Reinforcing Reachable Routes

Srinidhi Varadarajan, Naren Ramakrishnan, and Muthukumar Thirunavukkarasu
Department of Computer Science
Virginia Tech, VA 24061, USA
Email: {srinidhi,naren,mthiruna}@cs.vt.edu

Abstract

This paper studies the evaluation of routing algorithms from the perspective of reachability routing, where the goal is to determine all paths between a sender and a receiver. Reachability routing is becoming relevant with the changing dynamics of the Internet and the emergence of low-bandwidth wireless/ad-hoc networks. We make the case for reinforcement learning as the framework of choice to realize reachability routing, within the confines of the current Internet 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 current backbone routing algorithms. We identify research issues in reinforcement learning applied to the reachability routing problem to achieve a fluid and robust backbone routing framework. This paper also presents the design, implementation and evaluation of a new reachability routing 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. The paper is targeted toward practitioners seeking to implement a reachability routing algorithm.

1 Introduction

With the continuing growth and dynamicism of large scale networks, alternative evaluation criteria for routing algorithms are becoming increasingly important. The emergence of low-bandwidth ad-hoc mobile networks requires routing algorithms that can distribute data traffic across multiple paths and quickly adapt to changing conditions. Multi-path routing offers several advantages, including better bandwidth utilization, bounding delay variation, minimizing delay, and improved fault tolerance. Furthermore, current single path routing algorithms face route oscillations (or flap), since they switch routes as a step function. The solution has been to choose low variance routing metrics that are not amenable to route flap, which incidentally are also metrics that don't represent the true state of the network. Good multi-path routing involves gradual changes to routes and should work well even with high variance routing metrics.

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. Our approach is to study multipath routing within the confines of the current Internet protocol, which leads to interesting design decisions.

In this paper, we approach multi-path routing from the limiting perspective of reachability routing, where the routing algorithm attempts to determine all paths between a sender and a receiver. We present a survey of algorithm design methodologies, with specific reference to capturing reachability considerations. The paper is structured as a series of arguments and observations that lead to identifying reinforcement learning as the framework to achieve reachability routing. We consider tradeoffs in configuring reinforcement learning and pitfalls in traditional

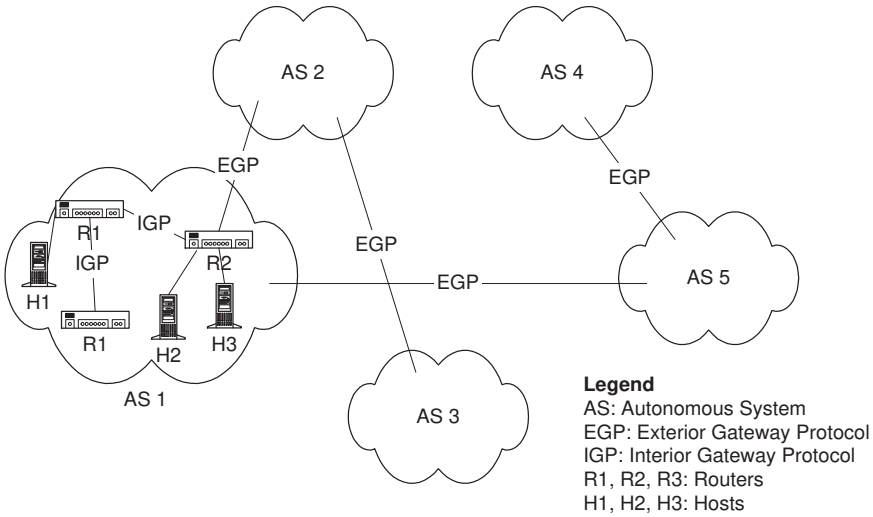

Figure 1: Organization of a network.

approaches. These ideas and arguments are focused in the end of the paper toward the practical design, implementation, and evaluation of a reachability routing algorithm on concrete topologies. By identifying novel dimensions for characterizing routing algorithms and showcasing important implementation considerations, our work helps provide organizing principles for the development of practical reachability routing algorithms.

2 Definitions

A **network** (see Fig. 1) consists of nodes, where a node may be a **host** or a **router**. Hosts generate and consume the data that travels through the network. Routers are responsible for forwarding data from a source host to a destination host. Physically, a router is a switching device with multiple **ports** (also called **interfaces**). Ports are used to connect a router to either a host or another router. On receiving a data packet through a port, a router extracts the destination address from the packet header, consults its routing table, and determines the outgoing port for that data packet. The routing table is a data structure internal to the router and associates destination network addresses with outgoing ports. Routing is thus a many-to-one function which maps (many) destination network addresses to an outgoing port. In the case of IP networks, this function maps a 32 bit IP address space to a 4-7 bit output port number. Intuitively, the quality of routing is directly influenced by the accuracy of the mapping function in determining the correct output port. The reader should keep in mind that routers are physically distinct entities that can only communicate by exchanging messages. The process of creating routing tables hence involves a distributed algorithm (the **routing protocol**) executing concurrently at all routers. The goal of the routing protocol is to derive loop-free paths.

Organizationally, a network is divided into multiple autonomous systems (**AS**). An autonomous system is defined as a set of routers that use the same routing protocol. Generally, an autonomous system contains routers within a single administrative domain. An Interior Gateway Protocol (**IGP**) is used to route data traffic between hosts (or networks) belonging to a single AS. An Exterior Gateway Protocol (**EGP**) is used to route traffic between distinct autonomous systems.

The effectiveness of a routing protocol directly impacts both the end-to-end throughout and end-to-end delay. Current network routing protocols are primarily concerned with deriving shortest cost routes between a source and a destination. This focus on an optimality metric[1] means that current protocols are tailored toward **single path**

[1]Note that the notion of optimality is used in this paper with respect to a node's view of the network, and does not reflect optimality

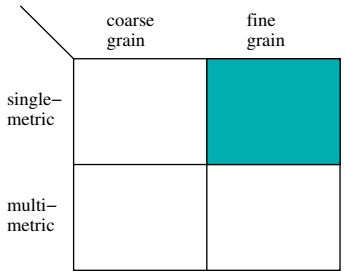

Figure 2: Four basic categories of algorithms for multi-path routing. The shaded region depicts the class of algorithms studied in this paper.

routing[2]. In the recent past, there has been an increasing emphasis on **multi-path routing**, where routers maintain multiple distinct paths of arbitrary costs between a source and a destination.

Multi-path routing presents several advantages. 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, multi-path routing makes effective use of the 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. In a single-path routing algorithm, use of a good congestion indicator (such as average queue length at a router) as a cost metric leads to route oscillations.

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 table is based on a different cost metric. We refer to this as a **multi-metric**, multi-path routing approach . A second approach is to allow multiple network paths between a source-destination pair for a *single cost metric*. This means that routers may use sub-optimal paths; for instance a router may send data on multiple paths to maximize network throughput. We refer to this a **single-metric**, multi-path routing approach.

Multi-path routing algorithms can also be distinguished by the routing granularity into **coarse grain**, connection- (or flow-) oriented or **fine grain**, connectionless approaches. The former adopts a path-per-connection view where all packets belonging to a single connection follow the same path. However, different connections between the same source and destination hosts may follow different paths. In contrast, connectionless networks have no mechanism to associate packets with any higher-level notion of a connection; hence multi-path routing in connectionless networks requires a fine-grained approach. For true multi-path routing, the routing algorithm should forward packets between a single source-destination pair along multiple paths, *which may not necessarily be shortest-cost paths*. The focus of this paper is on such fine grain multi-path routing algorithms within a single-metric domain (see Fig. 2). These algorithms can be trivially extended for use in both coarse grain as well as multi-metric routing domains.

One way to achieve this form of multi-path routing is to extend existing single path network routing protocols. This extension is non-trivial for two reasons. First, we need mechanisms to incorporate state corresponding to multiple (possibly non-optimal) paths into the routing table. More importantly, we need new loop detection 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 (and keep consistent) routing state proportional to the number of paths in the network.

according to some global criterion (such as minimizing total traffic). For a comprehensive treatment of globally optimal routing algorithms, refer to [3].

[2]This scheme can be trivially extended to the case when there are multiple shortest-path routes.

In this paper, 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 aforementioned state or consistency maintenance overhead. This paper introduces two forms of reachability routing. In **hard reachability**, the routing table at each router contains all *and only* loop free paths that exist in the network topology. **Soft reachability**, on the other hand, merely requires that all loop free paths be represented in the routing table. 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.**

As we will show later, practical limitations on the amount of state that can be carried by a network packet preclude *any solution* for hard reachability[3]. Hence, this paper addresses the problem of soft reachability. We argue that even this goal cannot be achieved by directly extending existing routing protocols or even by explicitly programming for it. Instead, we achieve reachability routing by exploiting the underlying semantics of probabilistic routing algorithms. The algorithms we advocate ensure correct operation of the network even under soft reachability.

3 Background

Before we look at algorithm design methodologies, it would be helpful to review the standard algorithms that form the bulwark of the current network routing infrastructure. While some of these have not been designed with reachability in mind, they are nevertheless useful in characterizing the design space of routing algorithms. The survey below is merely intended to be representative of current network routing algorithms; for a more complete survey, see [20]. This section addresses deterministic routing algorithms and the next addresses probabilistic routing algorithms. What is relevant for our purposes are not the actual algorithms but rather their signature patterns of information exchange.

3.1 Link State Routing (OSPF)

Link-state 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 [17, 18], developed by the IETF for TCP/IP networks. OSPF is an Interior Gateway Protocol in that it is used to communicate routing information between routers belonging to the same autonomous system [8].

The connectivity information broadcast by every router includes the list of its neighboring routers and the cost to reach every one of them, where a neighboring router is an adjacent node in the topology map. After such broadcasts have flooded through the network, every router running the link-state algorithm constructs a map of the (global) network topology and computes the cost — a single valued dimensionless metric — of each link of the network. Using the network topology, each router then constructs a shortest path tree to all other routers in the autonomous system, with itself as the root of the tree. This is typically done using Dijkstra's shortest path algorithm. While the shortest path tree gives the entire path to any destination in the AS, a router need only know the outgoing interface for the next hop along a path. This information is captured in the routing table maintained by each router. The routing table thus contains routing entries which associate a destination address in an incoming data packet with the appropriate outgoing physical interface. The defining characteristic of a link state algorithm is that *each router sends information about local neighbors to all participating routers*.

[3]To achieve hard reachability for single-metric fine grain routing, the data packet has to carry an arbitrary-length list of visited routers. Fixed-length network packet headers cannot accommodate this information.

Link-state algorithms are generally dynamic in nature. As the network topology or link costs change, routers exchange information and recompute shortest path trees to ensure that their local database is consistent with the current state of the network. The optimality principle ensures that as long the topological maps are consistent, the routing tables computed by each router will also be consistent.

To derive the time complexity of the link-state routing algorithm, note that computing the routing table involves running Dijkstra's algorithm on the network topology. If the network contains R routers, the asymptotic behavior of the standard implementation of Dijkstra's algorithm is given by $O(R^2)$. A heap based implementation of Dijkstra's algorithm reduces the computational complexity to $O(R\,log\,R)$. This computational cost is lower than the distance-vector protocol discussed in the next section. However, link-state algorithms trade off communication bandwidth against computational time. To derive the communication cost, note that the size of the routing topology transmission by each router is proportional to N, the number of neighbors connected to the router. Since the routing topology is broadcast to every other router, every routing transmission travels over all links (L) in the network. Hence, the communication cost of a routing topology transmission by a single router is $O(NL)$ and the cumulative cost of the routing transmissions by all routers is $O(RNL)$. We make three observations about link-state algorithms.

Observation 1. Routers participating in a link-state algorithm transmit raw or non-computed information among themselves, which is then used as the basis for deriving routing tables. The advantage of this scheme is that a router only sends information it is sure of, as opposed to 'hearsay' information used by the distance-vector routing protocols described in the next section.

Observation 2. Link-state algorithms are intrinsically targeted towards single-path routing since they base their correctness on the optimality principle. A trivial extension allows OSPF (in particular) to use multi-path routing when two paths have identical costs, since this does not violate the optimality principle. Another extension allows multiple shortest path trees, where each tree is based on a different cost metric.

Observation 3. Link-state algorithms have an explicit global information collection phase before they can populate routing tables and begin routing.

3.2 Distance Vector Routing (RIP)

As opposed to link-state algorithms, which have a global information collection phase, distance-vector algorithms build their routing tables by an iterative computation of the distributed Bellman-Ford algorithm. The most common manifestation of distance-vector algorithms in the TCP/IP Internet is in the form of the Routing Information Protocol (**RIP**) [13, 15]. RIP is based on the 1970s Xerox network protocols used in XNS networks, with adaptations to enable it to work in IP networks.

In the distance-vector protocol (DVP), every router maintains a routing database, which only contains the best known path costs to each destination router in the AS. In each iteration, every router in the AS sends its routing tables, to all its neighbors. On receiving a routing table, each target router compares the routing entries in the received routing table with its own entries. 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. The defining characteristic of DVP algorithms is that *each router sends information about all participating routers to its local neighbors.*

When the DVP algorithm begins, each DVP router knows the link cost to its neighbors. In the first iteration of the DVP algorithm, each router sends information about its neighbors to its neighbors. At the end of the 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, each router expands its horizon by 1, i.e., 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.

To derive the time complexity of this algorithm, note that on each iteration, a router receives $O(N)$ routing tables, where N is the number of neighbors. Each routing table contains R entries, where R is the number of participating routers in the AS. On each iteration, every router in the AS expands the network neighborhood that it knows about by 1. The algorithm stabilizes when each router has expanded its horizon to the diameter of the network D. Hence the time complexity of DVP is $O(NRD)$.

The traditional DVP suffers from a classic convergence problem called 'count to infinity.' Assume a network with 4 routers A, B, C and D connected linearly, i.e. $A \leftrightarrow B \leftrightarrow C \leftrightarrow D$. Assume that A's best path cost to D is x. If router D is removed from the network, C advertises a path cost (to D) of infinity to B, but in the same iteration A announces its previous best path cost x to B, without realizing that its route to D goes through B. Since x is less than infinity, B essentially ignores the update from C. In the next iteration, B then propagates its best cost to D to routers A and C. In the following iteration, A updates its path cost estimate to D since it received an update from B, which affects its lowest cost route to D. This change in the lowest cost is sent to B on the next iteration, which updates its estimate again. The routers are now stuck in a loop, incrementing their path costs on each iteration, till they reach the upper bound on path costs, which is nominally defined to be infinity.

The standard solution to the count to infinity problem is to enforce an upper bound on the path costs. The path cost metric generally used in DVP is the length of the path. Hence, the upper bound on path costs translates to an upper bound on the diameter of the network. The RIP (v1; [13]) restricts the diameter of the network to 15 hops.

The problem with the traditional solution is twofold. First, restricting the network to small diameters impedes scalability. Second, the length of a path is not a good indicator of the quality of the path. The problem with choosing better cost metrics — such as average queue length at a router or minimum available bandwidth along a path — is that it increases convergence time significantly. Several solutions have attempted to address this issue by speeding up the time taken to count to infinity. However, note that there is no solution to eliminate the count to infinity problem, using just the information collected by the DVP. The only solution to the count to infinity problem is to maintain explicit path information along with the best cost estimate. This mechanism is used by the path vector routing protocol described later.

The main advantage of the DVP is that amount of routing information sent is quite small. In contrast to the link-state algorithm, routing information is only sent to neighbors, which significantly reduces the network bandwidth requirement. Furthermore, DVP does not have an explicit information collection phase — it builds its routing tables incrementally. Hence, it can begin routing as soon as it has any path cost estimate to a destination. From the perspectives of this paper, we make two observations about distance-vector protocols.

Observation 4. Distance-vector protocols pass computed information or 'hearsay' among themselves. This hearsay is not qualified in any way — for instance, routers indicate their best path cost, but not the path itself.

Observation 5. Distance-vector protocols are intrinsically targeted towards single-path routing, since each router filters the routing updates it receives and only transmits the best route.

3.3 Comparing Link-State and Distance-Vector Protocols

The distance-vector and link-state protocols have traditionally been considered as two orthogonal approaches to network routing. Alternatively, we can view them as two extremes along a 'scope of information qualification' axis, which allows us to interpolate between these algorithms. In the link-state protocol, each router sends raw cost information about its immediate connectivity. In this case, we define the scope of information qualification to be 1, or the distance to the immediate neighbor. At the other extreme, we have the distance-vector protocol in which each router sends cost information about every other router, i.e., the scope of information qualification is infinity, or more precisely the diameter of the network. A generalized algorithm will employ a parameter α to denote the diameter of the neighborhood that is viewed as a single 'super node' by the routing algorithm. Within the super node, the

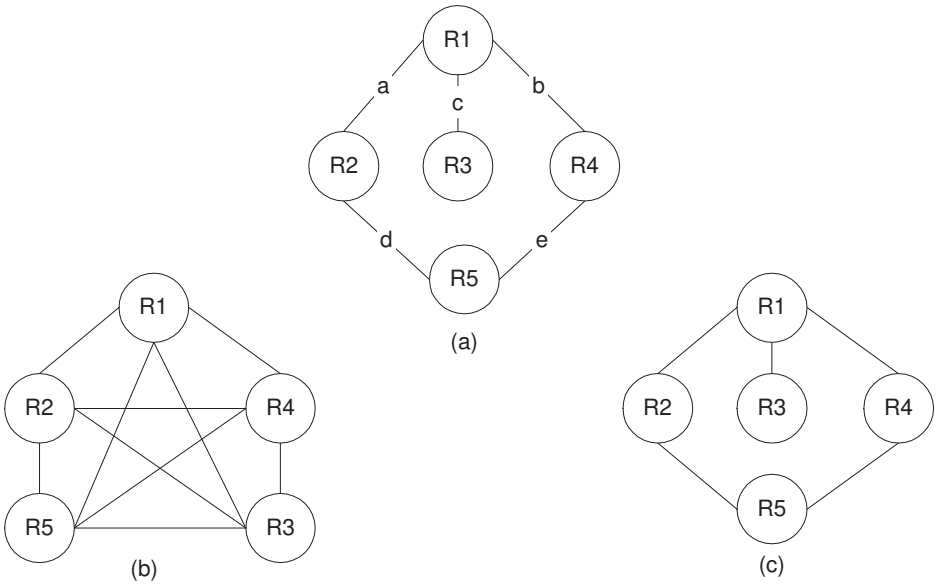

Figure 3: Topology of the data network (a) and the topologies of the corresponding control networks for a link-state algorithm (b) and a distance-vector algorithm (c).

distance-vector protocol is used to compute paths, and the link-state protocol operates at the level of super-nodes. As α tends to the diameter of the network, the size of the super node tends to the size of the entire network, which collapses the generalized algorithm to the distance-vector protocol.

In addition to the interpolatory viewpoint, it is instructive to contrast the operational behavior of the link-state and distance-vector routing protocols. We can think of a single network as consisting of two superimposed components: a data network, which only carries end user data and a control network, which carries the routing information used by routers to determine routes in the data network. This viewpoint studies the topology of the control network induced by a routing protocol and its relation to the topology of the data communication network (see Fig. 3).

Observation 6. A link-state algorithm broadcasts raw topology information to all routers in the network using a pruned flooding approach to eliminate data loops. Since the raw topology information can be locally collected by each router, the topology of the parallel control network is distinct from the topology of the data network. Every node in the control network is connected to every other node. This illustrates the fact that the environment about which we learn (to route) is distinct from the mechanism used to communicate the routing information. Such a distinction enables the separation of the data collection and routing phases.

Observation 7. In contrast, in the distance-vector algorithm each router communicates best-cost path information to all its neighbors. Computing the best-cost path requires that the paths present in the data network be present in the control network as well. Hence, the topology of the control network has to be identical to the data network topology. In effect, each link in the control network mirrors a physical link in the data network. This illustrates the fact that the mechanism used to communicate routing information is the same as the environment where the information is to be used.

3.4 Path Vector Routing (BGP, IDRP)

The path vector algorithm improves the basic distance-vector protocol to include additional information qualifiers to eliminate the count-to-infinity problem. The Border Gateway Protocol (BGP) and the Inter-Domain Routing

Protocol (IDRP) are two common implementations of path vector routing algorithms. Unlike the link-state and distance-vector routing algorithms, path vector algorithms are generally used between autonomous systems, i.e., path vector is an exterior gateway protocol, operating at the scope of a backbone 'network of networks.' The main motivation behind the path vector algorithm is to allow autonomous systems greater control in routing decisions.

In the path vector algorithm, routers are identified by unique numerical identifiers. Each router maintains a routing table, where each entry in the routing table contains a list of explicit paths — specified as a sequence of router identifiers (path-vector) — to a destination router. The list of path-vectors is ordered based on domain-specific policy decisions — such as contractual agreements between autonomous systems, rather than a quantitative cost metric. This scheme avoids imposing a single, universally adopted cost-metric.

In each iteration, every router in the AS transmits a subset of its routing tables to all its neighbors. In the transmitted subset, each routing table entry contains a single 'best' path-vector to destination router. The 'best' path-vector is the first path-vector in an ordered list of path-vectors. For each routing entry in a received routing table, a router (a) adds its router identifier to the path-vector, (b) checks the newly created path-vector to ensure there are no loops, (c) inserts the path-vector into its own routing table, and (d) sorts the list of path-vectors based on its selection criteria. Paths with loops are discarded, which in effect eliminates the count-to-infinity problem. The algorithm progresses similar to the distance-vector protocol, with each router expanding its horizon by 1 on each iteration. The algorithm finally stabilizes when each router has expanded its horizon to the diameter of the network.

Observation 8. Path vector algorithms are intrinsically targeted towards single-path routing, since each router filters the routing updates it receives and only transmits the best path-vector. Interestingly, the ingress router has a choice of routes; intermediate routers along a path do not have a choice.

Observation 9. Path vector algorithms pass qualified computed information among themselves. While the qualification serves to eliminate problems such as count to infinity, it is generally not sufficient to invert the computation function — to obtain the raw data carried by messages in a link-state algorithm. Lack of raw data complicates the *credit assignment* problem for cost-dependent reachability routing. The credit assignment problem here is primarily structural: of all the nodes, links, and subpaths that contribute to a certain quality metric in a path (e.g., transmission time, path cost), which ones should be rewarded (or penalized)?

3.5 Hierarchical Routing

In TCP/IP networks, each host is identified by a unique numerical identifier (IP address), which consists of a network component and a host component. The network component of the IP address is hierarchically organized, allowing a set of networks to be viewed as a single node in a higher layer of the hierarchy. This hierarchical organization is used to reduce the scope of the routing problem. At the lowest level, routing within a single network translates to routing among the end-hosts. At the highest level, the network can be viewed as a collection of nodes, where each node is a network in itself, running an internal routing algorithm, whose presence is opaque to the higher levels of the hierarchy. This organization allows each level in hierarchy the freedom to choose a routing algorithm suited to its needs.

4 Reinforcement Learning Algorithms

Reinforcement learning (RL) [14] is a branch of machine learning that is increasingly finding use in many important applications, including routing. The ant-based algorithms of Subramanian et al. [21] and the stigmergetic routing framework described in [10] are examples of reinforcement learning algorithms for routing. Here, populating routing tables is viewed as a problem of learning the entries; we hence use the term **learning** in this paper synonymously with the task of determining routing table entries.

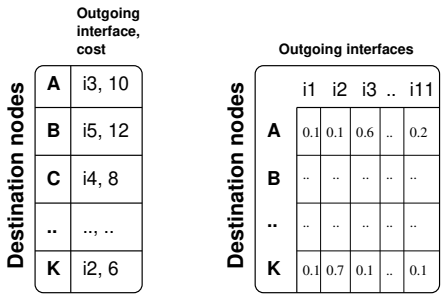

Figure 4: Routing table structure for (left) deterministic routing algorithms and (right) probabilistic routing algorithms.

The salient feature of RL algorithms is the probabilistic nature of their routing table entries. In the previously reviewed deterministic routing algorithms, a routing table entry contains an outgoing interface identifier and a cost. In contrast, routing table entries in RL algorithms contain all outgoing interfaces and associated use probabilities (see Fig. 4). 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. A router can hence use the probabilities for making forwarding decisions in a non-deterministic manner.

Observation 10. The probabilistic nature of routing tables in RL algorithms make them suitable for either single path or multi-path routing. If a router deterministically chooses the outgoing link that has the highest probability, it is implicitly performing single path routing. If the router distributes traffic in proportion to the link probabilities, it is performing multi-path routing.

Learning in RL is based on trial-and-error and organized in terms of episodes. An episode consists of a packet finding its way from an originating source to its prescribed destination. Routing table probabilities are initialized to small random values (taking care to ensure that the sum of the probabilities for choosing among all possible outgoing interfaces is one). A router can thus begin routing immediately except, of course, most of the routing decisions will not be optimal or even desirable (e.g., they might lead to a dead-end). 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.

In either mode of operation, choice of the outgoing interface can be viewed as an **action** taken by the router and RL algorithms assign credit to actions based on reinforcement (**rewards**) from the environment. The reinforcement may take the form of a cost update or a measurable quantity such as bandwidth or end-to-end delay. In response, the probabilities are then nudged slightly up or down to reflect the reinforcement signal. When such credit assignment is conducted systematically over a large number of episodes and so that all actions have been sufficiently explored, RL algorithms converge to solve stochastic shortest-path routing problems. Since learning is happening concurrently at all routers, the reinforcement learning problem for routing is properly characterized as a **multi-agent** reinforcement learning problem.

The multi-path forwarding capability of RL algorithms is similar in principle to **hot potato** or **deflection** routing [1], where each router assumes that it can reach every other router through any outgoing interface. The motivation in hot potato routing is to minimize router buffering requirements by using the network (or more precisely the delay bandwidth product) as a storage element. Routers maintain routing tables of the form shown in Fig. 4 (left). However, if more than one incoming packet tries to transit the same outgoing link, instead of buffering the excess packets as traditional routers do, hot potato routing selects a free outgoing link randomly and transmits the packets. The randomly routed packets will eventually reach their destinations, albeit by following circuitous paths.

Observation 11. While the nature of routing tables in hot potato routing is targeted toward single path routing, the ability to deflect packets for the same destination along multiple links, in fact, realizes soft reachability routing. In contrast to hot potato routing's mechanism of indiscriminately selecting alternatives, the goal in RL is to make an informed decision about reachable routes.

4.1 Novel Features of RL Algorithms

Algorithms for reinforcement learning face the same issues as traditional distributed algorithms, with some additional peculiarities. First, the environment is modeled as stochastic (especially links, link costs, traffic, and congestion), so routing algorithms can take into account the dynamics of the network. However, no model of the dynamics is assumed to be given. This means that RL algorithms have to sample, estimate, and perhaps build models of pertinent aspects of the environment. RL algorithms range from those that build elaborate models to those that function without ever building a model.

Second, reinforcement from trying out route possibilities almost always takes the form of *evaluative* feedback, and is rarely *instructive* [22]. For instance, a router conducting RL will be told that its decision to forward packet for destination C onto outgoing interface i_3 resulted in a travel time of 16ms, but not if this travel time is good, bad, or the best possible. Since trip time is composed of all subpath elapsed times, it is computed (and delayed) information, and can only be used as a reinforcement signal and not as an instructive signal. Credit assignment based on the reinforcement signal is hence central to RL algorithms, and is conducted over learning episodes. Episodes are typically sampled to uniformly cover the space of possibilities. To guarantee convergence in stochastic environments, some form of an iterative improvement algorithm is often used.

Finally RL algorithms, unlike other machine learning algorithms, do not have an explicit learning phase followed by evaluation. Learning and evaluation are assumed to happen continually. As mentioned earlier, this brings out the tension between exploration and exploitation. Should the router choose an outgoing interface that has been estimated to have a certain quality metric (exploitation) or should it choose a new interface to see if it might lead to a better route (exploration)? In a dynamic environment, exploration never stops and hence balancing the two tensions is important. The combination of trial-and-error, reinforcement from delayed information, and the exploration-exploitation dilemma make RL an important subject in its own right. For a nice introduction to RL, we refer the reader to [22]. A more mathematical overview is provided in the formally titled *Neuro-Dynamic Programming* [4].

4.2 Q-Routing: An Asynchronous Bellman-Ford Algorithm

To make our discussion concrete, we present the basics of Q-routing [6], one of the first RL algorithms for routing. It is an online asynchronous relaxation of the Bellman-Ford algorithm used in distance vector protocols. 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. In the original formulation presented in [6], Q is set to be the estimated time for delivery. We can think of the routing probabilities as being indirectly derived from Q. There are several alternatives here. For instance, the probability that router x will route a packet for destination d via i_s can be defined to be

$$\frac{Q_x(d, i_s)}{\sum_k Q_x(d, i_k)}$$

Alternatively, in [6], the authors actually learn a deterministic routing policy, so the packet is routed along

$$\mathrm{argmax}_k Q_x(d, i_k)$$

With this formulation, in Fig. 4, data packets bound for destination **A** will be routed to interface i_3.

The operation of the routing algorithms 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 best next router y, determined from Q. 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. In [6], the following update rule is presented:

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

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 a **learning rate** or a **stepsize** and is a standard fixture in iterative improvement algorithms [5]. It is typically set to produce a stepsize schedule that satisfies the stochastic approximation convergence conditions [4]. It should be clear to the reader that this is actually a relaxation of the Bellman-Ford algorithm.

Of course, Q-routing is not guaranteed to converge to the shortest path. In fact, as Subramanian et al. [21] 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. In other words, exploration only happens along the currently exploited path. Another problem with the Q-routing algorithm is that the routing overhead is proportional to the number of data packets.

4.3 Ants as a Communication Mechanism

To circumvent these difficulties, Subramanian et al. propose the separation of the 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. Ants proceed from randomly chosen sources to destinations independently of the data traffic. An ant is a small message moving from one router to another that enables the router to adjust its interface probabilities. 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:

$$p_k = \frac{p_k + \Delta p_k}{1 + \Delta p_k}$$

whereas the other probabilities are adjusted as:

$$p_j = \frac{p_j}{1 + \Delta p_k}$$

where $\Delta p_k \propto 1/f(c)$, with f being some non-decreasing function of the cost c.

The only pending issue is how the ants should be routed. Subramanian et al. provide two types of ants. In the first, so-called *regular ants*, the ants are forwarded probabilistically according to the routing tables. This ensures that the routing tables converge *deterministically* to the shortest paths in the network. In the *uniform ants* version, the ant forwarding probability is a uniform distribution i.e., all links have equal probability of being chosen. This ensures a continued mode of exploration. In such a case, the routing tables do not converge to a deterministic answer; rather, the probabilities are partitioned according to the costs.

Observation 12. The regular ants algorithm treats the probabilities in the routing tables as merely an intermediate stage toward learning a deterministic routing table. Except in the transient learning phase, this algorithm is targeted toward single path routing.

Observation 13. The constant state of exploration maintained by the uniforms ants algorithm ensures a true multi-path forwarding capability. This observation is echoed in [21].

The reader will appreciate the tension between exploration and exploitation brought out by the two types of ants. Regular ants are good exploiters and are beneficial for convergence in static environments. Uniform ants are explorers and help keep track of dynamic environments. Subramanian et al. propose 'mixing' the two types of ants to avail the benefits of both modes of operation.

4.4 Stigmergetic Control

The assumption of link cost symmetry made by both the ant algorithms is a rather simplistic, but serious one. In addition, the update equations are not adept at handling dynamic routing conditions and bursty traffic. The AntNet system of Di Caro and Dorigo [10] is a very sophisticated reinforcement learning framework for routing. Like the algorithm 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. However, the update rules are very carefully designed and implemented 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. Cycles encountered by an ant result in the ant being discarded. Every router also maintains a model of the local traffic experienced and this model is adaptively refined and utilized to score ant travel times.

5 Design Methodologies for Reachability Routing Algorithms

We now have the necessary background to study how reachability routing algorithms can be designed. We begin by identifying two dimensions along which they can be situated.

5.1 Constructive vs. Destructive Algorithms

Constructive algorithms begin with an empty set of routes and incrementally add routes till they reach the final routing table. Current network routing protocols based upon distance-vector, link-state, and path-vector routing are all examples of constructive algorithms. In contrast, destructive algorithms begin by assuming that all possible paths in the network are valid i.e., they treat the network as a fully connected graph. Starting from this initial condition, destructive algorithms cull paths that do not exist in the physical network. Intuitively, a constructive algorithm treats routes as 'guilty until proven innocent,' whereas a destructive algorithm treats routes as 'innocent until proven guilty.' The exploration mode of reinforcement learning algorithms allows us to think of them as destructive algorithms.

Let us consider the amount of work that needs to be done by an algorithm to achieve reachability routing. For a destructive algorithm, the work done is $\mathcal{W} \propto c$, the number of culled edges. In the case of constructive algorithms, the work $\mathcal{W} \propto l$, the number of added edges.

It is instructive to examine the intermediate stages of the operation of constructive and destructive algorithms. By its very nature, a destructive algorithm stays within the space of connected graph topologies. On the other hand, a constructive algorithm starts with a null set of routes and builds up toward the minimum 1-connected topology. In this interim, the routing tables depict multiple disjoint graphs and do not reflect a physical reality. Intuitively, this translates to a hold time, during which a constructive algorithm cannot route to all destinations, whereas a destructive algorithm can. Fig. 5 depicts this scenario.

Tied to the idea of a space of connected topologies is the notion of incremental computation of routing tables, as motivated by **anytime algorithms**. As originally defined by Dean and Boddy [9], an anytime algorithm is one that provides approximate answers in a way that i) an answer is available at any point in the execution of the algorithm

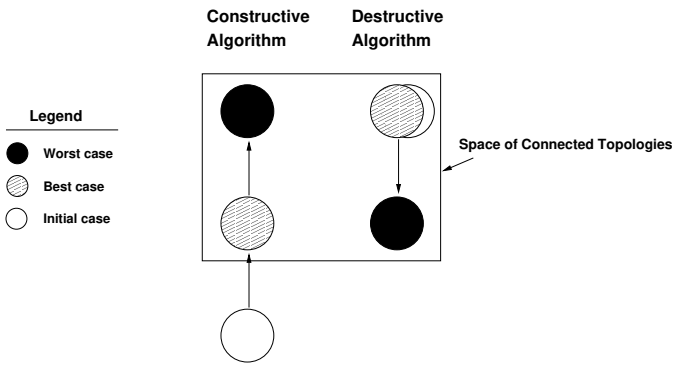

Figure 5: Space of solutions for constructive and destructive algorithms.

and ii) the quality of the answer improves with execution time. For our purposes, a chief characteristic of an anytime algorithm is its interruptibility. In Fig. 5, anytime algorithms can be thought to be traversing the line(s) in the directions shown. They are contrasted by algorithms that experience a sudden transition from the initial state to the final answer. Such algorithms require complete system state information to be able to make such an abrupt transition.

Observation 14. Constructive algorithms cannot function in an anytime mode, before they derive the minimally connected topology. In contrast, destructive algorithms lend themselves naturally to an anytime mode of operation. This means that a destructive algorithm can begin routing immediately.

5.2 Deterministic vs. Probabilistic Routing Algorithms

This is a distinction made earlier; deterministic routing algorithms such as link-state and distance-vector map a destination address to a specific output port. Probabilistic algorithms map a destination address to a set of output ports based on link probabilities.

Observation 15. For a deterministic algorithm, loops are catastrophic. If a data packet encounters a loop, an external mechanism (event or message) is required to break the loop. In contrast, probabilistic algorithms do not require an external mechanism for loop resolution, since the probability of continuing in a loop exponentially decays to zero.

We will explore these classes of algorithms along an axis orthogonal to the constructive versus destructive distinction, leading to four main categories of algorithms (see Fig. 6). Some categories are more common than others.

1. **Constructive Deterministic:** Current network protocols based on link-state, distance-vector, and path-vector algorithms fall in this category. As mentioned earlier, these algorithms focus on single-path routing. To extend them to achieve reachability routing, we need additional qualifiers for routing information. Recall that loops are fatal for deterministic algorithms; hence constructive deterministic algorithms need to qualify the entire path to achieve single-metric multi-path routing. This information qualification can take two forms. In the first form, routers build multiple distinct routing tables to every destination. The data packet then carries information that explicitly selects a particular routing table. This form of qualification requires that each router maintain a routing table entry for *every* possible path in the network, resulting in significant memory overhead. In the second form, data packets can carry a list of previously visited routers which can then be used to dynamically determine a path to the destination. This form of qualification trades time complexity for space complexity and is referred to as **path-prefix routing**. Note that path-prefix routing requires that each router know the entire topology of the network. While this is not an issue for link-state algorithms, it is contrary to the design philosophy of distance-vector algorithms.

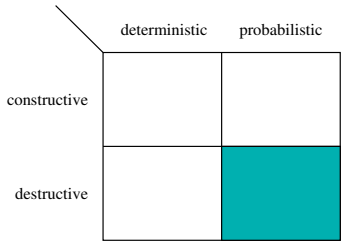

Figure 6: Design methodologies for reachability routing algorithms. We argue for the use of destructive probabilistic algorithms.

2. **Destructive Deterministic:** Destructive algorithms work by culling links from their initial assumption of a fully connected graph. In the intermediate stages of this culling process, the logical topology (as determined by the routing tables) will contain a significant number of loops. Since deterministic algorithms have no implicit mechanism for loop detection and/or avoidance, they cannot operate in destructive mode.

3. **Constructive Probabilistic:** This classification can be interpreted to mean an algorithm that performs no exploration. This can be achieved by having an explicit data collection phase prior to learning. Such algorithms lead to asynchronous versions of distributed dynamic programming [2]. Intuitively, such an algorithm can be thought of as a form of link-state algorithm deriving probabilistic routing tables rather than using Dijkstra's algorithm to derive shortest-path routing tables. The main drawback of this approach is that the communication cost of the data collection phase hinders scalability. This is also the reason why link-state algorithms are not used for routing at the level of the Internet backbone.

4. **Destructive Probabilistic:** By definition, an RL algorithm belongs in this category. In addition to the advantages offered by probabilistic algorithms (loop resolution, multi-path forwarding), RL algorithms can operate in an anytime mode. Since many RL algorithms are forms of iterative improvement, they conduct *independent credit assignment* across updates. This feature reduces the state overhead maintained by each router and enables deployment in large scale networks.

The above categorization clearly builds the case for investigating reachability routing algorithms from the perspective of destructive probabilistic algorithms, particularly as a unified design methodology for large scale networks. The rest of this paper hence concentrates on RL algorithms and identifies practical considerations for their design and deployment.

6 Practical Considerations

There is a stronger motivation to focus on destructive probabilistic algorithms for reachability routing. To see this, we need to analyze the requirements of multi-path routing within the constraints imposed by the current internetworking protocol IP. For a deterministic algorithm to achieve multipath routing, it needs some mechanism to qualify a route (or path) [24]. There are two extremes of qualification: (a) explicit route qualification and (b) implicit route qualification. In (a), each node in the graph has complete topology information, which it uses to build one or more routes to each destination. Each route specifies the complete path — as a list of routers — to the destination. When a data packet arrives at an ingress router, the router embeds the path into the data packet header and sends it to the next router. Each router retrieves the path from the data packet header, and forwards it to the specified 'next-hop' and so on. This scheme is similar to source routing since, from a routing perspective, the source host can be considered synonymous to the ingress router.

In (b), each router may or may not have complete topology information. The path is selected by imposing a metric upon the system, whose evaluation returns the same result independent of the router performing the evaluation. A simple example of such a metric is an optimality criterion. In this case, the path is qualified implicitly, since the data packet does not carry any explicit path information. The problem however, is that purely implicit route qualification leads to single path routing. It may be possible to achieve limited multi-path routing by selecting multiple implicit criteria and signaling the choice of the routing criterion within the header of the data packet.

However, practical design constraints do not permit any form of explicit signaling. In particular, the IP header does not have any space for either carrying a complete route or even signaling an implicit choice of a route. While earlier versions of IP permitted source-routing, it is not used in the current Internet due to security concerns. Furthermore, routers need to both know the complete network topology as well as maintain its consistency to ensure loop resolution. Given the dynamism of the Internet, and the relatively high communication latencies, it is practically impossible to consistently maintain network topology information across routers spanning the globe. Backbone routing algorithms hence have to work with incomplete topology information.

Given the above considerations, it is infeasible to achieve multi-path routing in a deterministic framework, even with complete knowledge of network. It thus does not bode well for achieving multipath routing with incomplete knowledge. Our viewpoint is that forsaking deterministic algorithms relaxes consistency constraints, which are critical for their functioning. This leads us to a probabilistic routing framework.

7 Elements of an Effective RL Framework

Our approach to reachability routing exploits the inherent semantics of Markov decision processes (MDPs) as modeled by reinforcement learning algorithms. RL embodies three fundamental aspects [22] of our routing context. First, RL problems are *selectional* – the task involves selecting among different actions. Second, RL problems are *associative* – the task involves associating actions with situations. Third, RL supports *learning from delayed rewards* – reinforcement about a particular routing decision is not immediate and hence supervised learning methods are not suitable.

Before developing the elements of an RL framework, we need to model our problem domain as an RL task. An 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 case, the states are the routers and an action denotes the choice of the outgoing link. Notice that state transitions here are deterministic, since a physical link always interconnects the same two routers. This means that the stochastics of the problem primarily emerge from any non-determinism in the router's policy of choosing among a set of outgoing links. This is in sharp contrast to typical RL settings where the choice of the action *and* the state-transition matrix are stochastic.

Rewards are supplied by the environment and the value function describes the goal imposed on the RL algorithm. The value function typically tries to maximize or minimize an objective function. For instance, learning shortest-cost paths by maximization can be modeled by negating link costs and setting the value function to be equal to the cumulative path cost. To model basic reachability routing, all rewards are set to zero except for the egress link leading to the destination, which is set to 1. To model cost-dependent reachability routing, rewards are set to reflect the quality of the paths.

Given the modeling of an 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. This section studies ways of configuring each of these aspects and their impact on a reachability routing framework.

7.1 Information Gathering

Since RL algorithms employ evaluative feedback, all of them rely on sample episodes to gather information. While data traffic routing is episodic in its behavior, the information carried by packets is not expressive enough for RL algorithms. Data packets only contain the source *host* address and, in particular, do not carry any information about the path traversed to reach the destination. Since it is not possible to determine the ingress router from the source host address and because routers maintain routing tables only to other *routers*, the information carried in a data packet is insufficient to aid routing. Furthermore data packets do not contain any fields that can carry path-cost metrics that are required for generating reinforcement signals in cost-dependent reachability routing. This argument forms the basis for explicit information carriers. In current networks this is achieved by routing messages. In the context of RL algorithms, the same effect is achieved by ants.

Even with explicit information carriers, it is imperative to distinguish data traffic patterns from ant/control traffic patterns. Simple-minded schemes like Q-routing fall into the trap of learning about *only* those paths traversed by data traffic. Ideally the construction and maintenance of a routing table should be independent of the data traffic pattern, since it is well known that the data traffic on the Internet is highly skewed in its behavior [7]. While it may be argued that reinforcing well used paths ('greasing') is desirable, it does not lead to reachability routing or even multi-path routing.

The ant algorithms described in Section 4.3 can be viewed as a mechanism to segregate control traffic from data traffic patterns. The parameters of interest are the rate of generation of ants, the choice of their destinations, and the routing policy used for ants. Current network routing protocols generate routing messages periodically at a rate independent of their target environment. The signature pattern here is the information carried by the control traffic and not the rate of control traffic. This suffices because these are deterministic algorithms and rate merely influences the recency of the information. In contrast, RL algorithms perform iterative stochastic approximation and the rate of ant generation implicitly affects their convergence properties [10], and hence the quality of the learned routing tables. It is for this reason that considerable attention is devoted to tuning ant generation distributions. For instance, RL algorithms may selectively use a higher ant generation rate to improve the quality of routes to oft-used destinations.

The second parameter of interest is the choice of ant destinations. It may be argued that it is beneficial to use non-uniform distributions favoring oft-used destinations. For instance, in the client-server model prevalent in the current Internet, data traffic is inherently skewed toward servers. Intuitively, it appears that a non-uniform distribution favoring servers will lead to better performance. However, from the perspective of reachability routing, we would like to choose destinations that will provide the most useful reinforcement updates, which are not necessarily the oft-used destinations. In the absence of a model of the environment, a uniform distribution policy at least assures good exploration. Model-based RL algorithms studied later in this section have more sophisticated means of distributing ant destinations.

The policy used to route ants affects the paths that are selectively reinforced by the RL algorithm. If the goal of the RL algorithm is to do some form of minimal routing, it is beneficial to improve the quality of 'good' routes that have already been learnt. To achieve this, the ant routing policy is the same as the policy used to route data traffic. However, from a reachability routing perspective our goal is to discover all possible paths. Hence the policy used to route ants is independent of the data traffic carried by the network. It is interesting to note that cost-dependent reachability routing may be achieved by using a judicious mix of the above two routing policies. This is not as intuitive as it appears – see Observation 2 of the next section.

7.2 Credit Assignment Strategies

In the context of an RL framework, effective credit assignment strategies rely on the expressiveness of the information carried by ants. The central ideas behind credit assignment are determining the relative quality of a route and

Modeling the RL Problem	
	- States
	- Actions
	- Rewards
	- Value functions
Information Gathering	
	- Rate of ant generation
	- Choice of ant destinations
	- Ant routing policy
Credit Assignment Strategies	
	- What to reinforce
	- Backward directions
	- Forward directions
	- How much to reinforce
	- Defining update formulas
Models in RL	
	- For learning
	- For planning

Table 1: Characteristics of an RL formulation for reachability routing.

apportioning blame. In our domain, credit assignment creates a 'push-pull' effect. Since the link probabilities have to sum to one, positively reinforcing a link (push) implies negative reinforcements (pull) for other links. All the RL algorithms studied earlier use positive reinforcement as the driver for the push-pull effect.

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 a reinforcement for the link along which the ant arrived [21] (backward learning). Asymmetric link costs – e.g., in technologies like xDSL, cable modems — can be accommodated by using the reverse link costs instead of forward link costs.

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 [10]. The ant carries a stack where each element of the stack describes a node, the accumulated path cost to reach that node and the chosen outgoing interface. When the ant reaches its destination, it is turned back to its source. During the backtracking phase, the information carried by the ant reinforces the appropriate interface in the intermediate nodes.

The above discussion has concentrated on 'what to reinforce,' rather than 'how much to reinforce.' For cost c accumulated by an ant, most RL algorithms generate a reinforcement update that is proportional to $\frac{1}{f(c)}$ where $f(c)$ is a non-decreasing function of c. Sophisticated approaches may include local models of traffic/environment to improve the quality of the reinforcement update. Di Caro and Dorigo [10] provide an elaborate treatment of this subject.

7.3 Models in RL Algorithms

The primary purpose of building a model is to improve the quality of reinforcement updates. For instance, in a simple model, a router may maintain a history of past updates and rely on this experience to generate different reinforcement signals, even when given the same cost update. This is an example where the router has a notion of a 'reference reward' that is used to evaluate the current reward [22]. More sophisticated models — such as **actor-critic** — have an explicit 'critic' module that is itself learning to be a good judge of rewards and reinforcements.

A model-based approach can also be used for directed exploration, where the model suggests possible destinations and routes for an ant. In RL literature, this is referred to as the use of a model for *planning*. Here, it is important that the model track the dynamics of the environment faithfully. An inconsistent model can be worse than having no model at all, in particular, when the environment *improves* to become better than the model and the model is used for exploration. Of the RL algorithms studied in this paper, Q-routing and the algorithms of Subramanian et al. [21] are model-free. The stigmergetic framework of [10] builds localized traffic models to guide reinforcement updates.

While a model-based approach improves the quality of reinforcement updates, it effectively violates the notion of independent credit assignment. The main benefit of forsaking independent credit assignment is that we can maintain context across learning episodes. However, we have to be careful to ensure that convergence of the RL algorithm is not compromised. Table 1 summarizes the main characteristics of RL algorithms that have to be configured for a reachability routing solution.

8 Observations

We now present a series of observations identifying research issues in the application of RL algorithms to the reachability routing problem.

1. Many RL algorithms model their environment as either a Markov decision process (MDP) or a partially observable Markov decision process (POMDP). Both MDPs and POMDPs are too restrictive for modeling a routing environment. For instance, to avoid network loops the choice of an outgoing link made at a node depends on the path used to arrive at the node. This form of *hidden state* has been referred to as *Non-Markov hidden state* [16] and can be solved with additional space complexity. However, there are other hidden state variables (e.g., downstream congestion) that cannot be locally observed and which need to be factored into the routing decision. While additional information qualifiers may improve the quality of the routing decision, the dynamics of the network, the high variance of parameters of interest, and communication latencies make it practically impossible to eliminate hidden state. Hence, any effective RL formulation of the routing problem has to work with incomplete information.

2. Since RL algorithms work by iterative improvement, the rate of reinforcement updates and the magnitudes of the updates affect their convergence. Consider the 'velcro' topologies shown in Fig. 7. Ideally, in Fig. 7 (left) we would like a multi-path routing algorithm to distribute traffic in a 1:10 ratio between the direct $A \rightarrow B$ path and the other paths. In Fig. 7 (right) we desire a multi-path routing algorithm that can distribute traffic in a 2:1 ratio between the direct $A \rightarrow B$ path and the other paths.

   In Subramanian et al.'s formulation of the RL algorithm [21], uniform ants are used for exploration and regular ants are used as shortest-path finders. Since uniform ants explore all links with equal probability, in Fig. 7 (left) they will carry high cost updates for the 'loopy' path with high probability. The probability of carrying the correct path cost update of 10 can be made infinitesimally close to zero. On the other hand, regular ants will discover and converge to the path cost of 10 along the loopy part of the graph. To achieve our goal of multi-path routing we can use a combination of uniform ants and regular ants, relying on the former to provide the correct cost update for the direct $A \rightarrow B$ path and the latter for the loopy path. In this example the learning problem has been effectively decomposed into two disjoint subtasks, each of which is suited for learning by a different type of ant.

   On the other hand, in Fig. 7 (right), regular ants will converge to the direct $A \rightarrow B$ path. Since uniform ants are incapable of deriving correct cost updates for the loopy path, both uniform and regular ants reinforce the direct $A \rightarrow B$ path. In this topology, even a mix of regular and uniform ants is incapable of achieving multi-path routing.

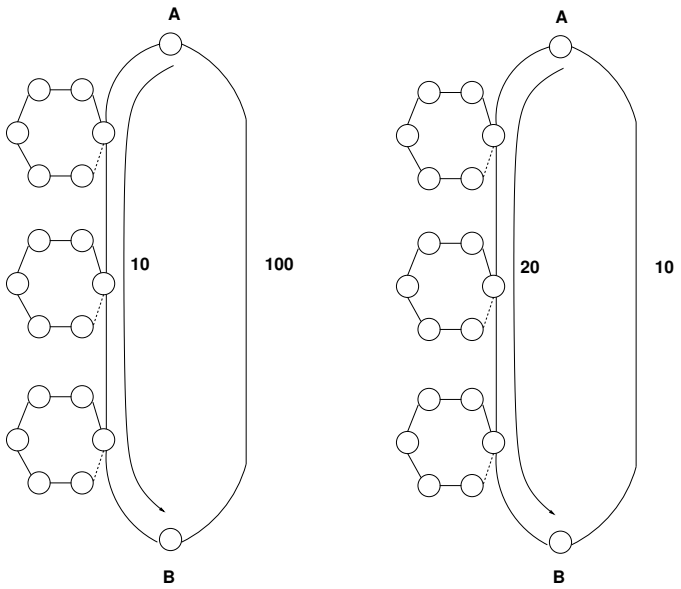

Figure 7: Two 'velcro' topologies that require substantially different types of information gathering mechanisms.

The AntNet algorithm [10] recognizes that loops can cause inordinately high cost updates and eliminates them by destroying the cost update. This effectively impacts the rate of received updates. While the beneficial side-effect of this strategy is that it reduces network traffic, its performance is no different from that of uniform ants which carry very small updates. The drastically reduced rate of correct updates equates the reinforcement effect to that of uniform ants.

Thus, information-gathering mechanisms in a network should take into account the rate-based nature of RL algorithms. Even seemingly intuitive exploration mechanisms (uniform ants) can be misled.

3. The above observation leads us to the question: can an RL algorithm adapt its behavior based on its 'position' within the network? This requires a) additional information qualifiers to determine the position, and b) co-ordinating the operation of the RL algorithm executing at distinct nodes [12]. For instance, an RL algorithm may provide an additional information qualifier that tracks the rate of successful explorations. This information can be used to cluster the nodes into equivalence classes, each of which involves co-ordinated reinforcement. In Fig. 7, the rate of successful explorations along the loopy paths can guide the nodes into co-ordination.

4. The reader may recall that our discussion so far has focused on soft reachability. To achieve hard reachability, each router needs to know the predecessor path of an arriving packet. As mentioned earlier, practical considerations preclude data packets from carrying this information. The question here is: can we do better than soft reachability using an RL algorithm?

   For instance, given a finite number of memory slots in a data packet header, can we embed router identifiers of sufficient resolving power that can eliminate certain categories of loops? We can pose this as a problem of maximizing/minimizing the probability of achieving a goal function. Goal functions may be eliminating more loops, eliminating larger/expensive loops, or exiting a loop, once entered.

5. RL algorithms typically use positive reinforcement as a driver for credit assignment. In this mode of operation, link probabilities go down (are negatively reinforced) only when some other link receives a positive reinforcement. Is it possible to have a primarily negative mode of reinforcement? This is harder than it appears.

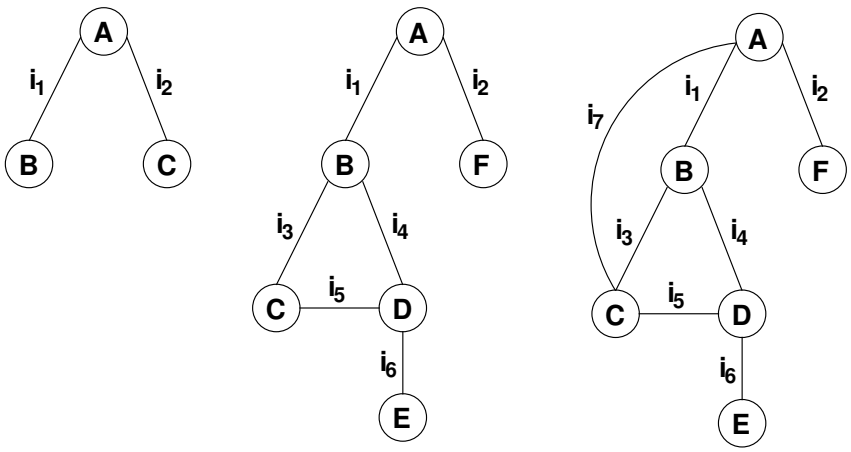

Figure 8: Three topologies for assessing the amount of information qualification required for negative reinforcement.

To see why, consider what negative reinforcement might mean in a reachability routing framework. While positive reinforcement merely indicates that a destination *may be reached* via the outgoing link, negative reinforcement implies that the destination *definitely cannot be reached* without encountering a loop. Note that reachability routing is fundamentally a binary process — destinations are either reachable or not reachable. Reinforcement of reachable destinations affords significant laxity in the decision process whereas non-reachability is necessarily definitive.

Such a drastic form of negative reinforcement constitutes *instructive feedback* as opposed to *evaluative feedback*, since we are informing the algorithm what the right answer should be. With evaluative feedback, shades of (positive) reinforcement can exist which will interact to ensure the convergence of the RL algorithm. With instructive feedback, we should be careful to ensure that convergence properties are not affected by incorrect instructions. This means that the onus is on us to explore all alternatives before concluding that a link does not lead to a given destination.

To create an RL algorithm that uses negative reinforcement, let us study situations where definite conclusions can be made about the non-reachability of destinations. The simplest case is illustrated in Fig. 8 (left). Here, if an ant originating at A and destined for B ends up at node C, C can send a negative reinforcement signal indicating that B is not reachable via i_2. The negative reinforcement signal relies on the fact that node C can clearly determine that it is a leaf node and is not the intended destination. Hence, no loop-free path to node B can be found via node C. At a leaf node, knowledge of the destination is sufficient to assess the availability of a loop-free path.

This simplistic scheme is not capable of resolving paths in Fig. 8 (middle). Consider an ant originating at node A and destined for node E. If the ant traverses the path $\prec A, i_1 \succ, \prec B, i_4 \succ, \prec D, i_5 \succ, \prec C, i_3 \succ$, node B can determine that the ant has entered a loop and send a negative reinforcement signal to node C. The negative reinforcement signal tells node C that destination E is not reachable via link i_3, which is incorrect. The observation here is that the destination address alone is insufficient to qualify the negative reinforcement signal.

Let us augment the information maintained by the routing algorithm to include source addresses. The routing table thus contains entries that associate a source-destination address pair with an outgoing link, a scheme called *source-destination* routing. If we employ source-destination routing on the network in Fig. 8 (middle), B's negative reinforcement signal effectively tells node C that link i_3 (in the C to B direction) cannot be used for a packet originating at A and destined for E, which is correct. Likewise, the reader can verify that the

counter-clockwise loop from B to D through C can be resolved.

Before we adopt this as a solution, consider Fig. 8 (right). In this case, a negative reinforcement signal from B indicates to C that link i_3 cannot be used for a packet from A destined for E, which is incorrect, since a packet from A arriving at C on link i_7 can indeed use outgoing link i_3. In this case, we need an additional information qualifier (the incoming link) to resolve the negative reinforcement signal.

The astute reader may have observed that even this information qualification is insufficient; technically, the entire predecessor path may be required to resolve negative reinforcement signals. The issue of interest here is, for a given topology, is it possible to adaptively determine the 'right' information qualifier to resolve negative reinforcement signals?

6. Reinforcement learning supports a notion of hierarchical modeling (e.g., see [11]) where different subnetworks/domains have different goals (value functions). Is it possible to have an information communication mechanism so that this hierarchical decomposition is automatic? Fundamentally, can RL be used to suggest better organization of communication networks?

7. Is it possible to classify/qualify graphs based on the expected performance of RL algorithms? Akin to Observation 3 above, this information can then be used for specializing RL algorithms for specific routing topologies. For instance, in the velcro topology studied earlier, the RL algorithm operating in the loopy part can determine that uniform ants have a low probability of reaching the destination and change its behavior in only this part of the network. Such a scheme can be combined with the previous observation to create a more fluid definition of hierarchical decompositions.

8. The Internet's routing model evolved from its original co-operative underpinnings to a competitive model, owing to commercial interests. Each administrative domain uses an internal value function that are not communicated to their peer domains. It is of scientific interest to determine the value function employed by a routing protocol.

   Inverse reinforcement learning (IRL) [19] is a recently developed framework that can be used to address precisely this question. As the name suggests, IRL seeks to reverse-engineer the value function from a converged policy. IRL's assumption that the policy is optimal with respect to *some* metric generally holds true in the routing domain. Operationally, IRL can be used on the temporal and spatial distributions of probe packets traversing an unknown network – which is treated as a black box.

   If IRL can be used to approximate the value function, it would enable differentiated services routing, without requiring any changes to the existing backbone routing infrastructure. An AS can observe the end-to-end behavior of another AS and use it to improve the performance for its own clients. From a game-theoretic perspective, this raises interesting questions of how competition and co-operation can co-exist among agents conducting inverse reinforcement learning.

9 Design and Implementation of a Reachability Routing Algorithm

As a demonstrator of the many ideas presented in this paper, we present the implementation and evaluation of a multi-path reachability routing algorithm in the reinforcement learning framework. The primary design objective here is to achieve cost-sensitive multi-path forwarding while at the same time, eliminating the entry of loops as much as possible. We begin with the uniform ants version of the Subramanian et al. [21] routing algorithm (as it is designed with multi-path routing in mind) and describe a series of improvements, culminating in a new model-based reachability routing algorithm.

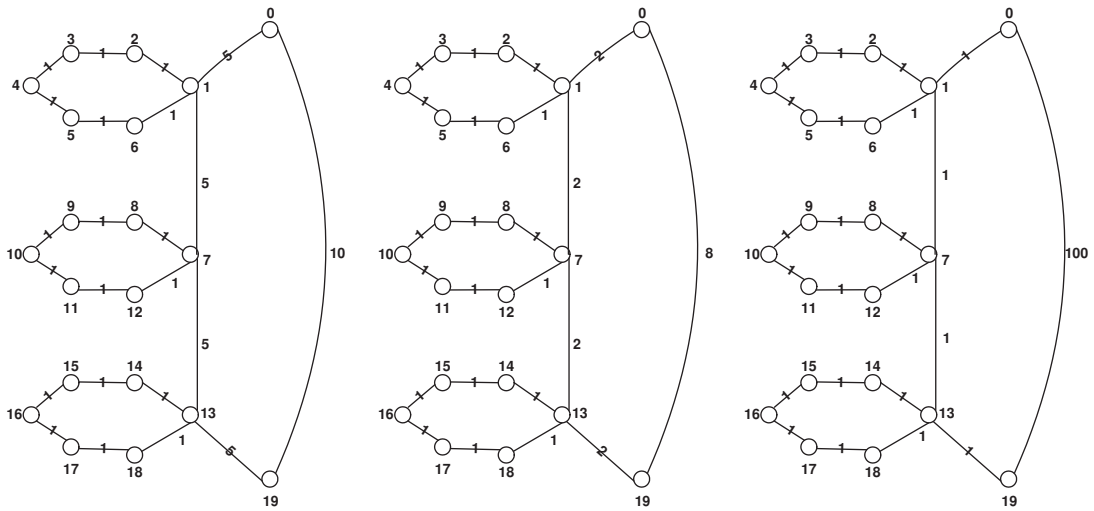

Figure 9: Uniform ants tend to reinforce a path with the least amount of decision making. Such a path may be the cheapest (left), among one of many cheapest paths (middle), or actually the costliest path (right).

Let us consider how the uniform ants algorithm behaves in the three 'velcro' topologies of Fig. 9. 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 exist high-quality loop-free subpaths 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. 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 multi-path forwarding and metamorphing 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 Fig. 9, 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.

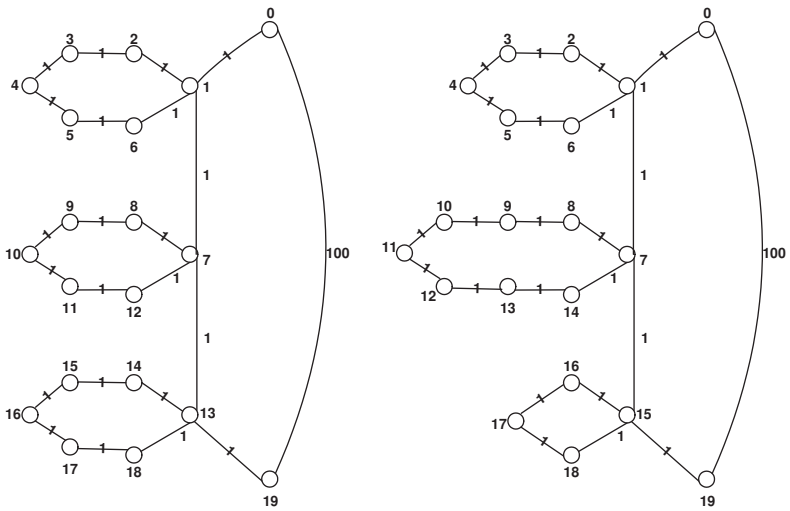

Figure 10: Two velcro topologies with substantially different sizes of subgraphs rooted at the fulcrum nodes.

Notice that it would not do to accumulate the statistics for *all* ants passing through a given node and intended for a given destination. To see why, consider Fig. 10 — both graphs here have fulcrums but the relative sizes of the subgraphs situated at the fulcrums are different. In Fig. 10 (right), the loop situated at node 7 is considerably larger than the one situated at node 15. Consider an ant destined for node 0 of Fig. 10 (right) and generated by one of the nodes in the subgraph rooted at node 7. Let us examine the statistics collection from the viewpoint of the fulcrum node 7. The ant has a considerable probability of looping back into the subgraph after visiting node 7, where it will eventually reach its sender again, and be absorbed. From node 7's point of view, through which the ant has passed, this would count as an ant that successfully reached its destination; leading to an incorrect reinforcement of an interface that is actually entering a loop. To circumvent this problem, it is imperative that node 7 maintain statistics about only those ants that it generates.

The role of the statistics table is to serve as a discriminant function for the choices indicated by the routing table. While the routing table entries reflect the reinforcement provided by the uniform ants, the statistics table effectively allows us to discard those interfaces that had a 100% probability of leading into a loop (assuming lossless links). We thus use the statistics table to reduce the scope of probability distribution to only those interfaces that have a $< 100\%$ probability of a loop-free path. The reader might argue that we can go a step further and deterministically choose the interface that has the lowest probability (from the statistics table) of leading into a loop. Besides going against the spirit of multi-path routing, this approach spells danger in transient network conditions where larger loops envelop the fulcrum loops, and once a packet enters the larger loop, it might never reach its intended destination. *In other words, one should be careful that improvements to reinforcement learning do not collectively constitute the realization of a deterministic algorithm.*

Two final improvements over the uniform ants algorithm of Subramanian et al. [21] are included in our implementation. The approach given in [21] reinforces all subpaths 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. Our 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 (e.g., see [10] for a different perspective on such 'selective' subpath reinforcement). And finally, in Subramanian et al.'s original formulation, the probabilities for forwarding uniform ants are apportioned among all interfaces, including the interface along which the ant arrived. In our 'no send-back' implementation, the incoming interface will not be chosen as a possible outgoing interface unless the node is a leaf node.

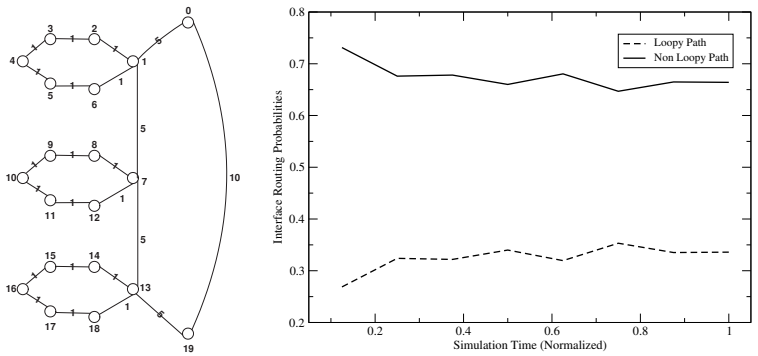

Figure 11: (left) A velcro topology with cost proportion 2:1 between the left and right paths, from node 0 to node 19. (right) Results of model-based reinforcement learning, revealing the convergence to the 2:1 cost ratio.

9.1 Results

The implementation choices outlined above lead to a new model-based approach to achieving cost-sensitive reachability routing. In contrast, [21] uses a model-free approach, which does not achieve multi-path routing in 'loopy' topologies. While [10] presents a model-based approach to routing, the model is used to improve routing decisions by taking into account local traffic distributions at each node. This approach also does not achieve multi-path routing in the topologies considered here.

In this section, we present simulation results from our implementation that clearly show the performance benefits of our approach. We focus on topologies modeled after the velcro graph — topologies with significant amount of decision making — 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 an 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.

Recall that our approach starts with the uniform ants algorithm of Subramanian et al. [21] and adds the three crucial components of (i) the statistics table, (ii) no subpath reinforcement, and (iii) no send back. In all the simulations presented here, we begin by apportioning the probabilities among all available interfaces, conduct the reinforcement updates and, when the statistics have stabilized begin employing the statistics table in conjunction with the learned routing probabilities. This switching threshold was chosen to allow stabilization of the routing table entries (as determined by the conventional reinforcements) and facilitates a meaningful comparison.

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 [23], and has been used in several networking courses to model routing algorithms.

We begin with the simple velcro topology shown in Fig. 11 (left) where the two paths from node 0 to 19 have a 2:1 cost ratio, if the loops in the left path are avoided. As Fig. 11 (right) shows, the uniform ants initially prefer the loop-free path by a ratio of 3:1. When the statistics table is employed, this ratio gets moderated to 2:1 which more accurately reflects the cost ratios of the two paths. For a more dramatic demonstration of the effect of the statistics table, let us turn to the topologies shown in Fig. 12 (left).

In the first topology of Fig. 12, the cost ratio is 1:25 (in favor of the loopy path). In the second topology, the cost ratio is 1:2.5. As the results show, both graphs demonstrate a marked change in the routing probabilities at switchover

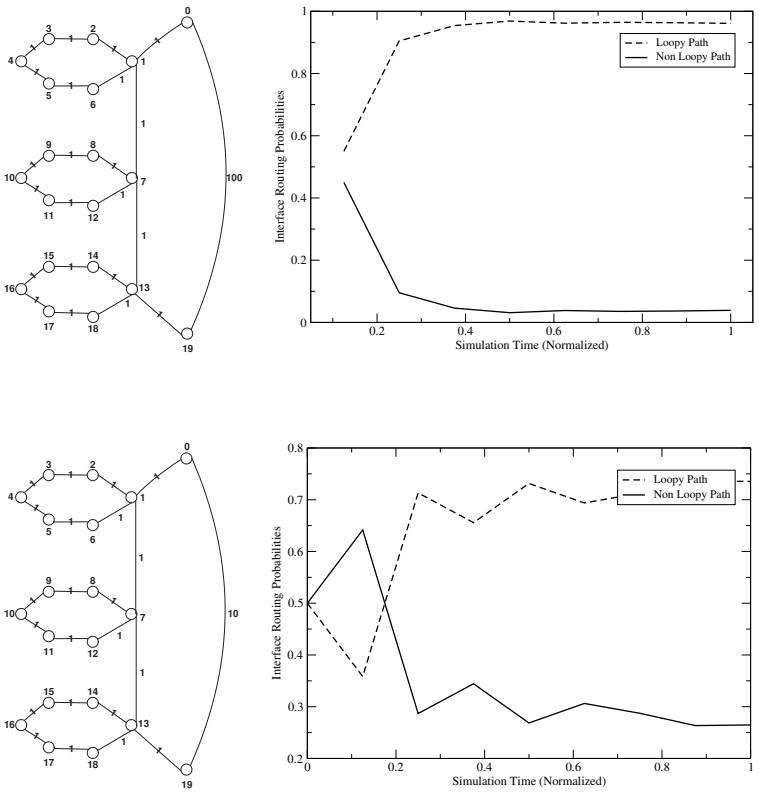

Figure 12: (top left) A velcro topology with cost proportion 1:25, and (top right) corresponding results for model-based reinforcement learning. (bottom left) A topology with cost proportion 1:2.5, and (bottom right) corresponding results.

time (0.125 on the 'Normalized Time' axis). The effect in Fig. 12 (top) 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.

Fig. 12 (bottom) 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* non-loopy path 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 non-loopy path. After the statistics table goes into effect, we observe a dramatic flip in the routing probabilities, which then converges to the ratio of the path costs.

Fig. 13 shows a topology similar to what we have considered so far, except that both the loopy 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. Fig. 14 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.

Fig. 15 describes a new topology and an experiment designed to show the importance of avoiding subpath reinforcement. Fig. 15 (middle) shows the results with subpath reinforcement and reveal that both paths from node 5 to node 2 are reinforced near equally, even though the path employing the direct link to node 4 has a higher cost. When subpath reinforcement is removed, as Fig. 15 (right) shows, the roundabout path gets a greater reinforcement, as desired.

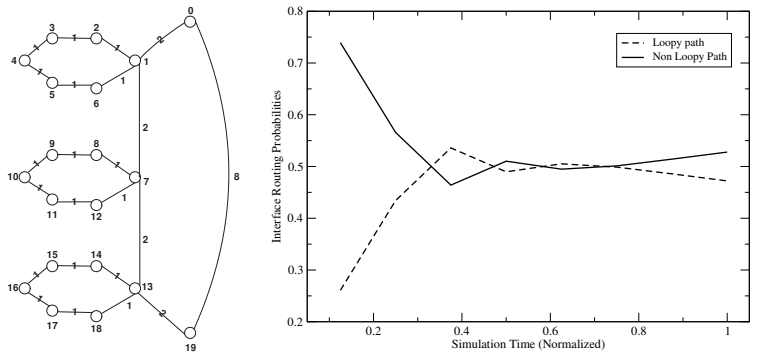

Figure 13: (left) A velcro topology with equal cost paths from node 0 to node 19 and (right) corresponding results.

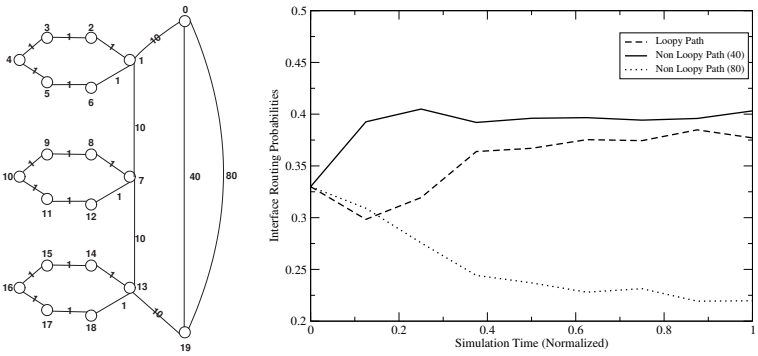

Figure 14: (left) A velcro topology with three paths from node 0 to node 19 and (right) corresponding results.

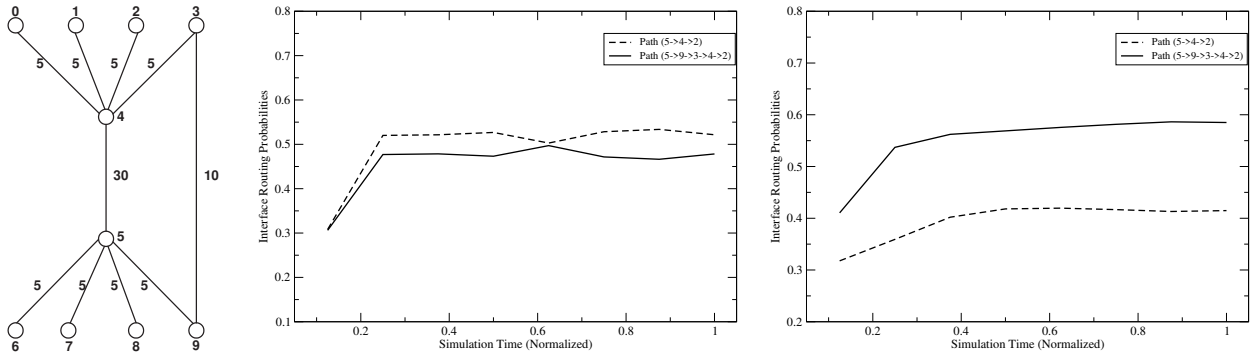

Figure 15: (left) A 'dumbbell' topology where the direct path from node 5 to node 4 is costlier than the round-about path. (middle) Using subpath reinforcement does not capture this aspect, whereas (right) avoiding subpath reinforcement learns the correct apportionment of probabilities.

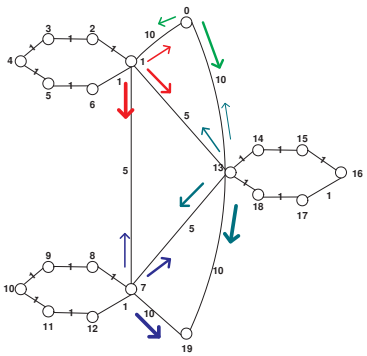

Figure 16: A topology with loops at many levels, the goal is to learn to route to node 19. The results of model-based reinforcement learning are superimposed as arrows at the various nodes. Different colors indicate different choices of starting nodes and the thickness of arrows indicate greater probabilities along those interfaces (for a given choice of starting node).

Finally, Fig. 16 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. 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.

10 Conclusion

In this paper, 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. Several possibilities for future work are now being investigated, many along the ideas presented in Sec 8.

References

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

[2] D.P. Bertsekas. Distributed Dynamic Programming. *IEEE Transactions on Automatic Control*, Vol. 27:pages 610–616, 1982.

[3] D.P. Bertsekas and R. Gallager. *Data Networks*. Prentice Hall, Englewood Cliffs, NJ, 1992. Second Edition.

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

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

[6] 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.

[7] 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.

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

[9] T. Dean and M. Boddy. An Analysis of Time-Dependent Planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI'88)*, pages 49–54. AAAI/MIT Press, 1988. St. Paul, MN.

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

[11] T.G. Dietterich. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *Journal of Artificial Intelligence Research*, Vol. 13:pages 227–303, 2000.

[12] C. Guestrin, M. Lagoudakis, and R. Parr. Coordinated Reinforcement Learning. In *Machine Learning: Proceedings of the Nineteenth International Conference (ICML 2002)*. Morgan Kaufmann, San Francisco, CA, July 2002.

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

[14] 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.

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

[16] A.K. McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, Department of Computer Science, University of Rochester, 1995, revised 1996.

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

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

[19] A.Y. Ng and S.J. Russell. Algorithms for Inverse Reinforcement Learning. In *Machine Learning: Proceedings of the Seventeenth International Conference (ICML 2000)*, pages 663–670. Morgan Kaufmann, San Francisco, CA, June 2000.

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

[21] 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.

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

[23] 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.

[24] 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.