Reinforcing Reachable Routes

Muthukumar Thirunavukkarasu

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE
in
Computer Science and Applications

Dr. Srinidhi Varadarajan
Co-Chair

Dr. Naren Ramakrishnan
Co-Chair

Dr. Calvin J. Ribbens

May 4, 2004
Blacksburg, VA

Keywords: Multipath routing, Reinforcement learning, Reachability,
Probabilistic algorithms

Reinforcing Reachable Routes

Muthukumar Thirunavukkarasu

Abstract

Reachability routing is a newly emerging paradigm in networking, where the goal is to determine all paths between a sender and a receiver. It is becoming relevant with the changing dynamics of the Internet and the emergence of low-bandwidth wireless/ad hoc networks. This thesis presents the case for reinforcement learning (RL) as the framework of choice to realize reachability routing, within the confines of the current Internet backbone infrastructure. The setting of the reinforcement learning problem offers several advantages, including loop resolution, multi-path forwarding capability, cost-sensitive routing, and minimizing state overhead, while maintaining the incremental spirit of the current backbone routing algorithms. We present the design and implementation of a new reachability algorithm that uses a model-based approach to achieve cost-sensitive multi-path forwarding. Performance assessment of the algorithm in various troublesome topologies shows consistently superior performance over classical reinforcement learning algorithms. Evaluations of the algorithm based on different criteria on many types of randomly generated networks as well as realistic topologies are presented.

Acknowledgements

I would like to thank my advisors Dr. Naren Ramakrishnan and Dr. Srinidhi Varadarajan. Dr. Ramakrishnan: Thank you for being a constant source of inspiration and encouragement and for your ability to find the right ideas at the right time to overcome all the obstacles in my thesis. I am also indebted to you for spending several hours reviewing this document for accuracy and providing valuable feedback during the several iterations that this document went through. Dr. Varadarajan: Thank you for introducing me to this project and for your valuable suggestions during the development and evaluation of the algorithm. I would also like to thank you for providing me with the necessary resources and most importantly of all for providing me with the source code of your simulator without which none of this would have been possible. I would also like to thank Dr. Calvin Ribbens for agreeing to be in my thesis committee and for giving valuable suggestions on my thesis.

I would like to thank the members of CSRL (Joy Mukherjee, Chris Knestrick, Atul Shenoy, Joe Ruscio, Bharath Ramesh, Mike Heffner and Ashwin Raju) for giving me constant support and encouragement. I would also like to thank my roommates (Arvind Adhi and Sandeep Prabhakar) and all the people in the Torgersen 2160 research space (Saverio Perugini, Robert Capra, Dan Moisa, Maulik Shukla, Douglas Slotta and Padmapriya Kandhadai) for making my stay at Virginia Tech a mixed balance of work and fun. I especially would like to thank Bharath for helping me with the mathematical derivations, Gaurav and Ashwin for proof reading my thesis document, and Jon Bernard for setting up my machine.

I would also like to thank my family for their love and encouragement all through my life. Last but not the least, I would like to thank my trusted computer (bertram) for churning out one result after the other without failing and stuck with me till the end.

Table of Contents

List of Figures

List of Tables

Chapter 1. Introduction

With the continuing growth and dynamism of large-scale networks, the need for alternate routing algorithms is becoming increasingly important. The emergence of low-bandwidth adhoc mobile networks requires routing algorithms that can distribute data traffic across multiple paths and quickly adapt to changing conditions.

Routing protocols construct tables at each node that specify for each destination the next-hop to use for data packet forwarding. A minimal requirement is that the computed routing tables be free of loops when the network is stable. In dynamic environments, a more stringent requirement is that the routing tables be loop-free not only when the network is stable but at *every instant* since loops, even if temporary, can rapidly degrade performance.

The effectiveness of a routing protocol directly impacts both the end-to-end throughput and end-to-end delay. Current network routing protocols are primarily concerned with deriving shortest-cost routes between a source and destination, i.e. they are tailored towards *single-path routing*. Recently, there has been an increased emphasis on *multi-path routing*, where routers maintain multiple distinct paths of arbitrary costs between a source and destination.

Multi-path routing presents several advantages over single-path routing. First, a multi-path routing protocol is capable of meeting multiple performance objectives – maximizing throughput, minimizing delay, bounding delay variation, and minimizing packet loss. Second, from a scalability perspective, it makes effective use of graph structure of a network (as opposed to single-path routing which superimposes a logical routing tree upon the network topology). Third, multi-path routing protocols are more tolerant of network failures. Finally multi-path routing algorithms are less susceptible to route oscillations, which enables the use of high-variance cost metrics that are better congestion indicators, whereas current single-path routing algorithms face route oscillations since they switch routes as a step function.

While multi-path routing is a desirable goal, the current Internet routing framework cannot be easily extended to support it. One solution is to develop a new multi-path routing framework, which necessitates changes to the Internet's networking protocol (IP). The main problem here stems from deployability concerns. The approach in this thesis is to study multipath routing within the confines of the current Internet protocol, which leads to interesting design decisions.

Multi-path routing can be qualified by the state maintained at each router and the routing granularity. For instance, a routing algorithm can maintain multiple, distinct, shortest-cost routing tables, where each routing table is based on a different cost metric. We refer to this as *multi-metric*, multi-path routing approach. The second approach is to allow multiple network paths between a source and destination based on a single cost metric. This means that routers may use sub-optimal paths; wherein the router could send data on multiple paths to maximize network throughput. We refer to this as *single-metric*, multi-path routing approach.

Multi-path routing algorithms can also be distinguished by the routing granularity into *coarse grain*, connection-oriented or *fine-grain*, connectionless approaches. The former adopts a path-per-connection view wherein all packets belonging to the same connection follow the same path. However, different connections between the source and destination hosts may follow different paths. In contrast, connectionless networks have no mechanism to associate packets to any higher-level notion of a connection; hence they use the fine-grained approach. For true multi-path forwarding, the routing algorithm should forward packets between a source-destination pair along multiple paths, which may not be necessarily the shortest-cost paths. The focus of this thesis is on such *fine grain multi-path routing algorithms within a single-metric domain*.

1.1 Problem Statement

One way to achieve multi-path routing is to extend single-path routing protocols. This extension is non-trivial for two reasons. First, we need mechanisms to incorporate state corresponding to multiple paths into the routing table. More importantly, we need new loop avoidance algorithms: current shortest-path routing algorithms use their optimality metric to implicitly eliminate loops. This assumption is untenable for multi-path routing in a single-metric domain. Resolving these issues typically requires routers to maintain routing state proportional to the number of paths in the network.

In this thesis, we approach multi-path routing from the terminal perspective of *reachability routing*. The goal of reachability routing is to determine all paths between a sender and a receiver, without the above mentioned state or consistency maintenance overhead. While *basic reachability routing* is primarily concerned with determining multiple paths through the network, practical implementations are also interested in determining the relative quality of these paths, a form we call *cost-dependent reachability routing*. In this thesis, we propose that reachability routing can be achieved by exploiting the underlying semantics of probabilistic routing algorithms and present the case for *reinforcement learning* (RL) as the framework of choice to realize reachability routing. In particular, by employing the probabilistic nature of RL algorithms, we can guarantee that the likelihood of a packet getting trapped in a loop is zero, although there is a non-zero probability of entering a loop.

1.2 Organization

The rest of this thesis is organized as follows. Chapter 2 gives background into the goals and approaches of various routing algorithms. It also gives a classification of various well known routing algorithms. In Chapter 3 we will present a new model-based routing algorithm based on RL and its implementation; Chapter 4 presents evaluation results. Chapter 5 concludes with a summary of contributions and directions for future research in the area.

Chapter 2. Background

This chapter gives an introduction to the various routing algorithms that are currently used in the Internet. We will present the goals of a routing algorithm and the various approaches to routing. We also give an overview of Reinforcement Learning (RL) [23], the strategies involved in a RL framework and how they can be applied to routing. We assume that the reader has a basic understanding of networking terminology such as: *router, link, port, packet, domain, sub network,* and *routing algorithm*.

2.1 Goals of a Routing Algorithm

The goals of a routing algorithm [31] are *Optimality*, *Simplicity and Low Overhead*, *Stability and Robustness*, *Scalability*, *Convergence,* and *Flexibility*.

Optimality of the algorithm can be defined as the ability of the algorithm to employ the best route from among the set of available routes to the destination. The metrics employed influence the optimality of the algorithm. Achieving optimality is easy with a static network but becomes an issue of concern in a dynamic network.

Simplicity is a desirable criterion in the design of a routing algorithm. In other words, the routing algorithm must offer its functionality efficiently, with a minimum of software and utilization overhead. Simplicity is particularly important when the software implementing the routing algorithm must run on a computer with limited physical resources.

Robustness describes the ability of the algorithm to be resistant to failures. This might require the algorithm to change routes in case of link failures and node failures. Today's best routing algorithms are often those that have withstood the test of time and that have proven stable under a variety of network conditions.

Scalability can be defined as the ability to increase the number of participants in the network without concomitant increase in routing overhead. This is the reason for making the routing model of the Internet to be a *multi-layered hierarchical* one, rather than a *flat* one.

Convergence is the process of agreement, by all routers, on optimal routes. In transient network conditions routes can become unavailable, and hence routers must distribute routing update messages that permeate networks, stimulating recalculation of optimal routes, and eventually causing all routers to agree on these routes. Routing algorithms that converge slowly can result in routing loops or network outages.

Flexibility of the algorithm implies the ability of the algorithm to adapt to changes in the network. Routing algorithms can be programmed to adapt to changes in network bandwidth, router queue size, and network delay, among other variables.

Our discussion implicitly assumes the algorithm to be a correct one. *Correctness* is an axiom that need not be explicitly mentioned.

Routing algorithms can be classified on the basis of the desirable outputs expected from them. Some of the desired outputs are:

Loop free paths: Achieving loop free paths is very essential in routing, as a looped path could indefinitely cause the packets to keep circulating within the network without reaching the destination, resulting in bandwidth wastage and processing overhead at the various intermediate nodes.

Multi-path Routing: Achieving multiple paths from source to destination can lead to effective usage of the network resources and also enable load balancing between multiple paths.

Reachability: Given a connected graph (network), the routing algorithm should find at least one path from any source to every other destination in the network. In one sense, reachability can mean to achieve all possible paths and in the other sense it may just mean to guarantee that there exists a path from every source to every other destination.

2.2 Various Approaches to Routing

There are various approaches to routing [31] and below we give a brief overview of these approaches and compare them.

2.2.1 Static versus Dynamic

Static routing algorithms are table mappings established by the network administrator before commencing routing. They cannot react to changes in the network, and are considered unsuitable for today's large, constantly changing networks.

Dynamic routing algorithms adjust to changing network circumstances by analyzing incoming routing update messages. On determining that a network change has occurred, the routing algorithm recalculates routes and sends out update messages which stimulate other routers in the network to change their routing tables accordingly.

2.2.2 Flat versus Hierarchical

Flat routing protocols distribute/receive information as needed to/from any router that can be reached. The goal is to discover the best route to a destination and no effort is made to organize the network. In short, flat routing protocols consider all routers to be situated on a flat space.

Hierarchical routing protocols group routers together, by function, into a hierarchy. Logical groups of nodes are created and are called domains, autonomous systems, or areas. These groups generally mimic the organization of most companies and therefore support their traffic pattern well. Two types of routers can be distinguished in a hierarchical routing scenario: *access routers* which route traffic within the domain and *backbone routers* which handle traffic across domains.

2.2.3 Host-intelligent versus Router-intelligent

Host-intelligent routing algorithms assume that the source end node will determine the entire route. This is usually referred to as *source routing*. In effect, the routers merely act as store-and-forward devices that forward packets until the data reaches its destination; since these algorithms do not consider the effects of network changes, they can be very data unreliable.

Router-intelligent algorithms assume that the hosts are ignorant of the routes. It is the routers themselves that determine the paths for data to be transmitted on. They calculate the best routes after having received appropriate updates on the status of the network; these algorithms are thus more robust and less sensitive to network changes.

2.2.4 Intra-domain versus Inter-domain

Intra-domain routing protocols handle the routing within a domain or autonomous system (AS). The administrator determines the routing policy to be enforced in the AS.

Inter-domain routing protocols handle routing across domains. These routing protocols are mostly implemented in the border routers of the AS.

2.2.5 Single-path versus Multi-path

Single-path routing protocols determine the various routes and select a single best route based on certain metrics to each destination.

Multi-path routing protocols learn routes and can select more than one route to each destination. They can provide substantially better throughput and reliability when compared to single-path routing protocols. As they are not just interested in finding the single best path to every destination, they have a much higher overhead compared to single-path routing protocols.

2.2.6 Deterministic versus Probabilistic

Deterministic routing algorithms associate, for every destination in the routing table, an outgoing interface identifier and a cost associated with choosing that interface.

Probabilistic routing algorithms associate, for every destination in the routing table, all outgoing interfaces and associated use probabilities. The probabilities are typically designed to reflect the router's sense of optimality; thus an interface with higher probability than another lies on a better path to the given destination. These probabilities must be set to ensure that, when marginalized across all interfaces, the probability is one.

2.2.7 Constructive versus Destructive

Constructive routing algorithms begin with an empty set of routes and incrementally add routes till they reach the final routing table. Intuitively, a constructive algorithm treats routes as 'guilty until proven innocent' [24].

Destructive routing algorithms begin by assuming that all possible paths in the network are valid, i.e. they treat the network as a fully connected graph. Intuitively, a destructive algorithm treats routes as 'innocent until proven guilty' [24].

In the subsequent sections, we will discuss some of the popular routing algorithms currently used on the Internet and classify them on the basis of the various dichotomies established above.

2.3 Distance Vector Routing Algorithm (RIP)

Distance-vector routing algorithms build their routing tables using an iterative computation of the distributed Bellman-Ford algorithm. The most common manifestation of the distance-vector algorithm on the Internet is the *Routing Information Protocol* (**RIP**) [11][13].

Every router maintains a routing database, which contains only the best known path costs to each destination router in the AS. As the name implies, every router forwards its current routing table to all of its immediate neighbors, as a vector of distances to all nodes in the network from itself [18]. Neighbors receiving the routing table check for every destination in the received table, to see if it can reach the destination using a better path (lower cost) through the router that advertised. If the received routing table entry has a better cost, the target router replaces its path cost and corresponding outgoing interface with the information received, and propagates the new information. The algorithm stabilizes when every router in the system has indirectly received routing tables from every other router in the AS.

At the end of the first iteration, each router knows the current best path costs to all routers within 1 hop from itself – a graph with a diameter of 2. With every passing iteration, the diameter of the graph known to a router increases by 1. The algorithm finally stabilizes when each router has expanded its horizon to the diameter of the network. Count-to-infinity and looping [18], due to transient network conditions, are potential problems in distance-vector algorithms.

2.4 Link State Routing Algorithm (OSPF)

Based on Dijkstra's shortest path algorithm, link-state routing algorithms avoid problems such as count-to-infinity and looping in the distance-vector approach by constructing a minimal spanning tree over the topology of the network. Link-state routing algorithms are characterized by a global information collection phase, where each router broadcasts its local connectivity to every other router in the network. Every router independently assimilates the topology information to build a complete map of the network, which is then used to construct routing tables. The most common manifestation of link-state algorithms is the *Open Shortest Path First* (**OSPF**) routing protocol [15][16], developed by IETF for TCP/IP networks.

Each node broadcasts a link-state packet, which contains the information about its neighbors, the interfaces associated with that node, and the metric associated with the interfaces. After such broadcasts have flooded through the network, every router running the link-state algorithm constructs a map of the network topology and computes the cost of each link of the network. Using that topology, each router then constructs a shortest path tree to all other routers in the autonomous system. From the tree, a router determines the outgoing interface for each destination and stores this information in its routing table.

Link-state algorithms are generally dynamic in nature, as the routers exchange information whenever there is a change in the topology or link costs and recompute shortest path trees to ensure consistency with the current state of the network.

2.5 Path Vector Routing Algorithms (BGP, IDRP)

In path vector routing algorithms, a node advertises paths to the destination instead of just the metric to that destination. Like distance-vector algorithms, every node advertises a vector with update information to its neighbors. Unlike distance-vector algorithms, the vector not only contains the metric of the path but also the path, a sequence of nodes, which lead to the destination.

The *Border Gateway Protocol* (**BGP**) and the *Inter-Domain Routing Protocol* (**IDRP**) are two common implementations of path vector routing algorithms. The main reason for advertising the path to the destination, rather than just the metric, is to prevent loops from occurring in the system. Upon receiving the update from a neighbor, a node can just parse the list of nodes in the path to check if the node itself is present in the path already. The occurrence indicates a loop and the node discards such paths.

Unlike link-state and distance-vector algorithms, path vector algorithms are generally used between autonomous systems i.e., operating at the scope of backbone networks. Also, the requirement to satisfy policies imposed by local domains in the Internet is possible with the path vector approach, since only those paths that satisfy all the policies and constraints imposed by all the domains along the path need to be chosen.

2.6 MP-Scout

MP-Scout [6] is a distributed routing algorithm based on backward learning to determine loop-free multi-paths. The algorithm periodically floods *scout* packets that explore paths to a destination in reverse. MP-Scout extends on SP-Scout (Single path routing algorithm) to provide multipath routing capability.

In SP-Scout, a node R sends out one scout [R, C_R] in every Broadcast Interval (BI) to all its neighbors where C_R is initially set to 0. Node P upon receiving from some node Q, the scout [R, $C_{R'}$] updates the cost of the scout to $C_{R''}$ where $C_{R''} = C_{R'}$ + cost (P->Q). If this is the first ever scout originating from R received by P, P forwards the scout to every other neighbor except Q. During the same BI, P might receive more scouts originating from R through other neighbors. From these scouts, it finds the cheapest cost to reach R but does not forward the scout as it has already forwarded one scout in the same interval. The neighbor from whom P received the cheapest cost is marked as the designated neighbor for the destination R. In the next interval, node P does not send out a scout from R until it has received a scout from the designated neighbor. If no scout came from the designated neighbor in the previous BI (due to node/link failure), P sends out the first scout it receives from R and re-determines its designated neighbor. SP-Scout distinguishes between BIs by associating a sequence number with each scout.

To extend SP-Scout as a MP-Scout, the scouts have a Multipath ID associated with them. The routers perform a prefix matching on the ID to determine the loops and all paths leading into loops are eliminated. Routed packets also have the ID in them and the routers route the packets accordingly. MP-Scout introduces two types of thresholds; a threshold on the number of paths that a router can store for a destination and a *data threshold*, where P discards scouts advertising costs to a destination that are greater than the data threshold of the minimal cost known thus far to the destination.

2.7 MDVA

In this and subsequent sections, we discuss three multipath routing protocols: *Multipath Distance Vector Algorithm* (**MDVA**), *Multipath Partial Dissemination Algorithm* (**MPDA**), and **MPATH**, all part of the dissertation of Vutukury [30]. The following table [26] lists out the common notations used in the algorithms.

N	Set of nodes in the network
N^i	Set of neighbors of node i

$\mathbf{S^i_j}$	Next-hop choices at i for destination j a subset of N^i
$\mathbf{SG_j}$	Routing graph implied by the S^i_j of destination j
$\mathbf{D^i_j}$	Distance of node i to j as known to i
$\mathbf{I^i_k}$	Cost of link (i, k)
$\mathbf{D^i_{jk}}$	Distance of node k to j as reported by k to i
$\mathbf{FD^i_j}$	Feasible distance is an estimate of D^i_j
$\mathbf{RD^i_j}$	Distance to j reported by node I to its neighbors
$\mathbf{SD^i_j}$	Best distance to j through S^i_j
$\mathbf{WN^i_j}$	Set of neighbors that are waiting for replies
$\mathbf{state^i_j}$	State maintained by node i for node j (ACTIVE or PASSIVE)

Table 2-1 Common notations used in MDVA, MPDA, and MPATH algorithms (Taken from [26]).

All three algorithms make use of loop-free invariants (LFI) [27] to ensure loop freedom at every instance. The LFI conditions capture all previous loop-free conditions in a unified manner that simplifies protocol design and correctness proofs. They are given by:

$$FD^i_j(t) \leq D^k_{ji}(t) \quad k \in N^i$$

$$S^i_j(t) = \{ k \mid D^i_{jk}(t) < FD^i_j(t) \}$$

These conditions state that, for each destination *j*, a node *i* can choose a successor whose distance to *j*, as known to *i*, is less than the distance of node *i* to node *j* that is known to its neighbors. The LFI conditions should be valid at any instance of time *t* (dynamic state of the network), hence FD^i_j, S^i_j, and D^i_{jk} are represented as functions of *t*.

The *Multipath Distance Vector Algorithm* [26] solves the count-to-infinity problem and computes multipaths to destinations. Circular computation of distances that occur in a Distributed Bellman-Ford (DBF) algorithm can be prevented if distance information is propagated along a Directed Acyclic Graph (DAG) rooted at a destination. Given a DAG, each node computes its distance using distances reported by “downstream” nodes and reports its distance to “upstream” nodes. This method called *diffusing computation* was suggested by Dijkstra et al [9] to ensure termination of distributed computation; a diffusion computation always terminates due to the acyclic ordering of the nodes. MDVA uses the DBF algorithm to compute D^i_j, and therefore SG_j (DAG), while propagating distances along the SG_j to prevent the count-to-infinity problem.

Each node maintains a *main table* that stores D^i_j, S^i_j, FD^i_j, RD^i_j, SD^i_j and WN^i_j. Each node also maintains a *neighbor table* for each neighbor k that contains D^i_{jk}. The *link table* stores the cost I^i_k of adjacent link to each neighbor k. Since MDVA is a constructive algorithm, the nodes initialize all the distances to null.

Nodes executing MDVA exchange messages of the form [*type, j, d*], where d is the distance of the node sending the message to destination j and type is *UPDATE, QUERY* or *RESULT.* A node updates its distance vectors with the arrival of any message, the change of cost of an adjacent link, or a change in status (up/down) of an adjacent link. When an adjacent link becomes available, the node sends an UPDATE message to all the destinations j over the link. When the adjacent link (i, m) fails, the neighbor table associated with m is cleared and the cost of the link is set to infinity.

In MDVA, a node can be in *ACTIVE* or *PASSIVE* state with respect to a destination j and is represented by state^i_j. Initially, all nodes are in PASSIVE state, and as long as link costs decrease, MDVA works identically to DBF and nodes will remain in PASSIVE state. However, if the distance to a destination increases, a diffusing computation [9] (ACTIVE state) is initiated by sending QUERY messages to all the neighbors with the best distance SD^i_j through S^i_j waiting for the neighbors to reply. When all the RESULT messages are received, the node can ensure that the neighbors have incorporated the distances that the node reported, and safely transits to PASSIVE state.

2.8 MPDA

Multipath Partial Dissemination Algorithm (**MPDA**) [27] is an extension of the PDA [30] algorithm, which is a shortest path routing algorithm in its own right. By incorporating loop free invariants into PDA, we obtain MPDA. PDA, a link state algorithm propagates enough link-state information in the network such that each router has sufficient information to compute shortest paths to all destinations.

Each router i running the PDA maintains: a *main topology table*; which stores the characteristics of each link known to router i, a *neighbor topology table;* which stores the link state information communicated by the router's neighbors, a *distance table*; which stores the distances from the router to each destination and also the distances from every neighbor to each destination, a *routing table*; which stores for each destination j the successor set S^i_j and feasible distance FD^i_j, and finally a *link table* which stores the cost of the adjacent link to each neighbor. The routers exchange link state update (LSU) [30] messages which contain information indicating *addition, deletion,* or *change* in cost of a link in the router's main topology table. On receiving a LSU from a neighbor or when there is a change in an adjacent link, the router constructs a shortest path tree based on the information received and reports the differences between successive trees to its neighbors. Conflicts in reported link information are resolved by choosing the link which reports a lower cost to a destination.

Similar to MDVA, the router is either in PASSIVE or in ACTIVE state. Initially the router is in PASSIVE state and when it receives an event that affects a change in topology, the router sends those changes to its neighbors, changing from PASSIVE to ACTIVE state, and waits for ACKs from its neighbors. In MPDA, all the neighbors acknowledge each LSU sent by a router before it sends the next LSU. During the ACTIVE phase, no update of the main topology table takes place while updates of the neighbor topology table and link table are allowed. When all the ACKs are received, the router transits from ACTIVE to PASSIVE state and updates its main topology table; and if there is a change, it immediately goes into ACTIVE state and sends out LSU to all its neighbors.

2.9 MPATH

MPATH [28][29] makes use of distance-vectors combined with the identity of the second-to-last node, also called predecessor node that is just before the destination node on the shortest path. It provides multiple paths of unequal cost to each destination that are free of loops at every instant – in steady state as well as during network transitions. It uses a synchronization mechanism that spans only one hop, making it more scalable than routing algorithms based on diffusing computations spanning multiple hops.

The basic approach consists of nodes first computing shortest distances to destinations and then using the distances along with loop-free invariants to obtain a loop-free routing graph for each destination. The goal of MPATH is to maintain the routing graph denoted by the link set $SG_{j,}$ in presence of changing link costs, such that it is a directed acyclic graph at every instant. The key idea is to generalize the shortest-path trees to shortest-multipaths. To build shortest multipaths, a node stores costs of links on its shortest path tree and a copy of each neighbor's shortest path tree. The nodes in MPATH exchange distances to destinations along with the addresses of the predecessor node on the shortest path to a destination. MPATH translates this information to link costs and internally works with links rather than distances, and when changes to topologies have to be reported, the internally represented topology is translated back to distances and predecessors.

2.10 Q-routing

Q-routing [4], one of the first Reinforcement learning (RL) algorithms for routing, is an online asynchronous relaxation of the Bellman-Ford algorithm used in distance-vector protocols. Reinforcement learning [12] is a branch of machine learning that is increasingly finding use in many important applications, including routing. Here, populating routing tables is viewed as a problem of learning the entries; we hence use the term *learning* in this thesis synonymously with the task of determining routing table entries. The salient feature of RL algorithms is the probabilistic nature of their routing table entries, making them suitable for either single-path or multi-path routing.

Any RL problem is defined by a set of *states*, a set of allowable *actions* at each state, *rewards* for transitions between states, and a *value function* that describes the objective of the RL problem. In our problem of multi-path routing, the states are the routers and an action denotes the choice of the outgoing link. The environment supplies the rewards and the value function describes the goal imposed on RL algorithm. In our case, rewards could be set to reflect the quality of the link (e.g. cost) and the value function typically tries to maximize or minimize an objective function (e.g. minimize cumulative sum of link costs when following a path).

In Q-routing, every router *x* maintains a measure $Q_x(d, i_s)$ that reflects a metric for delivering a packet intended for destination *d* via interface i_s. A deterministic routing policy is followed and the packet is routed along

$$argmax_k\, Q_x(d, i_k)$$

i.e. to choose the interface with the highest Q value for destination d, among all the k interfaces.

The operation of the routing algorithm is as follows. All the Q entries are initialized to some small values. Given a packet, a router x deterministically forwards the packet to the next best router y, determined from the Q value. Upon receiving this packet, y immediately provides x an estimate of *its* best Q (to reach the destination). x then updates its Q-values to incorporate the new information. The following routing update is presented in [4]:

$$Q_x(d, i_s) = Q_x(d, i_s) + \eta\, \{(max_k\, Q_y(d, i_k) + \zeta) - Q_x(d, i_s)\}$$

where ζ accounts for the time spent by the packet in x's queue and also the transmission time from x to y. η is called *learning rate* or a *step-size* and is a standard fixture in iterative improvement algorithms [3]. It is typically set to produce a step-size schedule that satisfies the stochastic approximation convergence conditions [2].

Q-routing is not guaranteed to converge to the shortest path. In fact, as Subramanian et al. [22] point out, the algorithm will switch to using a different interface only when the one with the current highest Q metric experiences a *decrease*. An improvement (e.g., shorter delay) in an interface that doesn't have the highest Q metric will usually go unnoticed. Another problem with the Q-routing algorithm is that the routing overhead is proportional to the number of data packets.

2.11 ANTS

In the previous section we gave a brief peek into RL but to more completely model routing as a RL problem, we need strategies for a) gathering information about the environment, b) deriving routing tables by credit assignment, and possibly c) building models of relevant aspects of the environment.

Learning in RL is based on trial-and-error and organized in terms of episodes [12]. An episode consists of a packet finding its way from an originating source to its intended destination. Routing table probabilities are initialized to small random values, thus enabling them to begin routing immediately except that most of the routing decisions will not be optimal or even desirable. To improve the quality of the routing decision, a router can 'try out' different links to see if they produce good routes, a mode of operation called *exploration.* Information learnt during exploration can be used to drive future routing decisions. Such a mode is called *exploitation.* Both exploration and exploitation are necessary for effective routing.

To overcome the problems of selective reinforcement of Q-routing, wherein exploration only happens along the currently exploited path, Subramanian et al. [22] propose the separation of data collection aspects from the packet routing functionality. In their ant based algorithms, messages called *ants* are used to probe the network and provide reinforcements for the update equations. Here ants perform the role of gathering information about the network. The parameters of interest in the case of ants are the rate of generation of ants, the choice of their destinations, and the routing policy used for ants.

RL algorithms perform iterative stochastic approximations and the rate of ant generation affects their convergence properties [8]. The second parameter of interest is the choice of the ant destinations. From the perspective of multi-path routing, we would like to choose destinations that will provide the most useful reinforcement updates; hence a uniform distribution policy assures good exploration. Finally, the policy used to route ants affects the paths that are selectively reinforced by the RL algorithm. As our goal is to discover all possible paths, the policy used to route ants should be independent of that of the data traffic. If we do not separate the policies, then we would end up with the same problem of selective reinforcement as Q-routing.

In the context of RL framework, effective credit assignment strategies rely on the expressiveness of the information carried by the ants. The central idea behind credit assignment is to determine the relative quality of a route and apportioning blame. In the case of routing, credit assignment creates a 'push-pull' effect. Since the link probabilities have to sum to one, positively reinforcing a link (push) results in negative reinforcements (pull) for other links.

In the simplest form of credit assignment, ants carry information about the ingress router and path cost as determined by the network's cost metrics. At the destination, this information can be used to derive reinforcement for the link along which the ant arrived [22](*backward learning*). Another strategy is to reinforce the link in the forward direction by sending an ant to a destination and bouncing it back to the source [8] (*forward learning*). Subramanian et al. [22] adapt the former approach. Ants proceed from randomly chosen sources to destinations independent of the data traffic. Each ant contains the source where it was released, its intended destination, and the cost c experienced thus far. Upon receiving an ant, a router updates its probability to the ant source (not the destination), along the interface by which the ant arrived. This is a form of backward learning and is a trick to minimize ant traffic.

Specifically, when an ant from source s to destination d arrives along interface i_k to router r, r first updates c (the cost accumulated by the ant thus far) to include the cost of traveling interface i_k in reverse. r then updates its entry for s by slightly nudging the probability up for interface i_k (and correspondingly decreasing the probabilities for other interfaces). The amount of the nudge is a function of the cost *c* accumulated by the ant. It then routes the ant to its desired destination *d*. In particular, the probability p_k for interface i_k is updated as:

$$p_k = \frac{p_k + \Delta p}{1 + \Delta p}, p_j = \frac{p_j}{1 + \Delta p}, 1 \le j \le n, j \ne k$$

where $\Delta p = \frac{\lambda}{f(c)}, \lambda > 0$ and *f(c)* is a non-decreasing function of *c*.

Two types of ants, namely *regular ants* and *uniform ants,* are supported to handle the routing aspect of the ants. Regular ants are forwarded probabilistically according to the routing tables, which ensure that the routing tables converge *deterministically* to the shortest paths in the network. Regular ants treat the probabilities in the routing tables as merely an intermediate stage towards learning a deterministic routing table. They are good exploiters and are beneficial for convergence in static environments. In uniform ants, the ant forwarding probability is a uniform distribution, wherein all links have equal probability of being chosen. This ensures a continued mode of exploration and helps keep track of dynamic environments. In such a case, the routing tables do not converge to a deterministic answer; rather, the probabilities are partitioned according to the costs. The constant state of exploration maintained by the uniform ants ensures a true multi-path forwarding capability.

2.12 AntNet

The AntNet system of Di Caro and Dorigo [8] is a very sophisticated reinforcement-learning framework which is adept at handling dynamic routing conditions. Like the algorithms of Subramanian et al., this system uses *ants* to probe the network and sufficient exploration is built in to prevent convergence to non-optimal tables in many situations. It is a model-based system, where every router maintains a model of the local traffic experience which is adaptively refined and utilized to score ant travel times.

Update rules are very carefully designed and implemented in AntNet to ensure proper credit assignment. For instance, the costs accumulated by ants are not used to update the link probabilities in reverse. Instead, a so-called *backward ant* is generated that travels the followed path in reverse and updates the link probabilities in the correct, forward direction. This overcomes the assumption of link cost symmetry made by both the ant algorithms in the previous section. Also, any ants encountering cycles in their paths are discarded and are not used to update the routing tables.

Table 2-2 summarizes the various routing algorithms studied in this chapter.

Approaches \ Routing Algorithms	Distance Vector	Link State	Path Vector	MP-Scout	MDVA	MPDA	MPATH	Q-Routing	ANTS	AntNet
Static										
Dynamic	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Host-intelligent										
Router-intelligent	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Intra-domain	✓	✓		✓	✓	✓	✓	✓	✓	✓
Inter-domain			✓							
Single-path	✓	✓	✓					✓		
Multi-path				✓	✓	✓	✓		✓	✓
Deterministic	✓	✓	✓	✓	✓	✓	✓	✓		
Probabilistic									✓	✓
Constructive	✓	✓	✓	✓	✓	✓	✓			
Destructive								✓	✓	✓

Table 2-2 Summary of the routing algorithms studied, along the various dichotomies established in this chapter.

Chapter 3. Using RL for Model-Based Routing

In this chapter we discuss how we model the problem of multipath routing as an instance of distributed model-based reinforcement learning. We will give an overview of the data structures used, the operation of the algorithm, and the design decisions involved at the various stages.

3.1 Motivation

The primary design objective is to achieve cost-sensitive multi-path forwarding while at the same time, eliminating the entry of loops 'as much as possible'. We have made a series of improvements to the uniform ants algorithm proposed by Subramanian et al [22], culminating in a novel model-based routing algorithm.

Let us begin by observing that uniform ants are natural multi-path routers; according to Proposition 2 in Subramanian et al. [22], the probability of choosing an interface is aligned in inverse proportion to cost ratios. The reader might be tempted to conclude that uniform ants support reachability routing; however consider the three 'velcro' topologies of Figure 3.1. These topologies have the same underlying graph structure but differ in the costs associated with the main branch paths (the direct path from 0 to 19, and the path through nodes 1, 7, and 13). Uniform ants explore all available interfaces with equal probability; while this makes them naturally suitable for multi-path routing, it also creates a tendency to reinforce paths that have the least amount of decision making. To see why, recall that the goodness of an interface is inversely proportional to a non-decreasing function of the cost of the path along that interface. The cost is *not* simply the cost of the shortest path along the interface, but is itself assessed by the ants during their exploration; hence the routing probability for choosing a particular interface is implicitly dependent on the number of ways in which a costly path can be encountered along the considered interface. The presence of loops along an interface means that there are greater opportunities for costly paths to be encountered (causing the interface to be reinforced negatively) or for the ants to loop back to their source (causing their absorption, and again, no positive reinforcement along the interface). The basic problem can be summarized by saying that 'interfaces that provide an inordinate number of options involving loops will not be reinforced, even if there exists high-quality loop-free sub-paths along those interfaces.' Mathematically, this is a race between the negative reinforcements due to *many* loops (and hence absorptions), and positive reinforcements due to *one (or few)* short or cheap paths. As a result, the interface with the fewer possibilities for decision making wins, irrespective of the path cost. Hence in the topologies shown in Figure 3.1, uniform ants will reinforce along: the costliest path (left), among one of many cheapest paths (center) and the cheapest path (right). Notice that using regular ants to prevent this incessant multiplication of probabilities is not acceptable, as we will be giving up the multi-path forwarding capability of uniform ants.

Ideally, we want our ants to have selective amnesia, behaving as uniform ants when it is important to have multipath forwarding and morphing into regular ants when we do not want loops overshadowing the existence of a cheap, loop-free path. We present a model-based approach that achieves this effect by maintaining a statistics table independent of the routing table. The basic idea is to make routers recognize that they constitute the fulcrum of a loop with respect to a larger path context. For instance, in Figure 3.1, nodes 1, 7, and 13 form fulcrums of loops, which should not play a role in multi-path forwarding from, say, node 0 to node 19. The statistics table keeps track, for every router (node) and destination, the number of ants generated by it and that returned (without reaching its intended destination). Using this statistic, for instance, node 1 can reason that all ants meant for destination 19 returned to it, when sent along the interface leading to node 2. This information can be used to reduce the scope of multi-path forwarding, on a per-destination basis. The statistics table serves as a discriminant function for the choices indicated by the routing table, while the routing table reflects the reinforcement provided by the uniform ants.

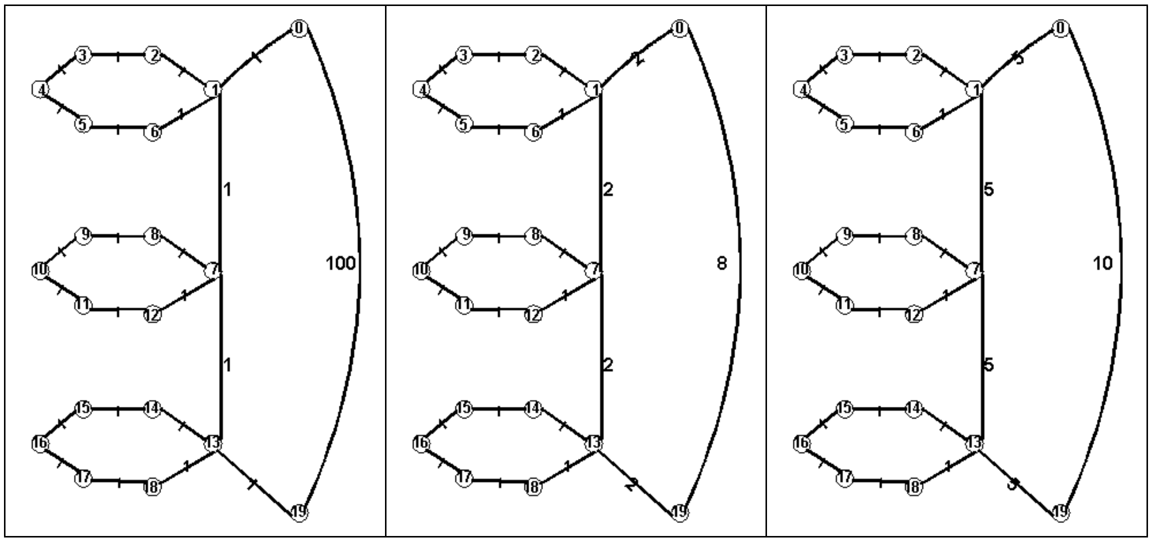


Figure 3.1 Velcro topologies with different cost ratios.

3.2 ANT Structure and Data Structure Design

In this section, we will give an overview of the ANT structure [22] and the data structures used for the routing and the statistics table at every node.

3.2.1 ANT Structure

Ants are small packets used to explore the network and gather information about the network. Periodically every source *s* generates, to every other destination *d*, ants of the form [*s, d, c,* o_i], where *c* is the cost associated with the ant and o_i is the outgoing interface from the source router. When the ants are created the cost *c* is initialized to 0. All the intermediate routers along the path from the source to destination increment the cost *c* to reflect the cost in reverse (when a message traverses a link from node a to node b, c is incremented by the cost of the link from b to a). When the ant reaches the destination *d*, the cost *c* is the end-to-end cost of sending a message from source *s* to destination *d*. Note the intermediate nodes along the path do not update that o_i.

3.2.2 Routing Table Structure

Routing table at every node is a two-dimensional array of the probabilities of using various interfaces to reach destinations. r_i[j][k], maintained at node *i,* is the probability with which the interface *k* of node *i* is chosen to reach destination *j*. Initially the probabilities for all destinations are distributed equally across all the interfaces. This is inline with the destructive property of RL routing algorithms wherein all interfaces are "innocent until proven guilty".

3.2.3 Statistics Table Structure

Statistics table is also a two dimensional structure like the routing table but every node has two statistics tables: sent_stat_table$_i$[j][k], corresponds to count of ants sent along interface *k* to destination *j* originating from node *i*, and recd_stat_table$_i$[j][k] is the number of ants sent along the interface *k* which returned to the source *i*. There are two reasons for maintaining the statistics of the ants at the source node only and not at the intermediate nodes.

The first reason is to allow for scalability of the network. If every intermediate node *n* along the path of an ant from source *i* to destination *j* increments its statistics table sent_stat_table$_n$[j][m] when it forwards the ant along the interface *m*, it will necessitate the ant to have a provision to save the outgoing interface for each node along its path, so that the node will be able to identify if the ant loops back to itself. Accommodating such a structure in large topologies can result in unbounded growth of the ant's size.

Second, the ants are not forwarded when they reach the destination or the source. By updating the statistics table only at the source nodes, if the ant doesn't loop back to itself, the source node can safely assume that it has reached the destination (Under 100% reliability conditions that no packets are dropped); whereas the intermediate nodes have no way of determining whether the ant reached the destination successfully, or looped back to the source node itself.

3.3 Overview of the Algorithm

```
begin:
    Uncontrolled Exploration
    Controlled Exploration
end.

Exploration (Uncontrolled/Controlled)

begin:

    for every node in the topology
    begin:
        GenerateAnt;            /* Periodically Generate Ant */
        SelectInterface;        /* (Uncontrolled/Controlled) */
        UpdateModel;
        ForwardAnt;
    end.

    On receiving an ant
    begin:

        if the receiving node is the source of the ant
        begin:
            UpdateModel;
            DestroyAnt;
        end.

        if the receiving node is neither the source nor the destination
        begin:
            UpdateRouteTable;
            SelectInterface;    /* (Uncontrolled/Controlled) */
            ForwardAnt;
        end.

        if the receiving node is the intended destination of the ant
        begin:
            UpdateRouteTable;
            DestroyAnt;
        end.

    end.            /* End of receive ant procedure */

end.    /*End of algorithm */
```

Table 3-1 Model-based routing algorithm.

3.4 Description of the Algorithm

The algorithm consists of two stages -- ***Uncontrolled Exploration*** and ***Controlled Exploration***. In both forms of exploration, every node generates ants periodically destined for every other node in the topology. The algorithm uses uncontrolled exploration to collect information about the topology and uses that information to build a model to control future exploration at the nodes. The information collected during the controlled exploration is used to update the model, as well. The two forms of exploration work almost identically except for the *SelectInterface* method. Below we will give a brief description of the various methods mentioned above.

3.4.1 GenerateAnt

Every node in the topology *periodically* generates ants to every other node in the topology. This method ensures that the choice of the destination node for each ant at each node is uniformly distributed, so that the number of ants generated to the various destinations is nearly equal. The generated ant is of the form [*s*, *d*, *0*, undefined], where *s* is the source node generating the ant and *d* is the intended destination. The initial cost *c* associated with the ant is set to *0.* As *SelectInterface* is the method that determines the output interface, the output interface is undefined when the ant is created.

3.4.2 SelectInterface

This method differentiates between the two forms of exploration mentioned above:

Uncontrolled Exploration: Here the choice of the outgoing interface at every node along the path from the source to destination is unbiased, i.e. every interface at that node has equal probability of being chosen as the outgoing interface. The node generating the ant chooses one interface from its interfaces and forwards the ant along that interface. If an intermediate node (not the intended destination node) receives an ant along the interface 'A' and has interfaces other than 'A', it forwards the ant on some interface other than 'A'. If it does not have any other interface then it *sends-back* along the interface 'A' itself.

Controlled Exploration: Here the choice of outgoing interface is controlled by a factor called the *threshold factor* ($\boldsymbol{\tau}$). The threshold factor not only affects the multipath capabilities of the routing algorithm, but also its correctness with respect to the routing of packets (measured by the percentage of packets successfully reaching their intended destinations) and its loop-free capabilities. A discussion on the effect of the threshold factor on various topologies is deferred to Chapter 5. Formally, the $\boldsymbol{\tau}$ factor works in the following manner:

A node '*i'* (source or intermediate), when it needs to forward the ant intended for destination *'j'*, finds the ratio of recd_stat_table$_i$[j][k] to sent_stat_table$_i$[j][k] for each of its interfaces *'k'*. All those interfaces whose ratios are less than the threshold **τ** are eligible for selection as a forwarding interface. The selection policy is to choose among the eligible interfaces equiprobably.

There are three special cases to be handled in the case of controlled exploration:

Case 1: When an ant arrives at a leaf node, i.e. there are no other interfaces other than the incoming interface, and if it is not the intended destination then the node *'sends-back'* the ant along the same interface.

Case 2: When all the interfaces at the intermediate node are ineligible, i.e. their statistic table ratios are above the threshold **τ**, then the node *'sends-back'* the ant along the interface it originally received the ant from.

Case 3: When all the interfaces at the source node are ineligible then the source node uses the uncontrolled exploration selection policy to break the deadlock. This case is a very rare occurrence and occurs only when **τ** is set to a very low value.

Once the outgoing interface is selected the next step is to forward the ant along the chosen interface (*ForwardAnt*). In the case of source node, before calling the *ForwardAnt, UpdateModel* is called to update the statistics table.

3.4.3 UpdateModel

The source node calls this method (the node generating the ant) only on two occasions:

When the node generates the ant [*i, j, c, k*], it increments its statistic table entry sent_stat_table_i[j][k] by 1 to indicate that interface *k* was chosen by *i* to forward the ant intended for destination *j*.

When the ant [*i, j, c, k*] loops back to the source node, the statistic table entry recd_stat_table_i[j][k] is incremented by 1 to indicate that the choice of interface *k* to route the ant intended to destination *j* resulted in a loop.

3.4.4 ForwardAnt

As indicated by its name, this method is used to forward the ants from the current node to the next node along the interface chosen by the *SelectInterface* method.

3.4.5 DestroyAnt

When the ant reaches the intended destination or loops back to its source itself, the ant is not forwarded further and the node absorbs the ant.

3.4.6 UpdateRouteTable

When any node *t* (intermediate or the intended destination) other than the source node, receives an ant [*i, j, c, k*] on interface *l* from node *y,* it updates the cost *c* by adding the cost of traversing the interface *l* in reverse, and then updates its routing table entries for node *i* as follows:

$$r_t[i][l] = \frac{r_t[i][l] + \Delta p}{1 + \Delta p},\ r_t[i][m] = \frac{r_t[i][m]}{1 + \Delta p},\ 1 \le m \le n, l \ne m$$

where $\Delta p = \frac{\lambda}{f(c)}, \lambda > 0$ and *f(c)* is a non-decreasing function of *c*.

3.5 Qualitative characteristics of model-based routing algorithm

The model-based routing algorithm presented above discards all useless loops. Consider the ‘velcro’ topologies shown in Figure 3.1, say for instance, when node 1 sends out a packet intended for a destination other than those nodes in the loop pivoted at 1, either on interface leading to node 2 or node 6, will result in the packet returning to node 1. From the statistics table, node 1 will learn that those interfaces are useless for forwarding packets to certain destinations and hence avoid them in the future. By discarding all the useless loops, this algorithm overcomes the problem of uniform ants algorithm wherein only the path with the least decision-making is reinforced.

The threshold factor $\boldsymbol{\tau}$ influences the reinforcement of the various paths of a topology. At very high values of $\boldsymbol{\tau}$, the algorithm tends towards behaving like uniform ants while continuing to avoid all the useless loops. For instance a $\boldsymbol{\tau}$ value of 1 means that an interface where all but one packet sent on it came back can still be selected as an outgoing interface. But this setting still avoids all the interfaces that lead to useless loops, as all packets sent along them must have come back to the sender. At very high $\boldsymbol{\tau}$ values, certain packets may encounter one or more loops along their path that is unavoidable. At very low values of $\boldsymbol{\tau}$, the nodes have a limited selection of interfaces to choose from due to the stringent criteria, which will affect our goal of multi-path routing, but will greatly decrease the probability of encountering a loop. The choice of $\boldsymbol{\tau}$ factor determines the multipath, correctness, and loop-avoidance capabilities of our algorithm. $\boldsymbol{\tau}$ can either be set to a fixed value (for the network, or on a per-router or per-router-per-destination basis) or can be adaptively refined to optimize model-based routing for various criteria.

Chapter 4. Evaluation of Model-Based Routing

This chapter is concerned with the evaluation of model-based routing for achieving reachability goals. We will first give an overview of the simulator which was used to implement the routing algorithm and also discuss the topologies used to test the routing algorithm. The evaluation, presented next, is multi-pronged and considers the following issues:

1. How does model-based routing algorithm perform on synthetic topologies that require significant decision making for resolving optimal path costs? (Section 4.3)
2. How quickly does the model-based algorithm converge to self-consistent routing probabilities along opposite directions of a network path? (Section 4.4)
3. Can model-based routing be used to mimic shortest-path routing? (Section 4.5.1)
4. What is the tradeoff between multipath forwarding capability and the potential for loops, as a function of network topology and algorithmic driver parameters? (Section 4.5.2)
5. What is the distribution of loop frequency in realistic topologies? (Section 4.6)
6. Does model-based routing converge to achieve cost-sensitive forwarding of data packets? (Section 4.7)

4.1 Experimental Setup

To measure the performance of our cost-sensitive reachability routing algorithm, we coded a detailed discrete event simulator in C, which simulates a standard point-to-point topology based network. The simulated network is modeled as a set of nodes interconnected over point-to-point links, with an associated cost. The discrete event simulator was derived from work done in [25], and has been used in several networking courses to model routing algorithms.

The simulator runs at a resolution of 1 μs and an integer value defined at the initialization of the simulation determines the duration of the simulation. In our case, the simulation runs were set to INT_MAX (2147483647 as defined in <limits.h>). As it is a discrete event simulator, every action takes place after the expiry of a timer and the simulator is programmed to run in *uncontrolled exploration* mode for the first one eighth of the time and in *controlled exploration* mode for the remaining time. Each node generated an ant every 10000 μs. For the purpose of this thesis, we programmed the link layer of the simulator to be reliable i.e. it does not introduce any errors or drop packets.

4.2 Topologies

An utility provided along with the simulator [25], which given the number of nodes in the network and number of interfaces per node, generates four different interconnected topologies for the network namely: tree, fully connected mesh, arbitrary graph, and loop topologies. The tree and arbitrary graph topologies were used to generate various topologies for the simulations.

Using the manual topology generator provided along with the simulator [25], complex topologies like the 'velcro' (explained in next section) and 'dumbbell' topologies were created. These topologies have some intrinsic characteristics helpful in bringing out the effectiveness of our algorithm.

A mesh topology generator was written in C, which given the number of rows and columns in the mesh, will generate a perfect mesh topology wherein all the interior nodes will be of degree 4 and all the boundary nodes will be of degree 2 or 3.

Finally, BRITE, the Boston university Representative Internet Topology gEnerator [14], was used to generate large Internet scale topologies. It provides a wide variety of generation models, as well as the ability to easily extend such a set by combining existing models or adding new ones. We used the *Router Waxman Flat Router-level* model, which is governed by a power law, to generate the topologies. A program in C was written to convert the topology format generated by the BRITE to the format required by our simulator. Topologies with sizes ranging from 20 to 200 nodes were generated using BRITE.

In the subsequent sections, we present the simulation results from our implementation that clearly shows the performance benefits of our approach. First, we validate the model based routing algorithm using synthetic topologies such as the velcro topologies. We also present a subtle modification to the algorithm, avoiding sub path reinforcement, which will result in better performance on certain types of topologies. Second, we quantify the convergence of our routing algorithm by measuring the correlation of path costs and hop counts between all packets sent to and originating from the nodes under consideration. In our case, the nodes under consideration were those with the maximum and minimum degree. Third, we study data traffic across the network based on converged routing tables and introduce a new factor called the *reachability factor* (**φ**) that controls the choice of the outgoing interfaces. We investigate the effect of the threshold factor (**τ**; refer to the previous chapter) and the reachability factor (**φ**) on various topologies with the help of an *operating curve* aimed at helping network administrators in choosing the ideal threshold and reachability factors for his network. We also show that by making the nodes always choose the interface with the highest probability for the intended destination, our model-based routing algorithm behaves in the same way as any other single-path deterministic routing algorithm i.e., it provides loop-free shortest-paths with guaranteed delivery for all packets. Finally, we show that even though the goal of every multi-path routing algorithm is to avoid loops, our model-based routing algorithm does not guarantee a complete elimination of loops. Nevertheless our algorithm guarantees that a packet will eventually exit the loop and reach its intended destination. We study the distribution of loops encountered by packets and show that a vast majority of packets encounter only a small number of loops, or none at all.

4.3 Validation of Model Based Routing (Velcro Topologies)

In this section, we focus on topologies modeled after the velcro graph for two reasons. First, these topologies embody the most difficult situations that can be encountered by a reachability routing algorithm. Second, it is very hard for deterministic algorithms to achieve true multi-path routing on such topologies without encountering a combinatorial explosion in state. Finally, existing RL approaches to multi-path routing perform poorly on these topologies, thus discriminating the benefits of our approach. It should be stressed that our approach is generalized and works for a wide variety of topologies, presenting the greatest benefits in topologies that involve significant decision making.

In Figure 4.1, the two paths (path with and without potential loops) from node 0 to node 19 have a 1:25 cost ratio in favor of the path with potential loops, while in Figure 4.2 the cost ratio is 1: 2.5. As the results show, both graphs demonstrate a marked change in the routing probabilities at switchover time (1/8 of the simulation time shown as 0.13 on the 'normalized time' axis). The effect in Figure 4.1 is to further drive the probabilities away from each other, from the uniform ants estimate of 55% versus 45% to the model-based assessment of 96% versus 4%. The latter percentages very nearly reflect the cost ratio of 1:25.

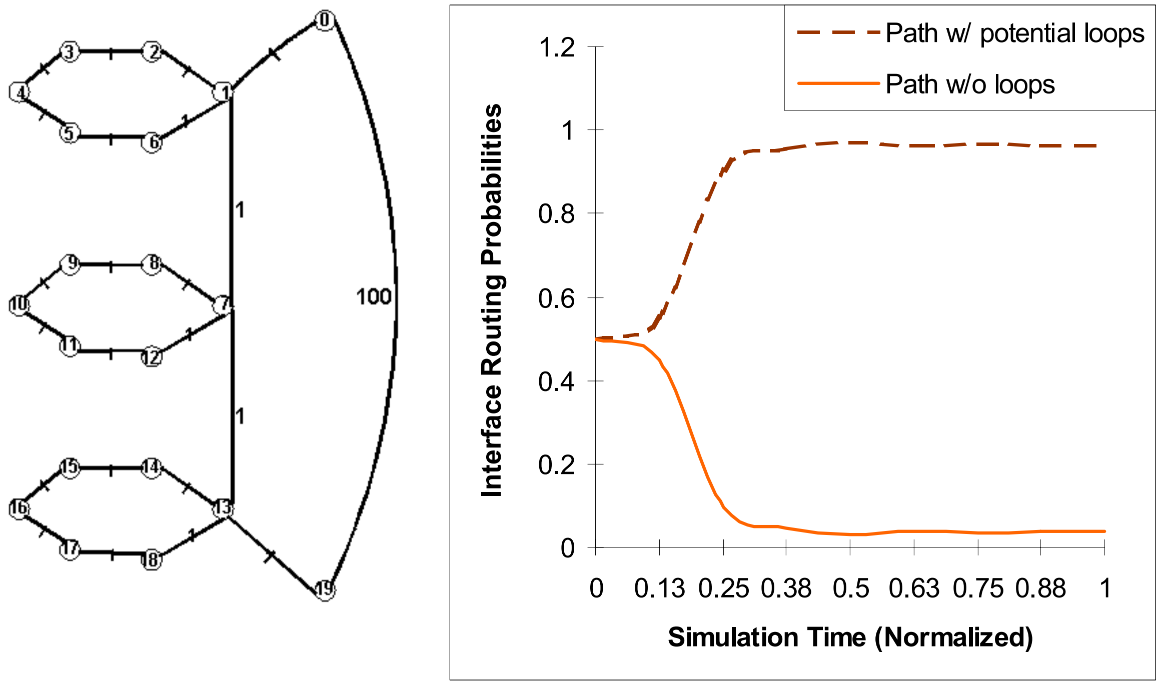
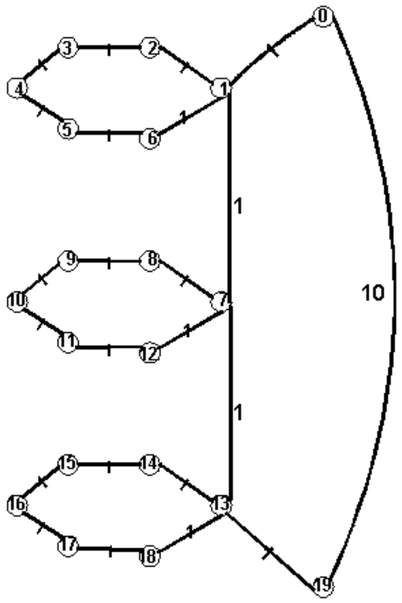


Figure 4.1 Velcro topology with cost ratio 1:25 & results of model-based routing.

Figure 4.2 clearly demonstrates the effectiveness of our model-based approach for cost-sensitive reachability routing. Recall from our earlier discussion that the uniform ants approach chooses the *higher cost* path without loops since it involves fewer decisions. In our model-based approach, node 0 begins by assigning a probability of 0.5 to each of the two links leading to node 19. Initially, the uniform ants approach tends to reinforce the higher cost loop-free path. After our model goes into effect, we observe a dramatic flip in the routing probabilities, which then converges to the ratio of the path costs.



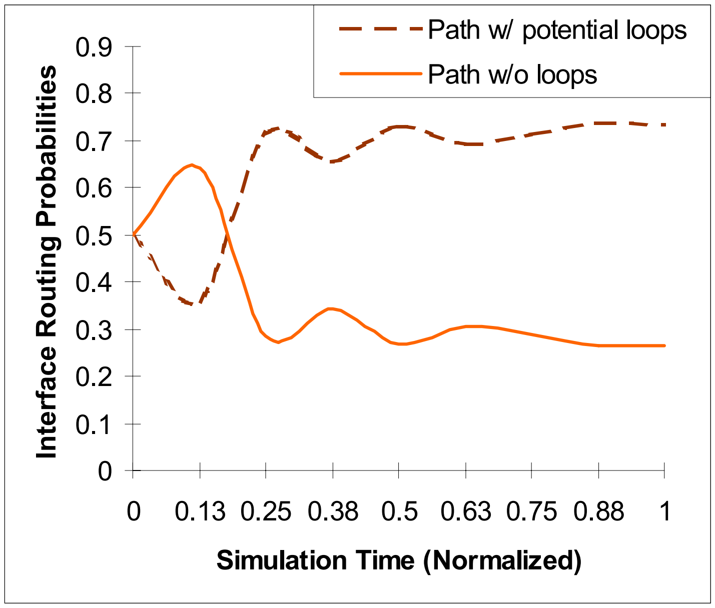
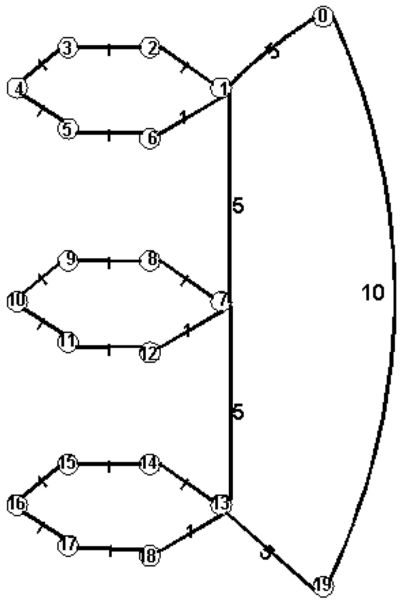


Figure 4.2 Velcro topology with cost ratio 1:2.5 & results of model-based routing.



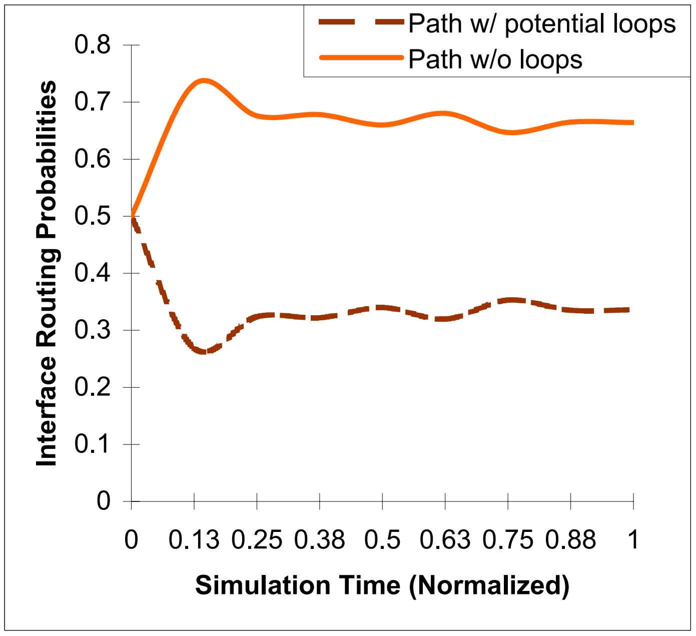


Figure 4.3 Velcro topology with cost ratio 2:1 & results of model-based routing.

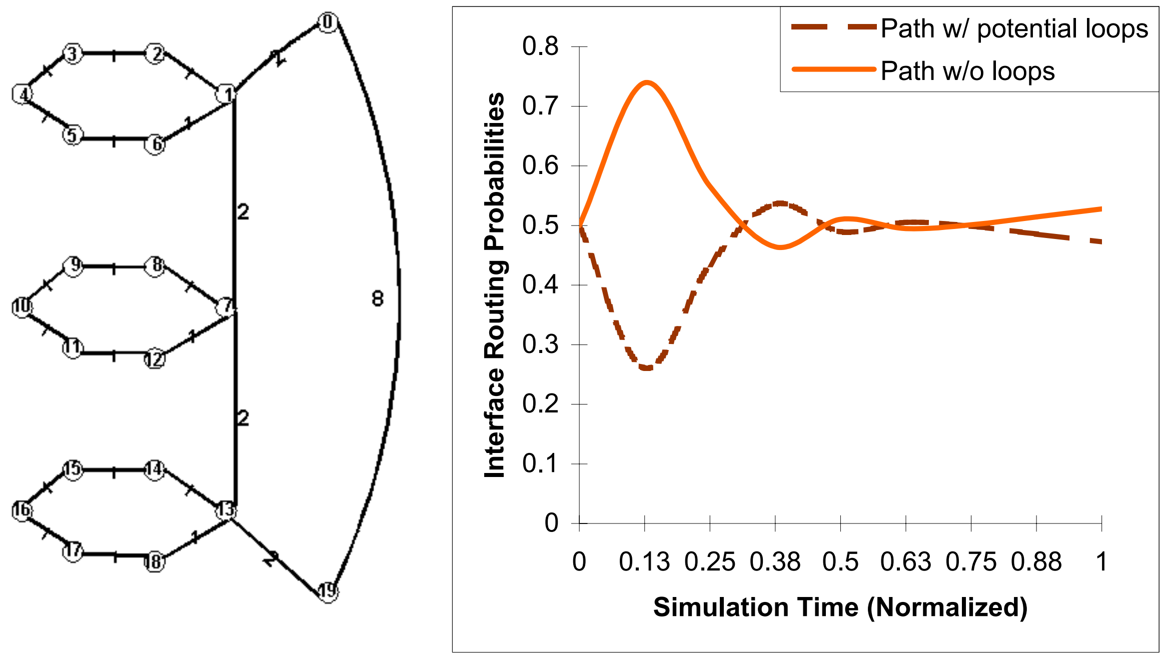


Figure 4.4 Velcro topology with cost ratio 1:1 & results of model-based routing.

In Figure 4.3, the two paths from node 0 to 19 have a 2:1 cost ratio, if the loops in the left path are avoided. As the corresponding graph shows, the uniform ants initially prefer the loop-free path by a ratio of 3:1. When the model is employed, this ratio gets moderated to 2:1, which more accurately reflects the cost ratios of the two paths.

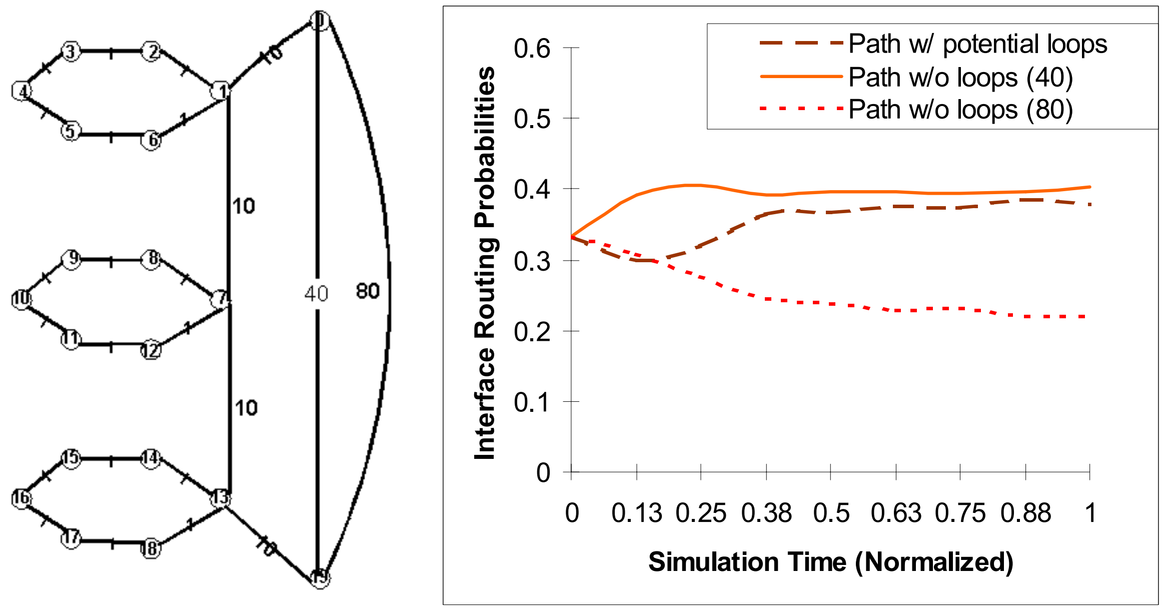


Figure 4.5 Velcro topology with 3 paths between nodes 0 and 19.

Figure 4.4 shows a topology similar to what we have considered so far, except that both the path with potential loops and loop-free paths have the same cost. As the results show, use of the statistics table causes both probabilities to converge to near equal values. Figure 4.5 drives home the point by introducing a third path between nodes 0 and 19 and our model-based approach once again learns to apportion equal probability among the loopy and the middle paths. As indicated by the costs, we obtain a 2:2:1 ratio of choosing among all three paths.

Finally, Figure 4.6 shows the operation of our algorithm on a topology where there are loops involving the fulcrums, in addition to loops rooted at the fulcrums. This is an example where we want loop resolution at one level, while retaining some element of the loops at another level (to achieve multi-path routing). All arrows in Fig. 16 depict interface probabilities for routing to destination node 19, from various nodes. Different colors indicate different choices of starting nodes and the thickness of arrows indicate greater probabilities along those interfaces. To understand the results, let us look at node 1 which has two paths of equal cost (and equal hops) to the destination. Nevertheless, the steady state routing probabilities reflect a preference to use the interface leading to node 7 over the one leading to node 13. This is because our algorithm tends to choose paths that have higher probability of reaching the destination, factoring all the possibilities for entering loops and absorption. As a simple recurrence calculation will show, node 7 is better than node 13 in terms of probability of reaching 19.

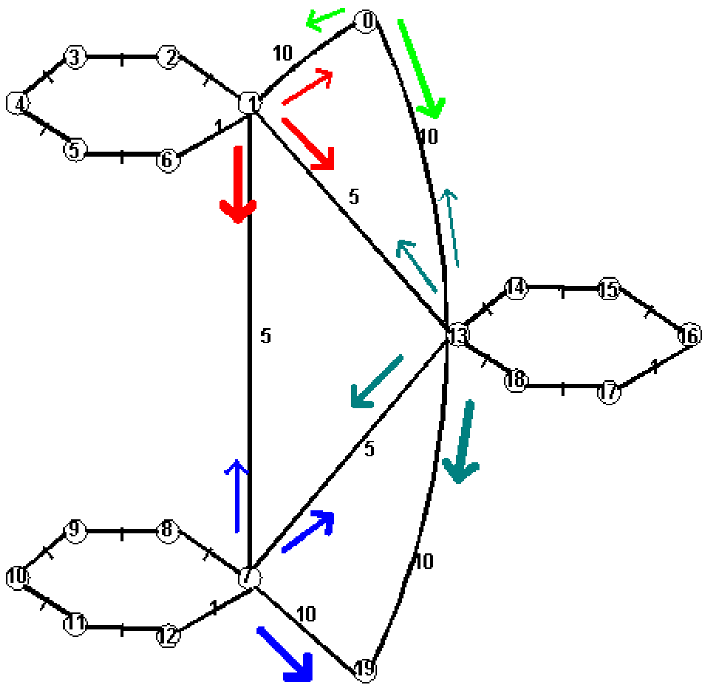


Figure 4.6 Results of model-based routing superimposed on velcro topology.

4.3.1 Caveats

The basic ants algorithm given in [22] reinforces all sub-paths along the path taken by an ant, and this can cause some nodes to experience greater reinforcements simply because they present interfaces to more destinations than other nodes.

Since model-based routing builds upon this algorithm, we inherit this property. For some topologies, this might cause slow convergence or a convergence that is not in accordance with the cost ratios. One solution to this problem is to conduct the reinforcement updates at a node only if that node was the intended destination of the ant. Arguably this goes against classical reinforcement learning algorithms but this consideration is echoed by many other researchers as important for practical deployment. For instance [8] offers a different perspective on such 'selective' sub-path reinforcement.

To test this idea, the model based routing algorithm was modified to prevent sub-path reinforcement from taking place. A *'dumbbell'* topology (Figure 4.7), where the direct path from node 5 to node 4 is costlier than the round about path, was the test topology. Using sub-path reinforcement (see Figure 4.8) does not capture the desired effect, whereas avoiding subpath (see Figure 4.9) learns the correct apportionment of probabilities.

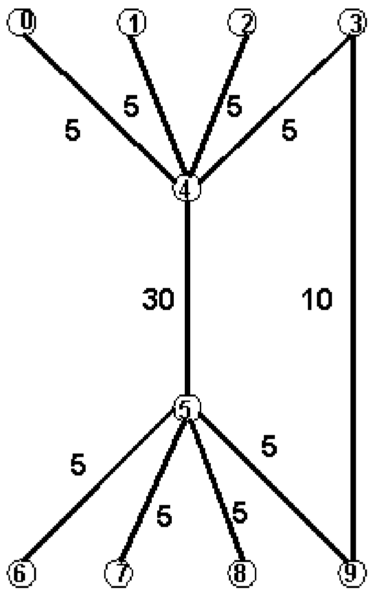


Figure 4.7 A 'dumbbell' shaped topology.

However since we do not have a priori knowledge of the topology, we continue to employ sub-path reinforcement for the remainder of this thesis. This caveat must however be kept in mind.

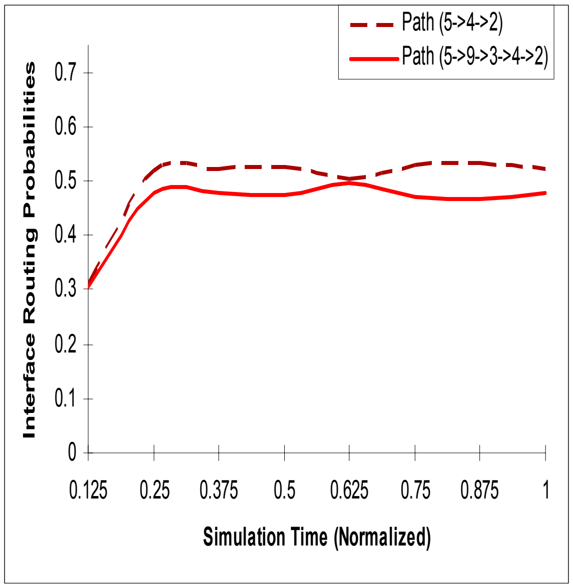


Figure 4.8 Routing table probabilities for node 5 to destination 2 with sub-path reinforcement.

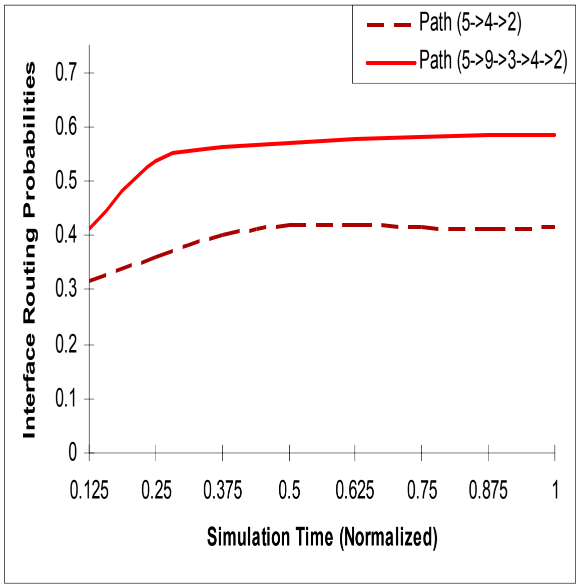


Figure 4.9 Routing table probabilities for node 5 to destination 2 with no sub-path reinforcement.

4.4 Convergence of model-based routing algorithm

To assess convergence characteristics, we employ linear correlation metrics. The *Pearson Product-Moment Correlation Coefficient* (r), or correlation coefficient for short, is a measure of the degree of linear relationship between two variables, usually labeled X and Y. While in regression the emphasis is on predicting one variable from the other, in correlation the emphasis is on the degree to which a linear model may describe the relationship between two variables. In regression the interest is directional, one variable is predicted and the other is the predictor; in correlation the interest is non-directional, and the relationship is the critical aspect [21].

The range of the correlation coefficient (r) is between –1 and 1. A positive correlation coefficient means that as the value of one variable increases, the value of the other variable increases; as one decreases the other decreases. A negative correlation coefficient indicates that as one variable increases, the other decreases, and vice-versa.

The correlation coefficient is the slope of the regression line when both the X and Y variables have been converted to z-scores. The z-scores are calculated as follows:

$$z_X = \frac{X-\overline{X}}{S_X}, z_Y = \frac{Y-\overline{Y}}{S_Y} \quad where\ \overline{X}\ and\ \overline{Y} : Mean\ of\ X\ and\ Y, S_X\ and\ S_Y : SD\ of\ X\ and\ Y$$

If *n* is the number of elements in X and Y, then the correlation coefficient ® is:

$$r = \frac{\sum_{i=1}^{n} z_X z_Y}{n-1}$$

The following two experiments were performed on 12 different BRITE [14] topologies with the number of nodes ranging from 20 to 100. In both the experiments, the nodes of interest were the minimum and maximum degree nodes. The first experiment dealt with the costs associated with the transmission of packets across the network while the second dealt with the hop count associated with the packets. Below we explain the experiments in detail.

4.4.1 Path Costs

In this experiment, we measure the average end-to-end path costs of the packets originating from and destined to the node under consideration. At the end of the simulation, we obtain two vectors, one with the average end-to-end path costs for packets from the node of interest to every other node in the topology and another with average end-to-end costs from every other node to the node of interest. Using the method mentioned in the previous section, we found the correlation coefficient between these two vectors. The average correlation for the minimum degree node across all topologies was found to be 0.996, with a standard deviation of 0.0026. Similarly, for the maximum degree nodes the average correlation was found to be 0.98, with a standard deviation of 0.012. For every topology, the minimum degree node has a higher correlation than the maximum degree node, which is due to the fact that the maximum degree node has more choice of interfaces to choose from, to the various destinations. Assuming symmetry of link costs, a high correlation indicates that by using our model-based routing algorithm the nodes exhibit consistency in learning similar paths to every other node in the topology.

From Figure 4.10 it can be seen that the end-to-end path cost correlation associated with the maximum degree node starts at a very low value (0.7) and as the simulation progresses with more and more ants exploring the network, the correlation value stabilizes to a value of 0.96 confirming the convergence of our model-based routing algorithm.

4.4.2 Hop Count

This experiment is very similar to the previous one except that instead of measuring the end-to-end path costs, we determined the average hop count for all packets originating from or destined to the node under consideration. In this case a high correlation indicates that the packets follow paths with nearly equal hop-counts in both directions. The average correlation for the minimum degree node was found to be 0.996 with a standard deviation of 0.0016 while for the maximum degree node it was found to be 0.981 with a standard deviation 0.012.

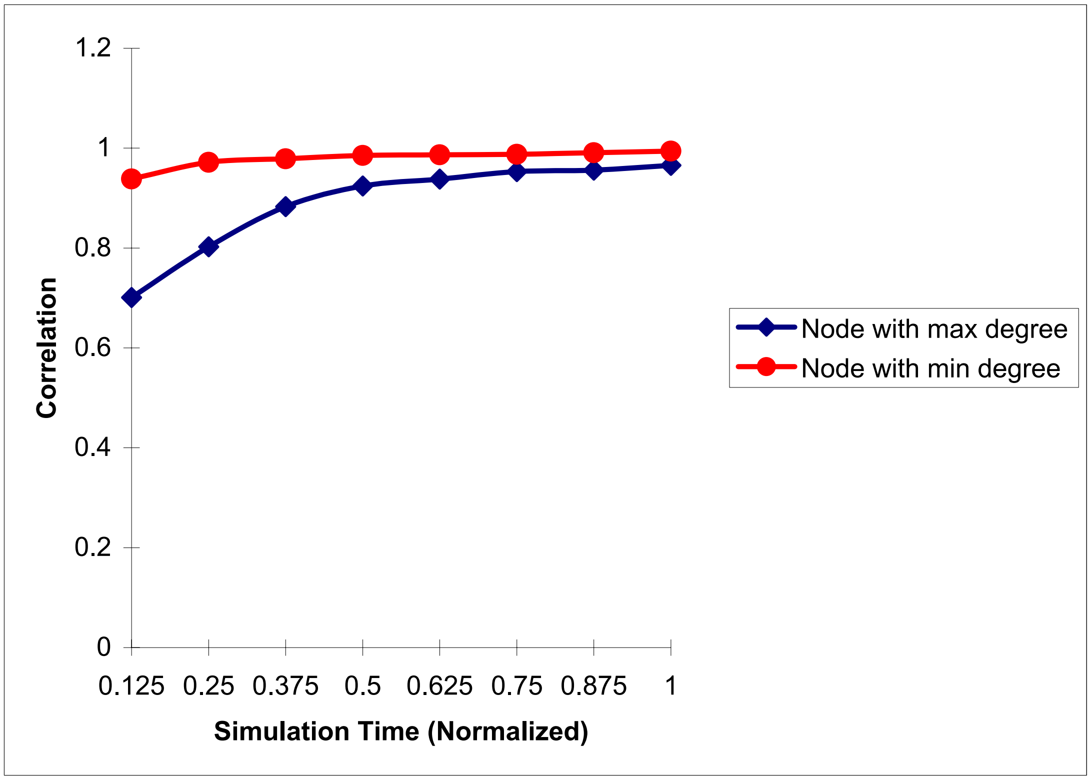


Figure 4.10 Convergence of end-to-end path cost correlation for a 60-node BRITE topology.

4.5 Packet Routing based on model based routing and uniform ants routing

In this set of experiments, a new application was written on top of the simulator to route packets based on the routing table learnt by ants exploring the network. Initially, we ran the model-based routing algorithm on the given topology to obtain a stabilized routing table. Next, we ran the application with the routing table and φ (reachability factor) as parameters and collected various statistics. Below we will discuss in detail the application, the reachability factor (φ), the statistics collected, and analysis of the statistics obtained from both model-based and uniform ants routing.

The functioning of the application is similar to the one described in Chapter 3 [Table 3-1] except that there is no update of the routing table. The routing table is pre-initialized to that obtained from the model-based routing simulation and remains constant throughout. By not updating the routing table based on the packets arriving at every node we are just *exploiting* the model and not *exploring* the network further.

The reachability factor φ controls the degree of freedom every node has in choosing the outgoing interface. At every node the outgoing interfaces were ordered in descending order of their probabilities for every destination. When a node *n* needs to route a packet intended for destination *d*, it picks the top φ interfaces for that destination *d* and uses their scaled up probabilities for selecting the outgoing interface. For a better understanding of the reachability factor, consider the following example. Say a node M has 4 interfaces A, B, C, and D with associated probabilities 0.4, 0.2, 0.15, 0.15 for destination N; then a φ value of 2 will allow the node M to choose from interfaces A and B with probabilities (0.4)/(0.4 + 0.2) and (0.2)/(0.4 + 0.2) respectively i.e. node M will choose interface A 66.67% of the time and interface 33.33% of the time to route the packet intended for destination N.

The statistics collected include the number of loops encountered by the packets along their paths, the number of packets encountering loops, the multipath capability of the packets, and the percentage of packets successfully reaching their intended destination. To determine the number of loops encountered by the packets, every packet has a stack associated with it. Every node, before forwarding a packet, checks to see if its *id* already exists in the stack. If its *id* is present in the stack, it increments the loop counter of the packet by 1 and pops the contents of the stack up to its *id* else pushes its *id* onto the stack and then forwards the packet. At the end of the simulation we have statistics on the number of packets encountering loops (*loop percentage*) and the total number of loops encountered by all the packets. Every packet also has a multipath flag associated with it that is set if any node along the path taken by the packet has more than one outgoing interface to choose from. This is used to determine the percentage of packets that could have potentially taken more than one path to reach their intended destination (*multipath percentage*). Finally, we determine the *success percentage* as the percentage of packets successfully reaching their intended destination.

4.5.1 Reachability factor φ =1

In our first set of experiments φ was set to 1 so that the nodes always choose the best outgoing interface (interface with the highest probability) for every packet. As every packet deterministically chooses the best interface at every node, the multipath percentage is zero. A φ value of 1 also results in the avoidance of loops and a one hundred percent success percentage as all the packets reach their intended destination. According to proposition 2 of Subramanian et al [22], the probability of choosing an interface is inversely proportional to the cost ratios (under the assumption of loop free paths), keep in mind that this proposition applies even for our modified model-based algorithm as all the avoidable loops are avoided and also we have shown from Section 4.3 that the probabilities are inversely proportional to the path costs. By choosing the interface with the highest probability, i.e. the interface that advertised a lower cost path to that destination, at every node we have achieved deterministic shortest path routing while still using the underlying probabilistic routing table. The following set of simulations were done on 20 to 100 node BRITE topologies with uniform cost distribution so that with φ = 1 the path taken by all the packets will not only correspond to the shortest path in terms of cost but also in terms of the number of hops. By sending packets across the network and keeping track of their hop count, we ascertained the shortest path length between every source-destination pair. At the end of the simulations, the average shortest path length for the topologies were calculated and compared with the theoretical shortest path lengths. We then attempt to fit this empirical data onto parameterized formaulas.

Below we discuss the derivation of average shortest-path lengths for *exponentially distributed graphs* based on [17]. The Router Waxman model of BRITE uses an exponentially distributed generation function to create the topologies. According to [17], the generating function $G_0(x)$ should be normalized such that $G_0(1) = 1$.

We use the following generating function [17] for our derivation:

$$G_0(x) = \frac{1 - e^{-1/\kappa}}{1 - xe^{-1/\kappa}}$$

According to [17], the average shortest path length is given by

$$l = \frac{\ln(N / z_1)}{\ln(z_2 / z_1)} + 1, \; \textit{for } N >> z_1 \textit{ and } z_2 >> z_1$$

where N corresponds to the number of nodes in the topology, and z_m corresponds to the average number of mth-nearest neighbors where $z_1 = G_0'(1)$ and $z_2 = G_0''(1)$. We derived l to be

$$l = 1 + \frac{\ln N + \ln(e^{1/\kappa} - 1)}{\ln 2 - \ln(e^{1/\kappa} - 1)}$$

From this equation we derived the value of κ to be

$$\kappa = \frac{1}{\ln[1 + \frac{2^{(l-1)/l}}{N^{1/l}}]}$$

Based on the above derivations, a least square fit was conducted on the simulation results, which returns both κ and the square of the correlation coefficient with values ranging of 0 and 1, indicating bad or good fit respectively. In our case, the fit returned a value of 0.986551, which indicates that the best fit line summarizes the data very well as shown in Figure 4.11.

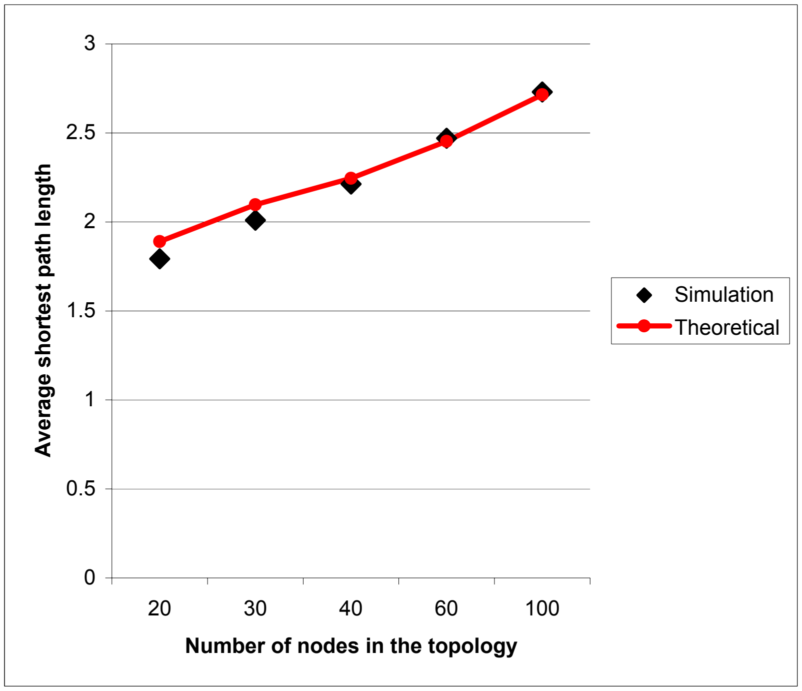


Figure 4.11 Least square fit between the theoretical and actual shortest path length.

4.5.2 Reachability factor φ = maximum degree

By setting the reachability factor **φ** to the maximum degree of the topology, every node will be allowed to choose among all its interfaces to be the outgoing interfaces (based on the probability associated with it for the intended destination). The simulations were run on the following topologies: 20 to 200 node BRITE topologies, 10X4 & 8X5 mesh topologies and the velcro topologies described in 4.3. Operating curves between percentage of packets encountering loops and percentage of those with multipath capabilities were plotted for various topologies at different values of threshold factor (**τ**).

As opposed to **φ** = 1, **φ** = maximum degree results in multi-path forwarding of the packets and also some proportion of packets entering into loops. All the packets reached their intended destinations except for those that looped back to their source resulting in a high success percentage. To overcome the drop in success percentage, the packets were forwarded even when they looped back to the source and counting this episode as just another loop encountered along the path. With this modification all the packets successfully reached their intended destinations but with a linear increase in the percentage of loops (to account for all those packets that were earlier absorbed by their source). All packets had a TTL of 255 and none of them were dropped due to reaching the TTL limit. Below we present the operating curves for various topologies under both the cases: 1) absorption of packets at their source and 2) no absorption of packets. The next several pages summarize these operating curves for all the topologies, before making observations.

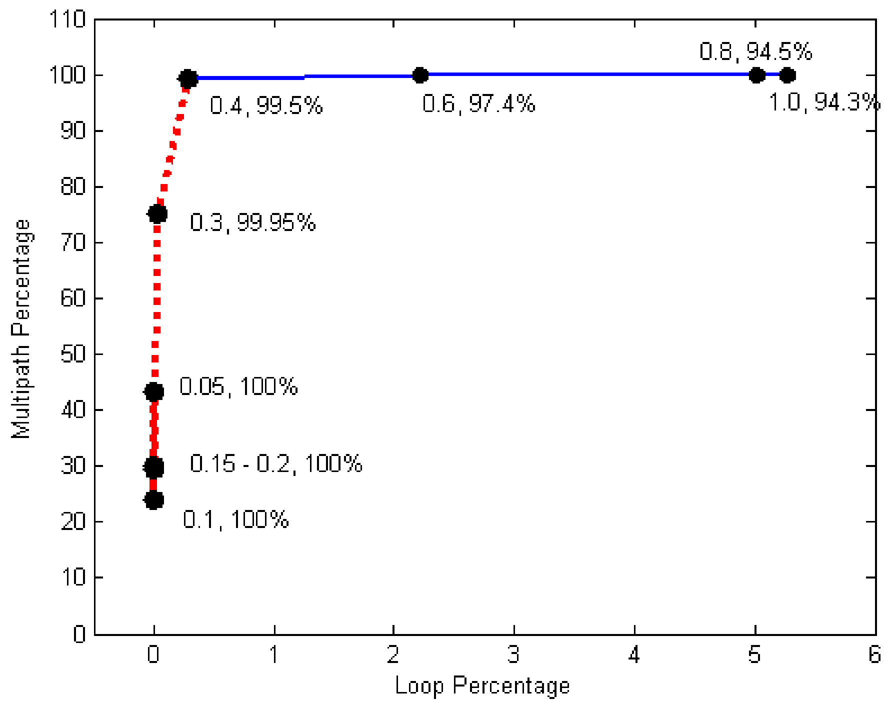


Figure 4.12 Operating curve for a 20 node BRITE topology with source absorption. Each point is labeled with its τ value and success percentage.

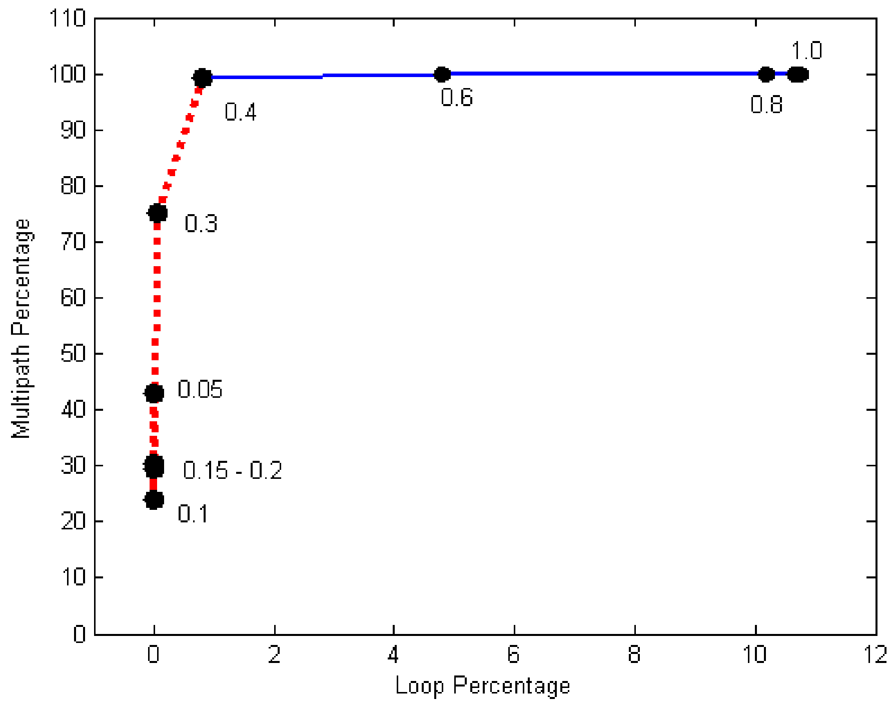


Figure 4.13 Operating curve for a 20 node BRITE topology with no source absorption. Each point is labeled with its τ value.

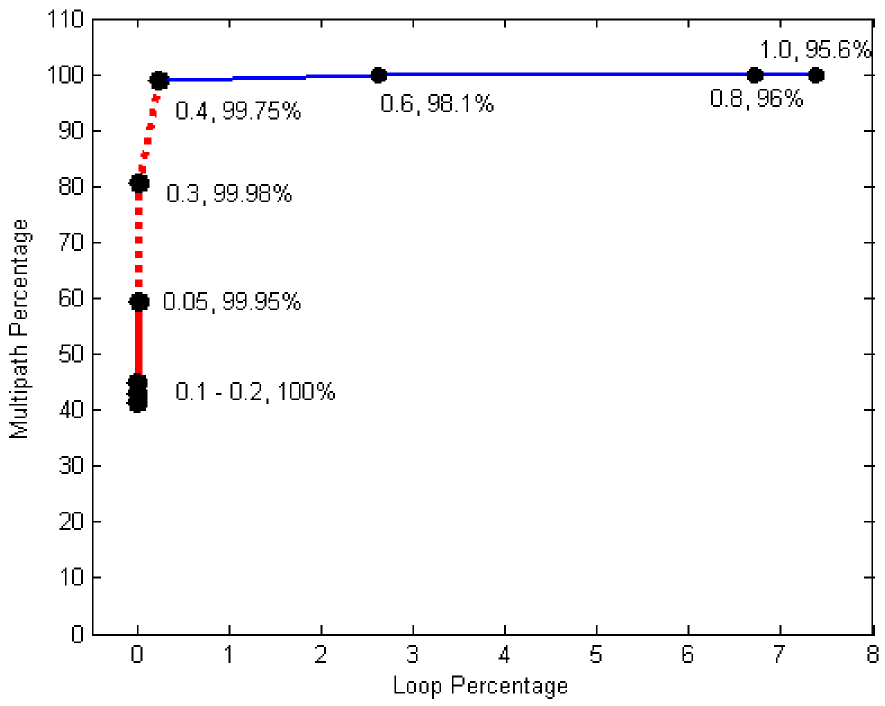


Figure 4.14 Operating curve for a 40 node BRITE topology with source absorption. Each point is labeled with its τ value and success percentage.

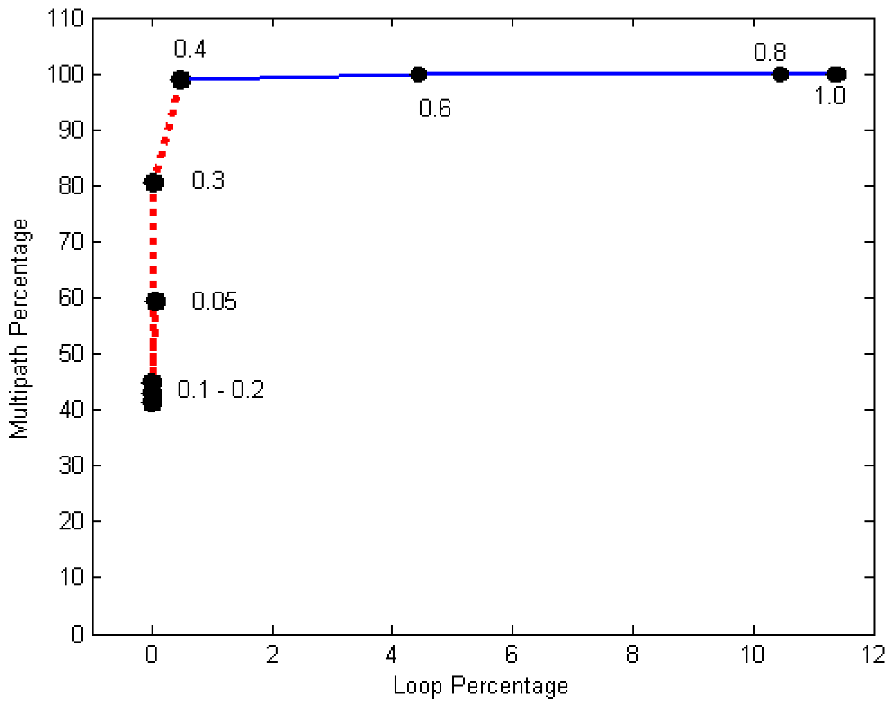


Figure 4.15 Operating curve for a 40 node BRITE topology with no source absorption. Each point is labeled with its τ value.

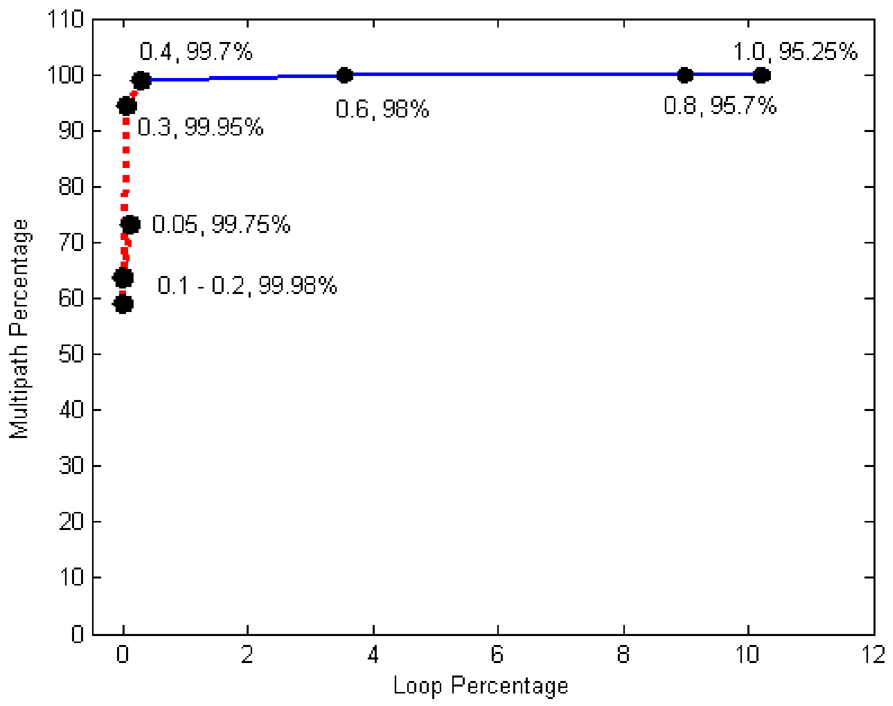


Figure 4.16 Operating curve for a 60 node BRITE topology with source absorption. Each point is labeled with its τ value and success percentage.

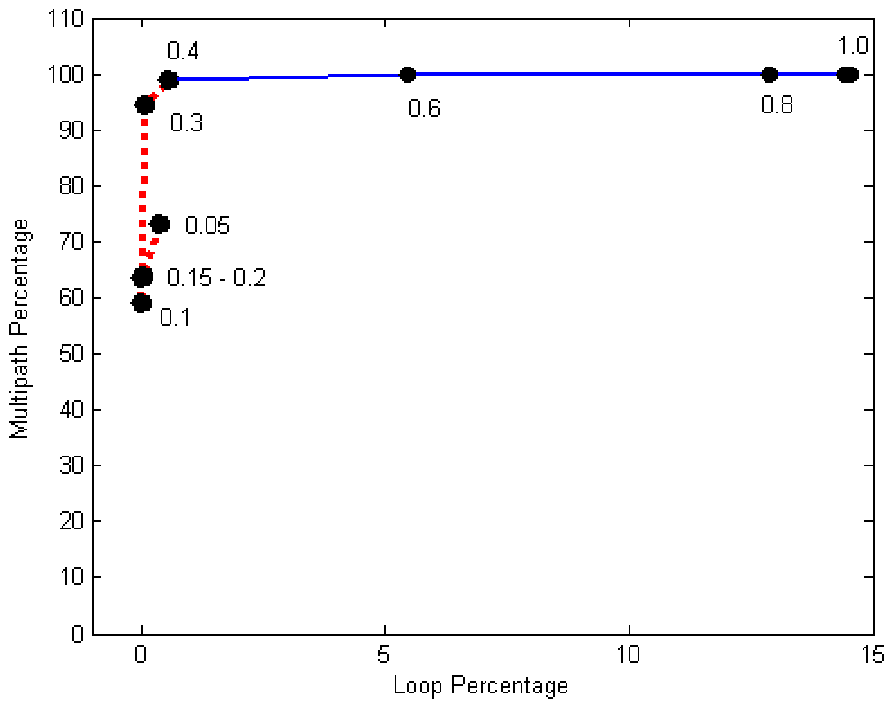


Figure 4.17 Operating curve for a 60 node BRITE topology with no source absorption. Each point is labeled with its τ value.

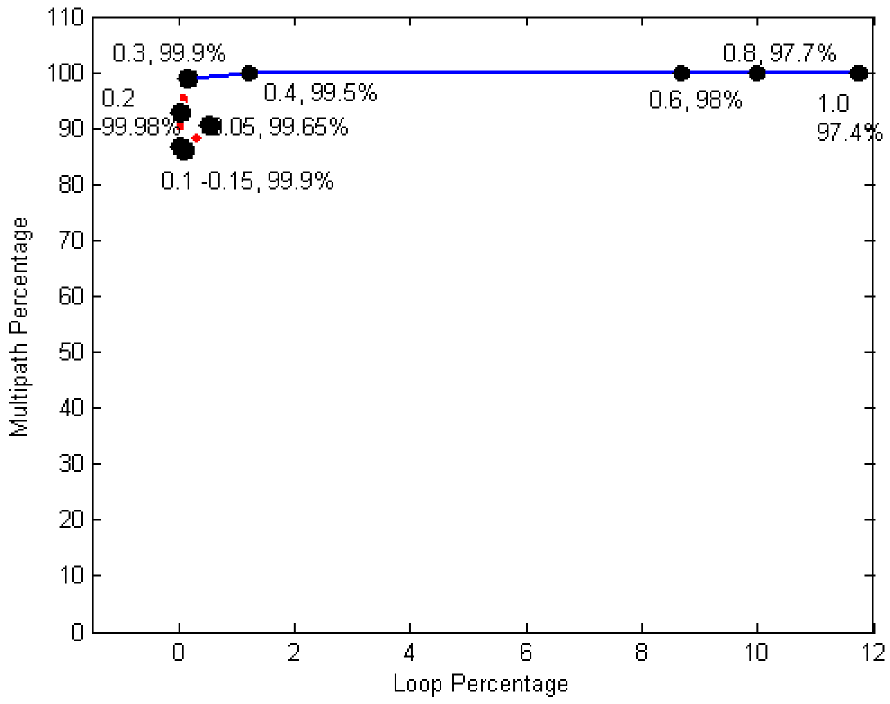


Figure 4.18 Operating curve for a 200 node BRITE topology with source absorption. Each point is labeled with its τ value and success percentage.

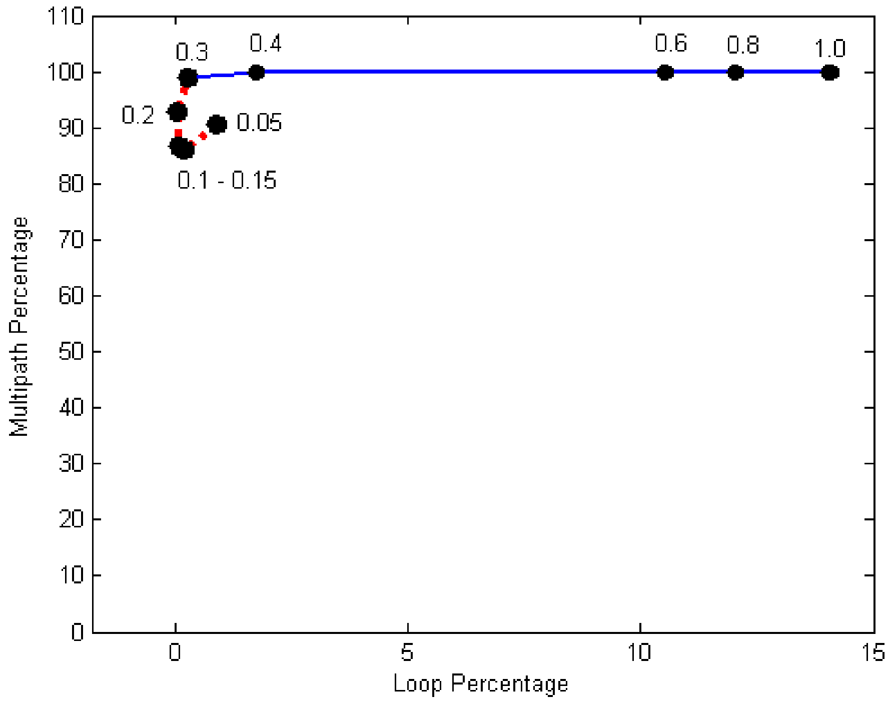


Figure 4.19 Operating curve for a 200 node BRITE topology with no source absorption. Each point is labeled with its τ value.

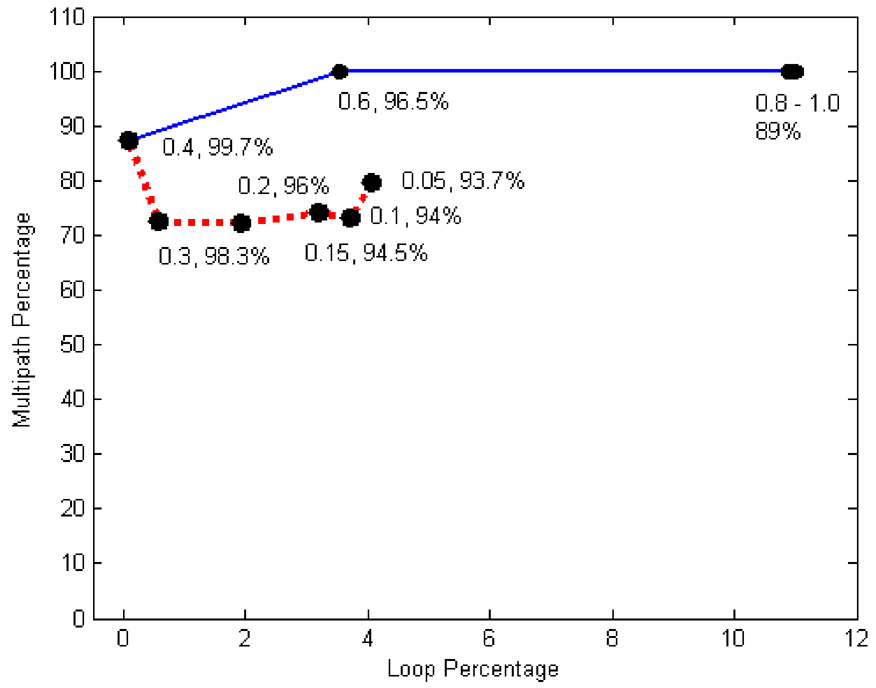


Figure 4.20 Operating curve for an 8X5 mesh topology with source absorption. Each point is labeled with its τ value and success percentage.

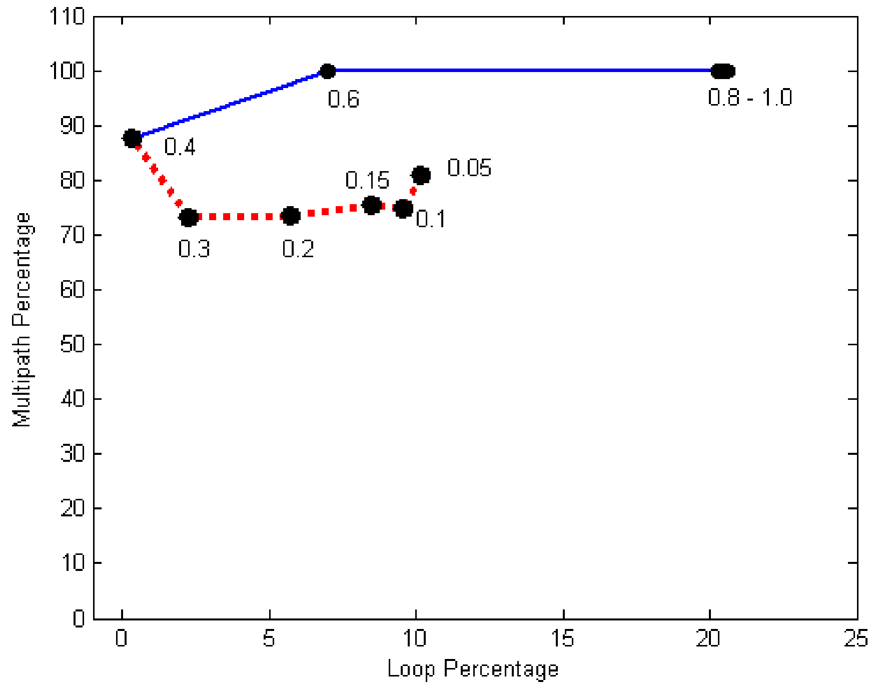


Figure 4.21 Operating curve for an 8X5 mesh topology with no source absorption. Each point is labeled with its τ value.

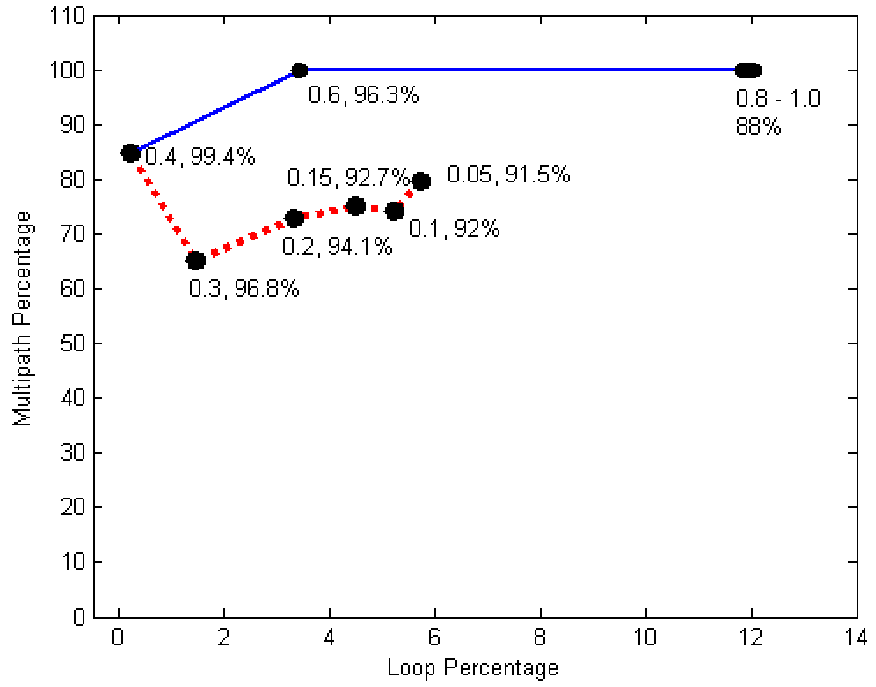


Figure 4.22 Operating curve for a 10X4 mesh topology with source absorption. Each point is labeled with its τ value and success percentage.

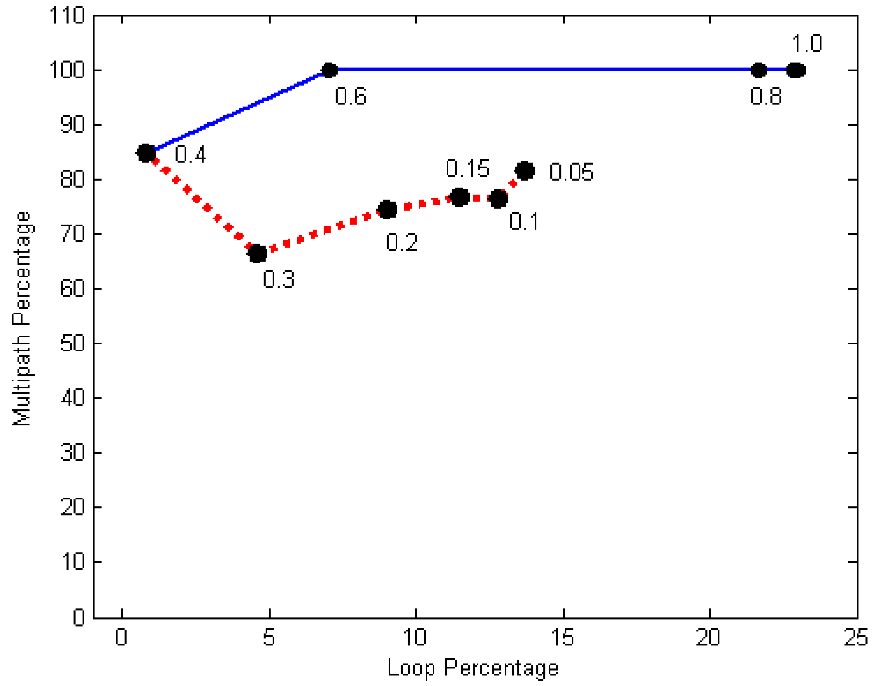


Figure 4.23 Operating curve for a 10X4 mesh topology with no source absorption. Each point is labeled with its τ value.

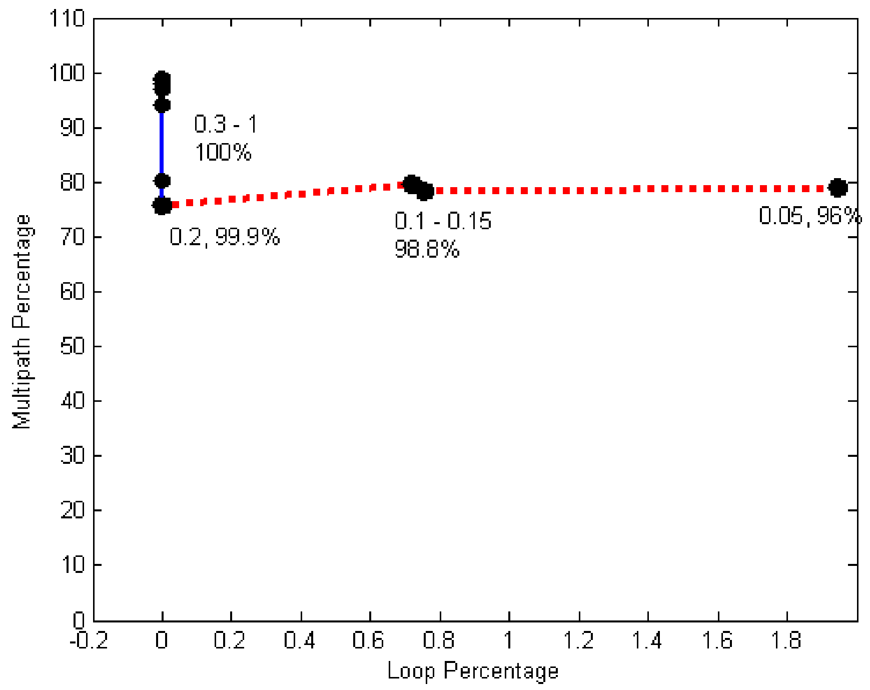


Figure 4.24 Operating curve for a velcro topology of Figure 4.1 with source absorption. Each point is labeled with its τ value and success percentage.

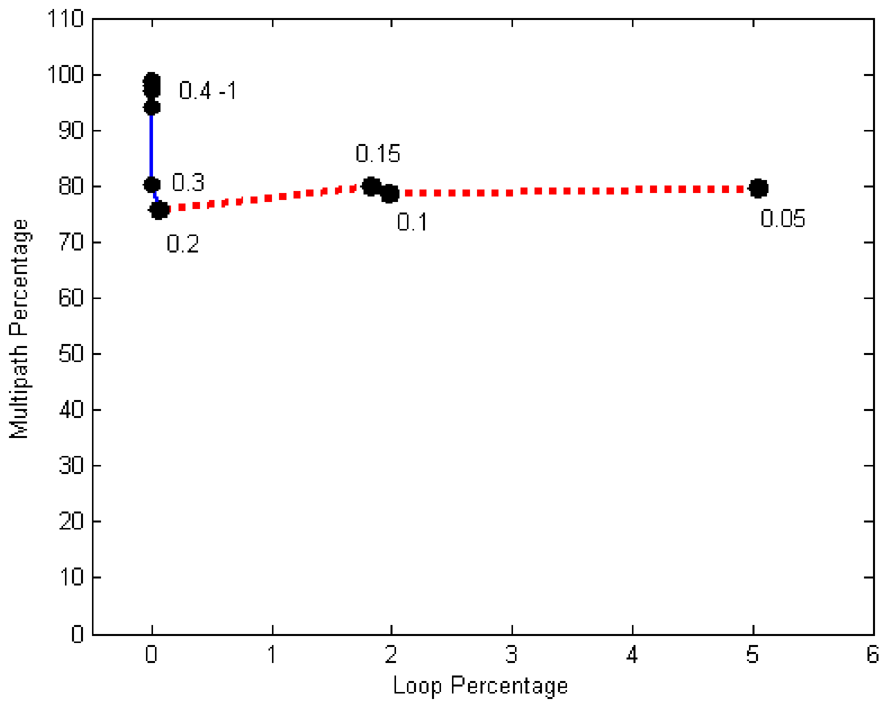


Figure 4.25 Operating curve for a velcro topology of Figure 4.1 with no source absorption. Each point is labeled with its τ value.

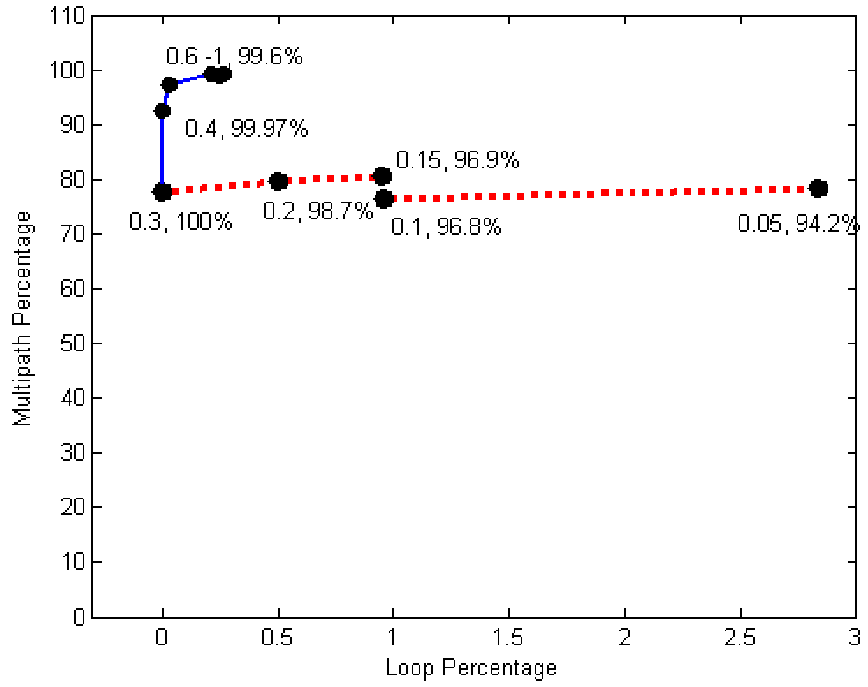


Figure 4.26 Operating curve for a velcro topology of Figure 4.5 with source absorption. Each point is labeled with its τ value and success percentage.

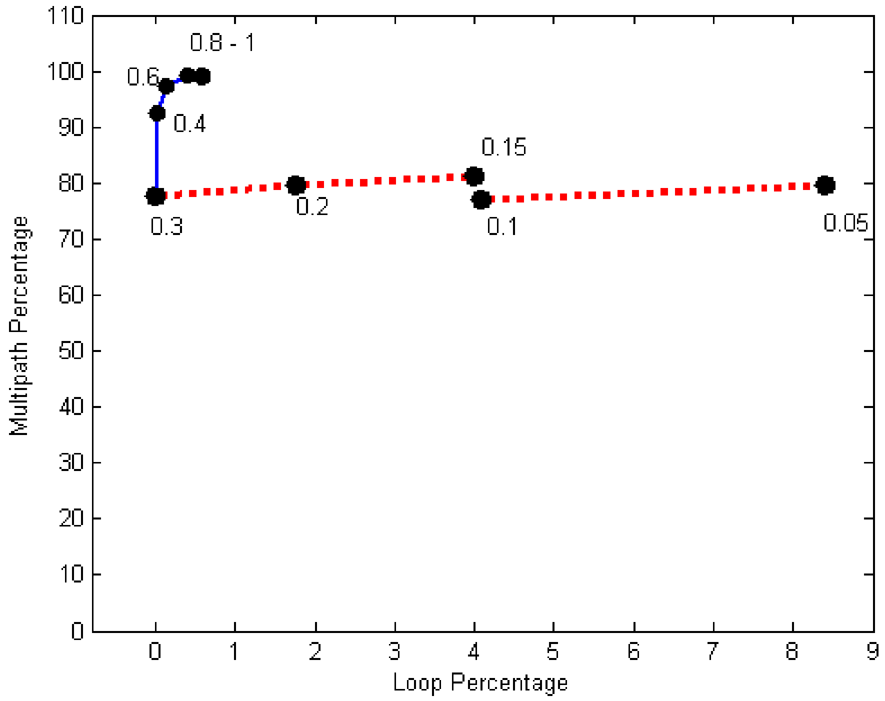


Figure 4.27 Operating curve for a velcro topology of Figure 4.5 with no source absorption. Each point is labeled with its τ value.

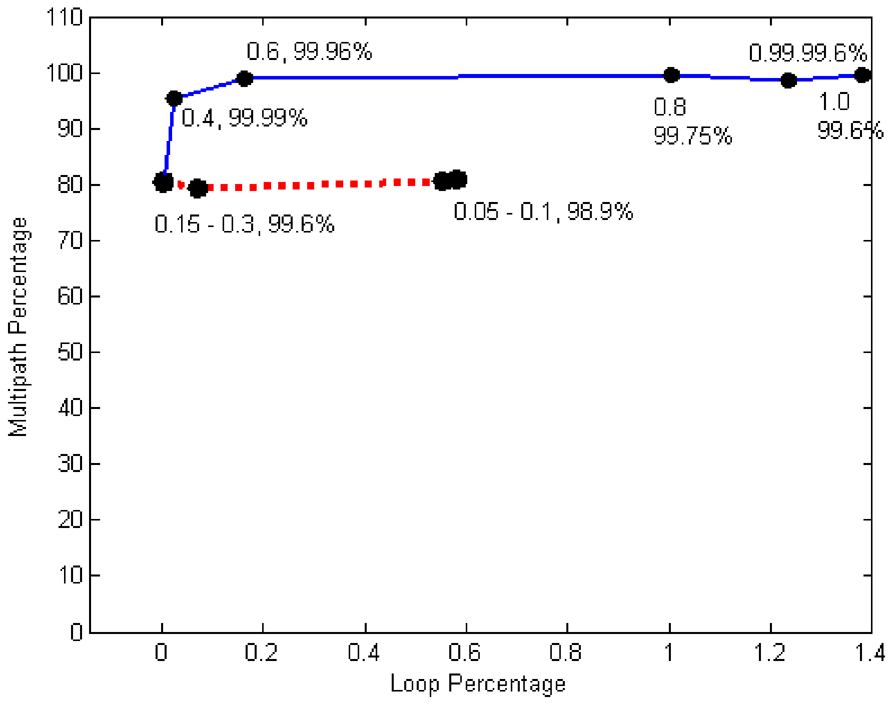


Figure 4.28 Operating curve for a velcro topology of Figure 4.6 with source absorption. Each point is labeled with its τ value and success percentage.

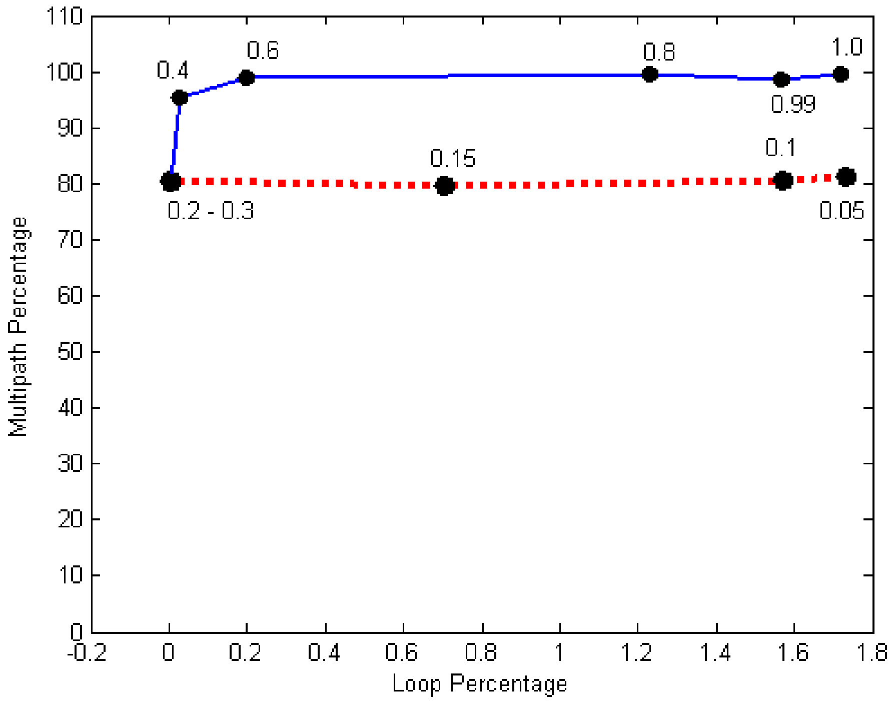


Figure 4.29 Operating curve for a velcro topology of Figure 4.6 with no source absorption. Each point is labeled with its τ value.

4.5.2.1 Observations

Let us take Figure 4.12 and study it closely. As threshold factor (**τ**) increases, we see that we go from a region with (no loops, 20% multipath) to (5% loops, 100% multipath). It is heartening to note that the curve first increases in the direction of accommodating multipath before introducing loops, rather than the other way around.

Second, notice that different portions of the graph are labeled/colored differently. Every operating curve is marked using two colors, a dotted red line and a solid blue line, to distinguish a region where the model is completely in force from one where it is not. As explained in Chapter 3, at very low values of the threshold factor (**τ**), when all the interfaces at an *intermediate* node are ineligible, i.e. their statistic table ratios are above the threshold **τ**, then the node *'sends-back'* the ant along the interface it originally received the ant from resulting in an increased percentage of packets entering into loops. Similarly at very low values of **τ**, when all the interfaces at the *source* node are ineligible then the source node uses the uncontrolled exploration selection policy to break the deadlock. As Figure 4.12 shows, around a **τ** value of 0.4, the model comes into force in that all routing decisions are based on learning rather than defaults.

On comparing Figure 4.12 with Figure 4.13 (the latter of which does not have source absorption), we notice that the difference in the percentage of success of packets reaching their destination with and without source absorption is reflected in the difference of percentage of packets encountering loops with and without source absorption.

The BRITE topologies (Figure 4.12 to Figure 4.19) also exhibit another interesting behavior. As the number of nodes in the topology increases, the minimum multipath percentage also increases. This is due to the fact that at very low threshold values, the model-based routing algorithm routes a large number of packets deterministically in smaller topologies.

The shape of the operating curve greatly depends on the intrinsic graph theoretic property of the topologies. The reader can observe from the figures that each class (BRITE, mesh and velcro) of topologies generates its own specific shape of operating curves. Figure 4.28 and Figure 4.29 have their operating curve very similar to the operating curve generated by the mesh topologies as the topology in Figure 4.6 can be viewed as a triangulated mesh topology.

The reader should observe that all the operating curves at **τ** = 1 exhibits the behavior of the uniform ants algorithm [22]. This is due to the fact that all the interfaces at every node are eligible to be the outgoing interface for the intended destination which conforms to the selection policy of uniform ants algorithm.

4.6 Distribution of loop frequency

Finally, we show that even though the presence of loops is unavoidable, the number of packets that encountered ‘k’ loops along their paths to their respective destinations exponentially decays with increase in ‘k’, i.e. the majority of the packets encounter between 0 to 2 loops, suggesting a power law. Below we show the plot between loop distribution and packet frequency for an 8X5 mesh and a 40-node BRITE topology. Due to the cyclic nature of the mesh, certain packets encounter as many as 20 loops before they reach their intended destination.

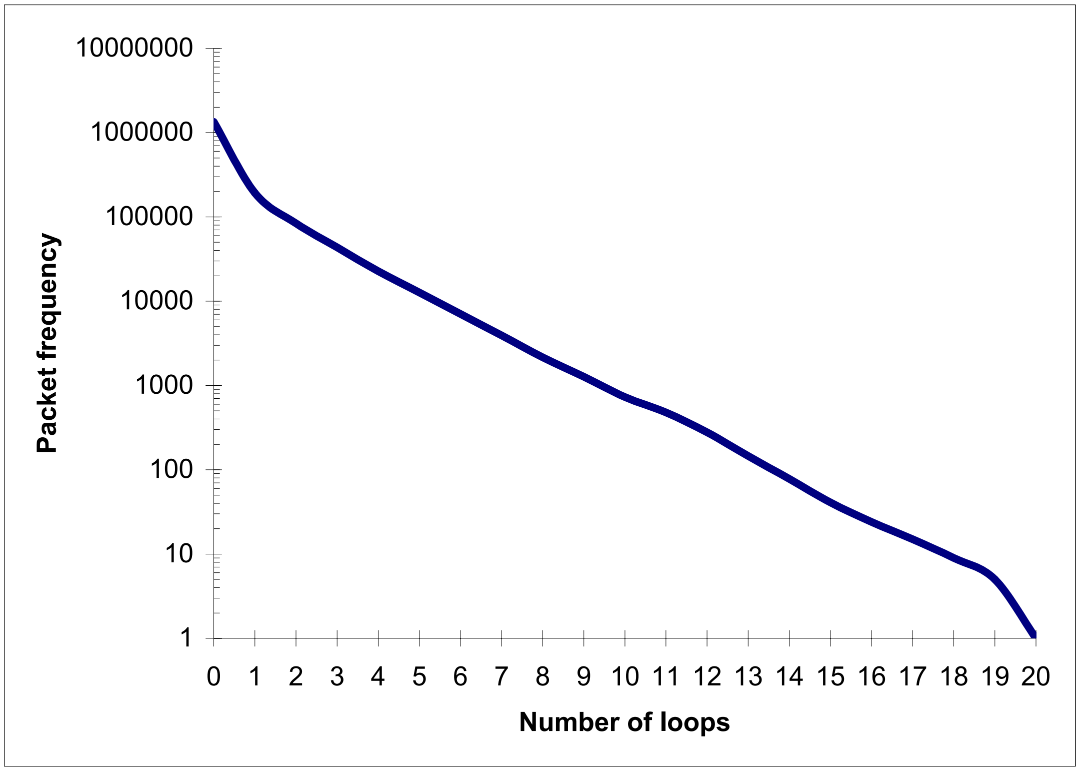


Figure 4.30 Distribution of loops encountered by packets in an 8X5 mesh. Notice the logarithmic scale of the y-axis.

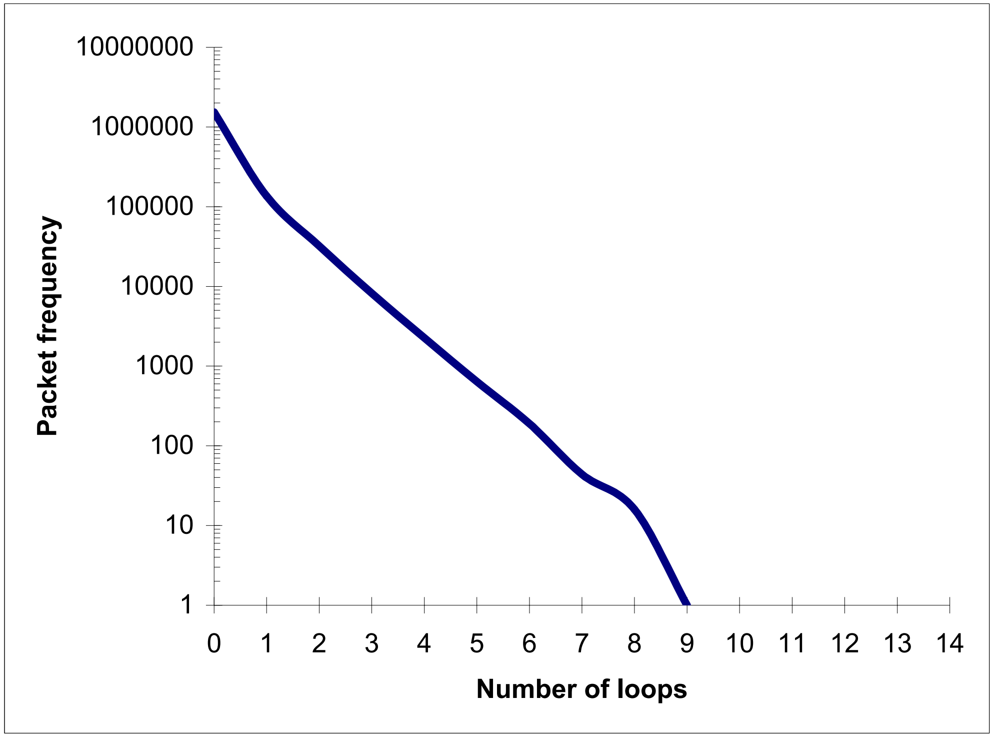


Figure 4.31 Distribution of loops encountered by packets in a 40-node BRITE topology. Notice the logarithmic scale of the y-axis.

4.7 Verification of cost-sensitive routing in BRITE topologies

The goal of this experiment was to perform a large-scale validation of the cost-sensitivity properties of our reachability routing algorithm. First, all the paths taken by various packets at φ = maximum degree were enumerated. To achieve this, every packet had a stack associated with it that kept track of the nodes visited by it *en route* to its destination. At the destination, the paths taken by the packets from each source were ranked in increasing order of their costs. The destination nodes only kept track of unique paths from each source and also maintained the frequency associated with each path. The purpose of this instrumentation was to ensure that the frequency of costs, as measured through pursued paths, mirrored the distribution of traffic along these paths.

At the end of the simulation, the summation of frequency over the top [x-9%, x%] of the paths for every source-destination pair was determined, for $x \in [10,20 \ldots 100]$. Figure 4.32 and Figure 4.33 clearly show the cost-sensitive routing of our model-based algorithm and also that sub-path reinforcement has no effect on BRITE topologies. (The experiments were performed on 60-node BRITE topologies).

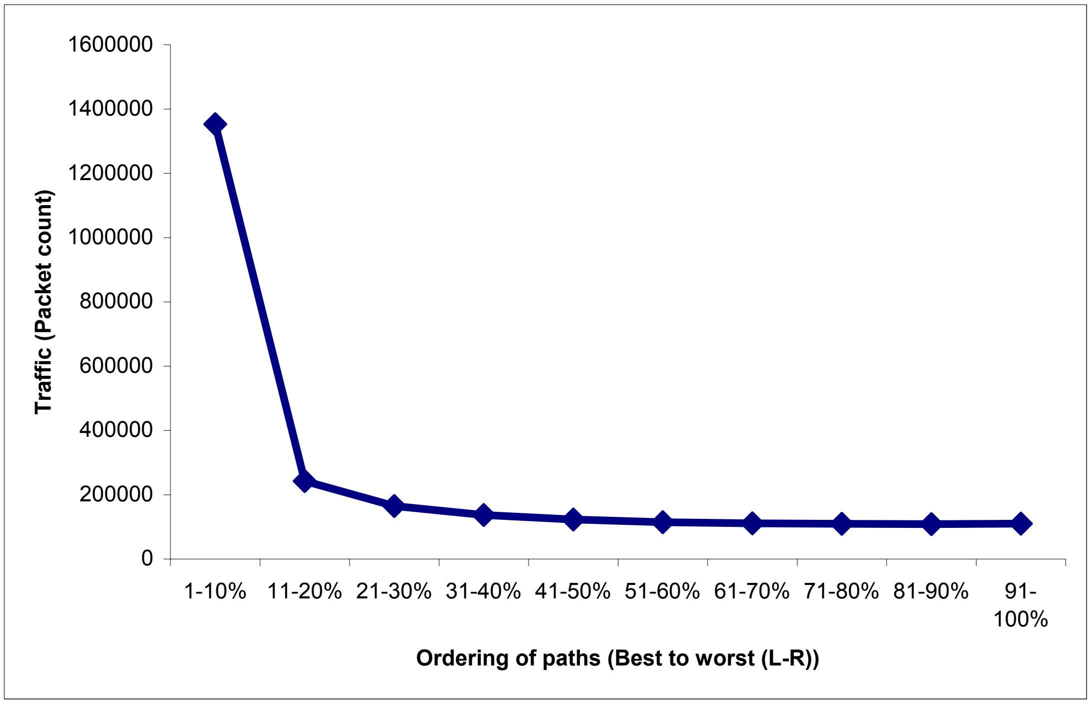


Figure 4.32 Traffic distribution in 60-node BRITE with sub-path reinforcement.

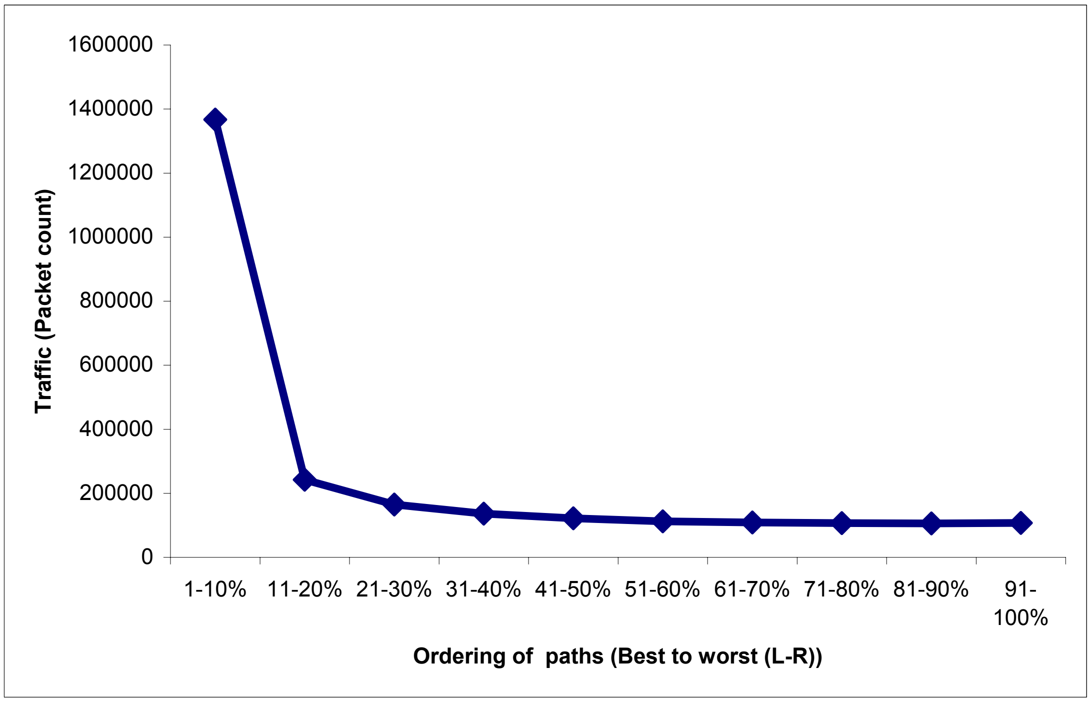


Figure 4.33 Traffic distribution in 60-node BRITE with no sub-path reinforcement.

In the second part of the experiment, the paths of four source-destination pairs were enumerated and graphs were plotted between path costs (cumulative) and percentage of traffic along those paths. The four source-destination pairs of interest were: (i) largest degree node and second largest degree node (Figure 4.34 and Figure 4.35), (ii) largest degree node and smallest degree node (Figure 4.36 and Figure 4.37), (iii) smallest degree node and largest degree node (Figure 4.38 and Figure 4.39), and (iv) smallest degree node and second smallest degree node (Figure 4.40 and Figure 4.41). All these graphs also reiterated the cost sensitive nature of our model-based routing algorithm.

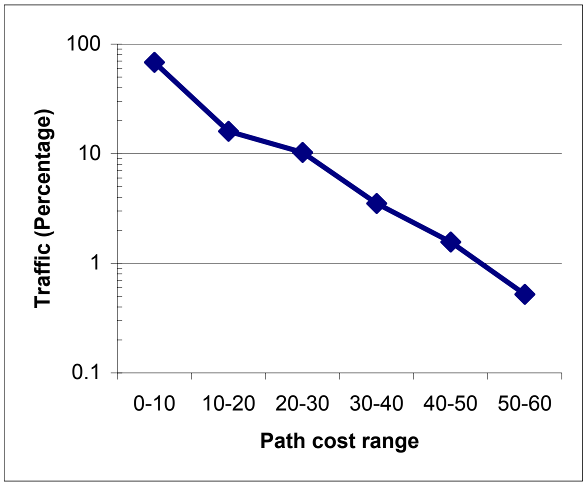


Figure 4.34 Traffic distribution from largest degree to second largest degree nodes in BRITE topology (sub-path reinforcement).

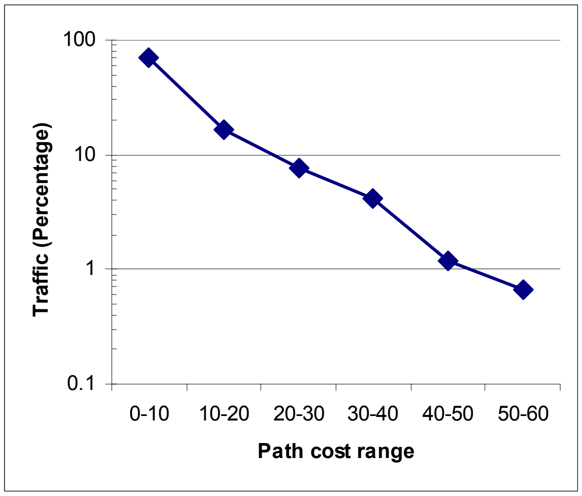


Figure 4.35 Traffic distribution from largest degree to second largest degree nodes in BRITE topology (no sub-path reinforcement).

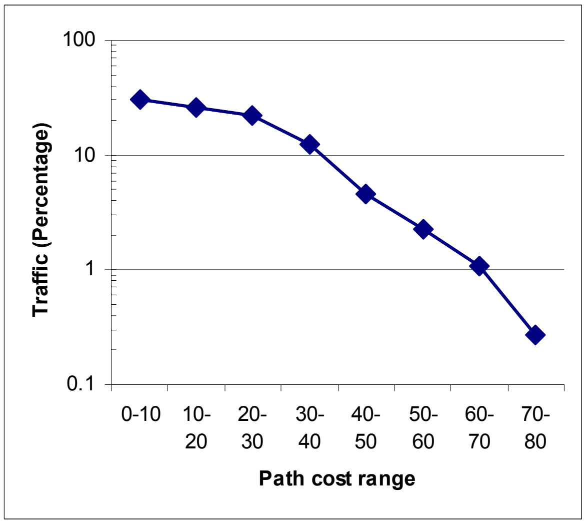


Figure 4.36 Traffic distribution from largest degree to smallest degree nodes in BRITE topology (sub-path reinforcement).

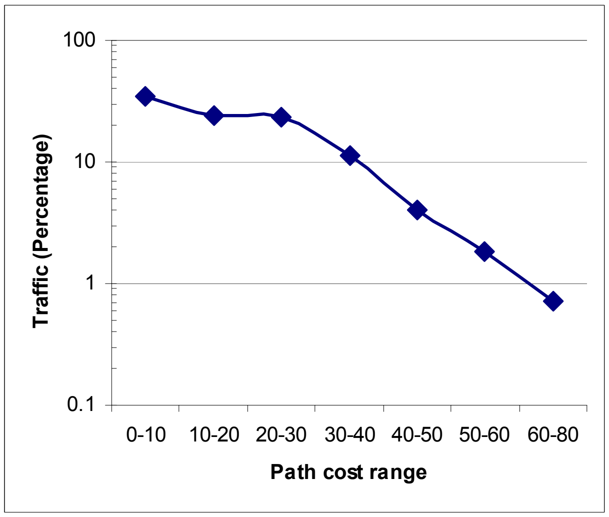


Figure 4.37 Traffic distribution from largest degree to smallest degree nodes in BRITE topology (no sub-path reinforcement).

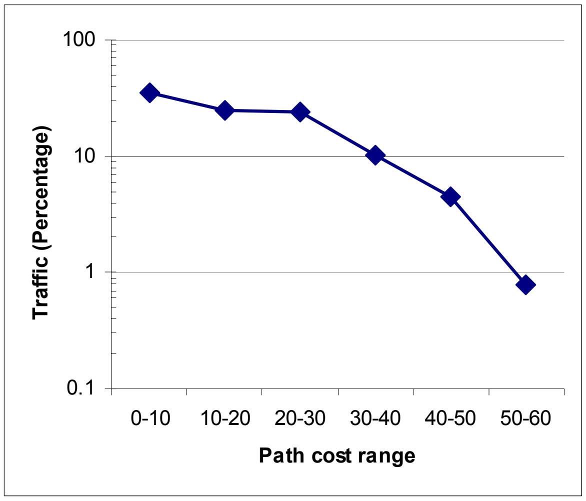


Figure 4.38 Traffic distribution from smallest degree to largest degree nodes in BRITE topology (sub-path reinforcement).

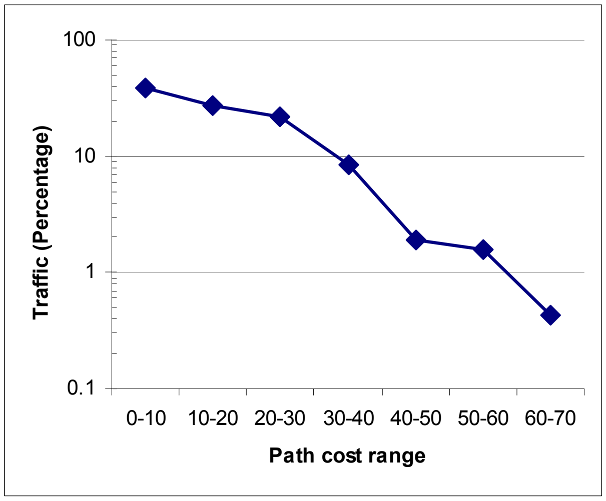


Figure 4.39 Traffic distribution from smallest degree to largest degree nodes in BRITE topology (no sub-path reinforcement).

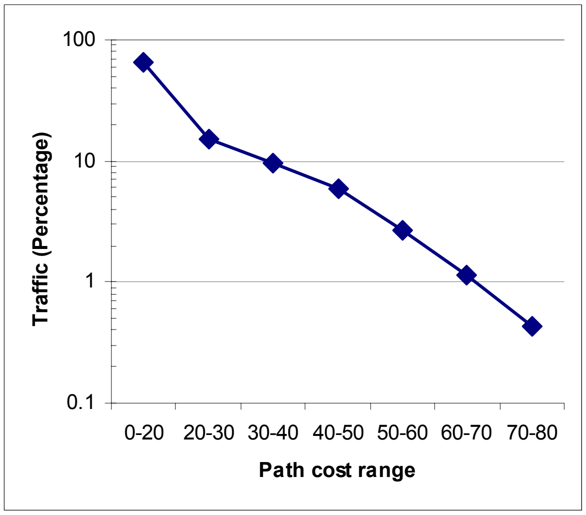


Figure 4.40 Traffic distribution from smallest degree to second smallest degree nodes in BRITE topology (sub-path reinforcement).

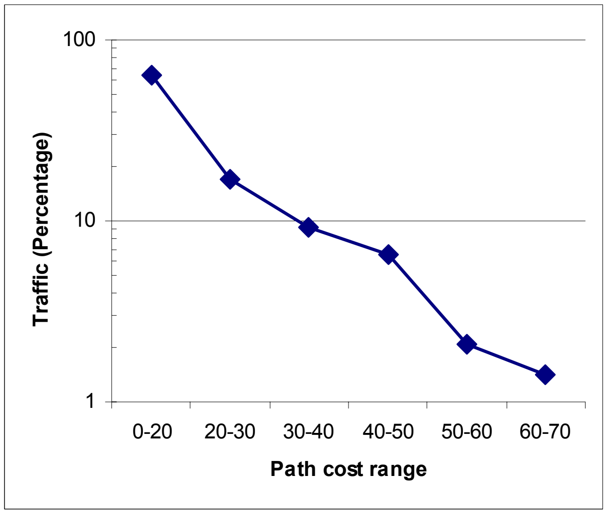


Figure 4.41 Traffic distribution from smallest degree to second smallest degree nodes in BRITE topology (no sub-path reinforcement).

Chapter 5. Conclusion and Future Work

In this thesis, we have argued for the reinforcement learning approach to achieve reachability routing, where the goal of the routing algorithm is to efficiently distribute traffic among all paths leading to a destination. We also presented a new model-based RL algorithm, which achieves true cost-sensitive reachability routing, even in network topologies that pose problems to both deterministic routing as well as classical RL formulations. The evaluation results clearly indicate that our approach achieves true multi-path routing, with traffic distributed among the multiple paths in inverse proportion to their costs. By helping maintain the incremental spirit of current backbone routing algorithms, this approach has the potential to form the basis of the next generation of routing protocols, enabling a fluid and robust backbone routing framework.

5.1 Future Work

We now present four possible directions for future work.

- **Adaptive configuration of the threshold factor ($\boldsymbol{\tau}$)**

The threshold factor ($\boldsymbol{\tau}$) is currently set to a fixed value for all the nodes in the topology. From the operating curve, the network administrator determines the optimal value of $\boldsymbol{\tau}$ at which the routing yields high success and multipath percentage while keeping the percentage of packets entering into loops low. As part of the future work, we can determine the $\boldsymbol{\tau}$ value dynamically based on available information and periodically adjust its value to obtain the optimal routing requirements. The $\boldsymbol{\tau}$ value could be dynamically adapted on a per-node basis or on per-source-destination-pair basis at every node.

- **Instructive Feedback**

Our RL algorithm works primarily using evaluative feedback from neighboring routers. It would be interesting to extend the framework to accommodate instructive feedback. But to provide instructive feedback, a router must have sufficient discriminating capability to perform credit assignment. It is typically of the case that any resulting instruction will be of the 'negative' kind i.e., 'for destination X, *do not* use interface i_y'. How such negative instructions can co-exist with positive reinforcements is an important research issue, not only for our application domain, but also the larger field of reinforcement learning.

- **Modeling topologies with hierarchical addressing**

Currently the algorithm assumes all topologies to be flat such that all nodes in the topology are numbered from 1 to n. By supporting hierarchical addressing of the nodes, the model built at every node could be at a sub network basis instead of being at a per node basis, i.e. a node could collect statistics for a group of nodes as a single entity and build its model accordingly. Such an approach encourages problem decomposition and enables scaling up to large network sizes.

• **Reverse engineering routing protocols**

The model-based reinforcement learning algorithm presented here promises to serve as an abstraction of reachability routing algorithms in general. One idea for further research is to automatically mine the model by analyzing implemented routing algorithms' behavior, rather than incrementally learning it from scratch, as we have done here. In other words, we can seek to 'imitate' the functioning of another algorithm by suitably configuring our model. This problem has its roots in *inverse reinforcement learning,* where we are aiming to 'recover' an algorithm from observed (optimal) behavior. The first steps toward such reverse engineering have been recently taken [19].

Bibliography:

[1] P. Baran. **On Distributed Communication Networks.** IEEE Transactions on Communications Systems, Vol. CS-12: pages 1–9, 1964.

[2] D.P. Bertsekas and J.N. Tsitsiklis. **Neuro-Dynamic Programming.** Athena Scientific, Belmont, MA, 1996.

[3] D.P. Bertsekas and J.N. Tsitsiklis. **Parallel and Distributed Computation: Numerical Methods.** Athena Scientific, Belmont, MA, 1997.

[4] J. Boyan and M. Littman. **Packet Routing in Dynamically Changing Networks: A Reinforcement Learning Approach.** In Advances in Neural Information Processing Systems 6 (NIPS6), pages 671–678. Morgan Kaufmann, San Francisco, CA, 1994.

[5] J. Chen, P. Druschel, and D. Subramanian. **A New Approach to Routing with Dynamic Metrics.** In Proceedings of the IEEE INFOCOM Conference on Computer Communications, pages 661–670. IEEE Press, New York, March 1999.

[6] J. Chen, P. Druschel, D. Subramanian. **A Simple, Practical Distributed Multi-Path Routing Algorithm.** TR98-320. Department of Computer Science, Rice University. July 1998.

[7] R. Coltun. **OSPF: An Internet Routing Protocol.** ConneXions, Vol. 3(8): pages 19–25, 1989.

[8] G. Di Caro and M. Dorigo. **AntNet: Distributed Stigmergetic Control for Communications Networks.** Journal of Artificial Intelligence Research, Vol. 9, pages 317–365, 1998.

[9] E.W. Dijkstra and C.S. Scholten. **Termination Detection for Diffusing Computations.** Information Processing Letters, 11:1-4, August 1980.

[10] C. Guestrin, M. Lagoudakis, and R. Parr. **Coordinated Reinforcement Learning.** In Machine Learning: Proceedings of the Nineteenth International Conference (ICML 2002), pages 227-234. University of New South Wales, Sydney, Australia, July 2002.

[11] C. Hedrick. **Routing Information Protocol.** Request for Comments 1058, Network Working Group, June 1988.

[12] L.P. Kaelbling, M.L. Littman, and A.W. Moore. **Reinforcement Learning: A Survey.** Journal of Artificial Intelligence Research, Vol. 4: pages 237–285, 1996.

[13] G. Malkin. **RIP Version 2.** Request for Comments 2453, Network Working Group, November 1998.

[14] A. Medina, A. Lakhina, I. Matta, J. Byers. **BRITE: Universal Topology Generation from a User's Perspective.** Technical Report, BUCS-TR2001-003, Boston University, 2001.

[15] J. Moy. **OSPF Version 2.** Request for Comments 1247, Network Working Group, July 1991.

[16] J. Moy. **OSPF Version 2.** Request for Comments 1583, Network Working Group, March 1994.

[17] M. E. J. Newman, S. H. Strogatz, and D. J. Watts. **Random graphs with arbitrary degree distributions and their applications.** Physics Review E 64, 026118 pages 1-16, (2001).

[18] L. Peterson, B. Davie. **Computer Networks: A Systems Approach.** Morgan Kauffmann Publishers Inc., Second Edition 2000.

[19] D. Shiraev. **Inverse Reinforcement Learning and Routing Metric Discovery.** M.S. Thesis, Department of Computer Science, Virginia Tech, August 2003.

[20] M. Steenstrup. **Routing in Communications Networks.** Prentice Hall, 1995.

[21] D.W. Stockburger. **Introductory Statistics: Concepts, Models, and Applications.** Atomic Dog Publishing, 2001.

[22] D. Subramanian, P. Druschel, and J. Chen. **Ants and Reinforcement Learning: A Case Study in Routing in Dynamic Networks.** In Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97), pages 832–839. Morgan Kaufmann, San Francisco, CA, 1997.

[23] R.S. Sutton and A.G. Barto. ***Reinforcement Learning*.** MIT Press, Cambridge, MA, 1998.

[24] S. Varadarajan, N. Ramakrishnan, M. Thirunavukkarasu. **Reinforcing Reachable Routes.** Computer Networks, Vol. 43, No. 3, pages 389-416, Oct 2003.

[25] S. Varadarajan. **Ethereal: A Fault Tolerant Host-Transparent Mechanism for Bandwidth Guarantees over Switched Ethernet Networks.** PhD thesis, Department of Computer Science, State University of New York, Stony Brook, 2000.

[26] S. Vutukury and J.J. Garcia-Luna-Aceves. **MDVA: A Distance-Vector Multipath Routing Protocol.** In Proceedings of the IEEE INFOCOM Conference on Computer Communications, pages 557–564. IEEE Press, 2001.

[27] S. Vutukury and J.J. Garcia-Luna-Aceves. **A Simple Approximation to Minimum-Delay Routing.** ACM SIGCOMM Computer Communication Review, Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication**,** Volume 29 Issue 4 pages 227 - 238, August 1999.

[28] S. Vutukury and J.J. Garcia-Luna-Aceves. **An Algorithm for Multipath Computation using Distance-Vectors with Predecessor Information.** In Proceedings of 8th International Conference of IEEE Computer Communications and Networks, pages 534-539. IEEE Press, 1999.

[29] S. Vutukury and J.J. Garcia-Luna-Aceves. **A Distributed Algorithm for Multipath Computation.** In Proceedings of GLOBECOM (Global Telecommunications Conference), Volume 3, pages 1689-1693. IEEE Press, 1999.

[30] S. Vutukury. **Multipath Routing Mechanisms for Traffic Engineering and Quality of Service in the Internet.** PhD Thesis, Department of Computer Science, University of California, Santa Cruz. 2001.

[31] **Routing Basics: Internetworking Technology Handbook.**
http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/index.htm

Vita

Muthukumar Thirunavukkarasu was born in Neyveli, India in June of 1980. He received his Baccalaureate degree in Computer Science and Engineering from College of Engineering, Anna University. As part of his undergraduate research, he developed a protocol for delay-sensitive, loss tolerant multicast applications using RSVP.

While pursuing his Master's Degree in Computer Science at Virginia Tech, Muthukumar worked as a Graduate Teaching Assistant for the following courses: Introduction to Data Structures and Software Engineering (CS 1704, sophomore level), Computer Network Architecture and Programming (CS 4254, senior level) and Communication Networks (CS 5516, graduate level). His research interests during his graduate studies have included computer networks, artificial intelligence, and data mining. He plans to take up a position with IBM Silicon Valley Labs, San Jose CA upon graduation in May 2004.