Important point to note here is that increasing the number of nodes within the transit domains and the stub domains has no effect on the interdomain routing table generation time nor the space to store the interdomain tables. I_t remains constant in both the case mentioned above as the number of multi-homed domains remain constant.

In graph 5.4, we increase the graph size by increasing the number of transit domains in the graph. There are two ways of increasing the multi-homed domains. Either we increase the number of transit domains or we add extra transit-stub or stub-stub edges which increases the number of multi-homed stub domains. Increasing the number of multi-homed domains has effect both on the intradomain tables generation time and the interdomain tables generation time. Looking at the equation O($tx^3 + txsy^3$), when we increase the number of transit domains t the time taken to generate intradomain routing tables increases as we have to run Floyd Warshal on more domains. Similarly increasing the transit domains increases the time taken to generate interdomain routing tables as we have to compute more routes for each multi-homed domain.

In graph 5.3, we increase the number of transit-stub and stub-stub edges to increase the number of multi-homed domains. This does not have any effect on the intradomain time as the number of domains remains constant. But the time taken to compute interdomain routing tables does increase. Due to this reason we see that as compared to graph 5.4 the slope of the graph 5.3 is less as intradomain computation time is not affected in the later case. In the last graph 5.5, we observe that increasing the transit domains has a linear increase in routing file size. Increase in transit domains increases the number of entries in the interdomain routing table hence the increase in routing file size.

The objective of the thesis was to enhance the current GT-ITM software by providing routing support so that route from any node to any other node can be determined without the user having to

provide his own implementation. The idea was to use a routing algorithm which is time-efficient, space efficient (uses less memory) and also generates routing tables which occupy less space and generates routes similar in nature to those found on the Internet. A primitive implementation was to use Floyd Warshal algorithm which was not efficient both in terms of time and space for graphs with large number of nodes. So we decided to break the routing problem into intradomain and interdomain and treat them separately. Finding intradomain routes was done using Floyd Warshal as the number of nodes within a domain is much lesser as compared to the total number of nodes. This way we achieved some gain both in terms of time and space. For generating interdomain routing tables, we implemented a BGP-like protocol which takes into account domain level policies and gives some control to the user to control the routes. Looking at some examples in the previous chapter we see that by controlling the policies we could change the route taken between two nodes. Looking at the graphs above we see that the time taken to generate routing tables and the space required to store the routing tables is a linear function of the number of nodes in the graph. Thus our solution satisfies all the objectives defined in the thesis.

Bibliography

[1] K. L. Calvert, M. B. Doar, and E. W. Zegura, “Modeling Internet Topology”, *IEEE Communications Magazine*, 35(6), June 1997, pp. 160–163.

[2] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee, “How to Model an Internetwork”, *Proceedings of IEEE INFOCOM ’96*, San Franscisco, USA, March 1996, pp. 594–602.

[3] A. Medina, A. Lakhina, I. Matta and J. Byers, “BRITE: An Approach to Universal Topology Generation”, in Proceedings of MASCOTS ’01, August 2001.

[4] C. Jin, Q. Chen, and S. Jamin, ”Inet topology generator,” Technical Report CSE-TR-433-00, EECS Department, University of Michigan, 2000.

[5] D. E. Knuth, *The Stanford GraphBase: A Platform for Combinatorial Computing*, Addison-Wesley, 1993.

[6] Hongsuda Tangmunarunkit, Ramesh Govindan, Sugih Jamin, Scott Shenker, and Walter Willinger, ”Network topology generators: Degree-based vs structural”, *ACM SIGCOMM, 2002*

[7] M. Faloutsos, P. Faloutsos, and C. Faloutsos, ”On power-law relationships of the internet topology”, *in Proc. SigComm. ACM, 1999*

[8] T. Griffin and G. Wilfong, “An Analysis of BGP Convergence Properties,” *Proceedings of ACM SIGCOMM 1999 Symposium*, Cambridge, Massachusetts, August 1999, pp. 277–288.

[9] L. Subramanian, S. Agarwal, J. Rexford, and R. Katz, “Characterizing the Internet Hierarchy from Multiple Vantage Points,” *Proceedings of IEEE INFOCOM 2002*, New York, USA, June 2002.

[10] L. Gao and J. Rexford, ”Stable Internet routing without global coordination,” in *Proc. ACM SIGMETRICS*, June 2000.

[11] Policy Disputes in Path-Vector Protocols (1999) Timothy G. Griffin F. Bruce Shepherd Gordon Wilfong griffin

[12] Real-time Model and Convergence Time of BGP (2002) Davor Obradovic

[13] Improving BGP Convergence Through Consistency Assertions (2002) Dan Pei, Xiaoliang Zhao, Lan Wang, Dan Massey, Allison Mankin, S. Felix Wu, Lix ia Zhang

[14] Internet Routing Instability (1997) Craig Labovitz, G. Robert Malan, and Farnam Jahanian

[15] Origins of Pathological Internet Routing Instability, Craig Labovitz, Rob Malan, and Farnam Jahanian

[16] Stable model symantics: http://www.tcs.hut.fi/Software/smodels/

[17] BGP Routeviews project. http://www.routeviews.org/

Vita

Date and Place of Birth:	Pune, India January 1980
Education:	Bachelors in Electronic Engineerting, special subject: Networking University of Pune, 2001
Professional Positions:	Graduate Research Assistant University of Kentucky Lexington, Kentucky, 2001-2003
Honors:	Research Assistantship University of Kentucky, 2001–2003

Professional Publications:

"Extending and Enhancing GT-ITM" In *Mometools SIGCOMM Workshop*, Germany, 2003.

(Aditya Namjoshi)