A Distributed Reinforcement Learning Scheme for Network Routing

Michael Littman Justin Boyan

July 1993
CMU-CS-93-165

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

In this paper we describe a self-adjusting algorithm for packet routing, in which a reinforcement learning module is embedded into each node of a switching network. Only local communication is used to keep accurate statistics at each node on which routing policies lead to minimal delivery times. In simple experiments involving a 36-node, irregularly connected network, this learning approach proves superior to a nonadaptive algorithm based on precomputed shortest paths.

The authors would like to thank for their support the Bellcore Cognitive Science Research Group, the National Defense Science and Engineering Graduate fellowship program, and National Science Foundation Grant IRI-9214873.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of Bellcore, the National Science Foundation or the U.S. Government.

Keywords: Reinforcement learning, Network routing

1 Introduction

We present an algorithm for routing packets efficiently in an irregularly-connected communication network with unpredictable usage patterns. The algorithm, related to certain "distributed" packet routing algorithms [4, 5], must learn a routing policy which balances minimizing the number of "hops" a packet will take with the possibility of congestion along popular routes. It does this by experimenting with different routing policies and gathering statistics about which policies minimize total delivery time.

Although in principle such an approach could be very expensive in terms of learning time and storage space, the algorithm we describe here is a variant of a reinforcement learning algorithm called *Q-learning* [7] which adapts to changes in network traffic and requires little more space than that needed to represent a complete routing policy. The application of Q-learning here differs from its traditional use in that the network is actually constructed from a distributed *collection* of learners, each of which is responsible for a portion of the problem. This approach appears to be very effective in routing packets efficiently under high load.

The experiments in this paper were carried out using a discrete event simulator to model the motion of packets through a local area network. Packets are periodically introduced into the network at a random node with a random destination. Multiple packets at a node are stored in an unbounded FIFO queue; however, we set a limit on the total number of packets active in the network at a time, generally 1000. In unit time, a node takes the top packet in its queue, examines its destination, and chooses a neighboring node to which to send the packet. A packet sent directly to its destination node is removed from the network immediately.

2 Routing as a Reinforcement Learning Task

A packet routing policy answers the question: to which adjacent node should the current node send its packet in order to get it as quickly as possible to its eventual destination? Since the policy's performance is measured by the total time taken to deliver a packet, there is no "training signal" for directly evaluating or improving the policy until a packet finally reaches its destination. However, using an idea from the field of reinforcement learning, we can update the policy more quickly and using only local information.

In our learning scheme, the policy is distributed throughout the network

as follows: each node keeps an estimate, for every neighbor/destination pair (y, d), of how long it takes for a packet with destination d to arrive if first sent to neighbor node y. When a node x is asked to route a packet, it sends it to that neighbor $\bar{y}$ which x estimates will have the lowest total delivery time. Instead of then waiting for the packet to reach d before updating the policy, x queries $\bar{y}$ to find out how long $\bar{y}$ expects the given packet to take to get to d. Since $\bar{y}$ is presumably closer to d, its estimate is considered more accurate and thus can be used to update x's delivery time estimate.

More precisely, let $Q_x(\bar{y}, d)$ be the time that node x estimates it takes to deliver a packet P bound for node d by way of x's neighbor node $\bar{y}$, including any time that P would have to spend in node x's queue.[1] Upon sending P to $\bar{y}$, x immediately gets back $\bar{y}$'s estimate for the time remaining in the trip, namely

$$Q_{\bar{y}}(\bar{z}, d) = \min_{z \in \text{neighbors of } \bar{y}} Q_{\bar{y}}(z, d)$$

If the packet spent q units of time in x's queue, then x can revise its estimate as follows:

$$\Delta Q_x(\bar{y}, d) = \eta \, (\overbrace{Q_{\bar{y}}(\bar{z}, d) + q}^{\text{new estimate}} - \overbrace{Q_x(\bar{y}, d)}^{\text{old estimate}})$$

where η is a "learning rate" parameter (0.7 in our experiments).

In the field of reinforcement learning, the Q-function $Q_x(y, d)$ is often approximated by a neural network (see e.g. [3, 6]); this can allow the learner to incorporate diverse parameters of the system, such as local queue size and time of day, into its distance estimation. For the implementation described here, however, we represented Q as a table.

3 Results

We tested our routing algorithm on a variety of network topologies, including the 7-hypercube, a 116-node LATA telephone network, and an irregular 6×6 grid. Varying the level of network traffic, we measured the average delivery time for packets in the system after learning had settled on a routing policy, and compared these delivery times with those given by a conventional routing scheme based on shortest paths. The result was that in all cases, the learning algorithm was able to sustain a higher level of network traffic than the non-learning one.

[1] We denote the function by Q because it corresponds to the "Q-function" used in the reinforcement learning technique of Q-learning [7].

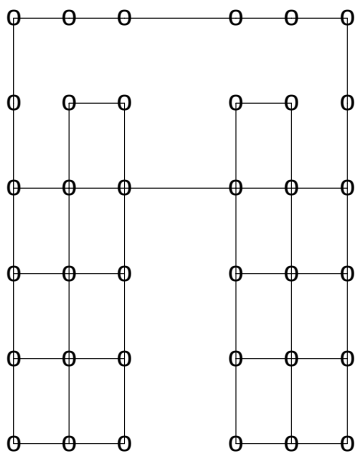

Figure 1: Network topology

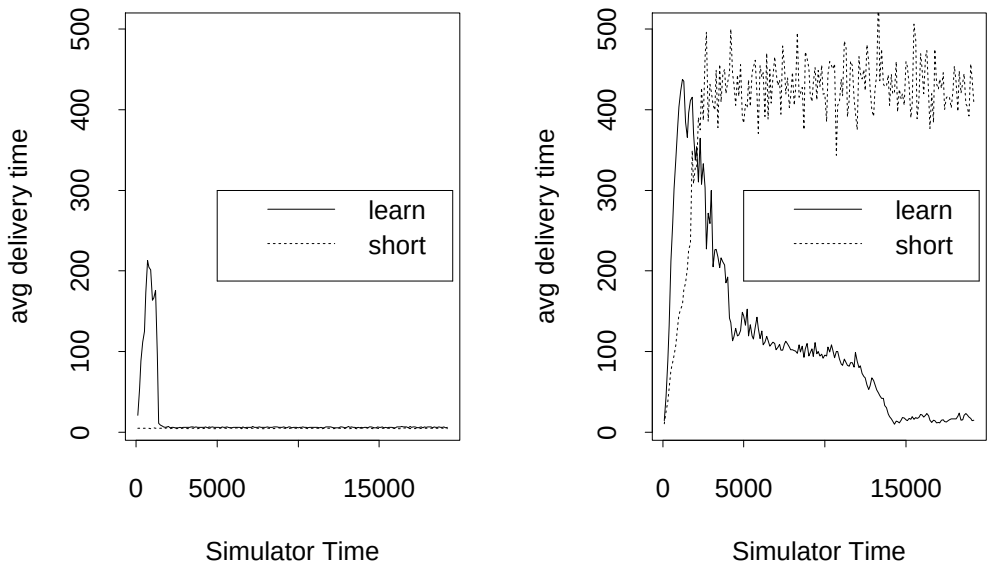

Figure 2: Performance under low load and high load

Here we present the results for the irregular grid network (pictured in Figure 1). Under conditions of low load, the network learns fairly quickly to route packets along shortest paths to their destination.[2] The performance vs. time curve plotted in the left part of Figure 2 demonstrates that our routing algorithm, using no prior knowledge of the network topology, learns to route about as well as the shortest path router, which performs optimally in low load.

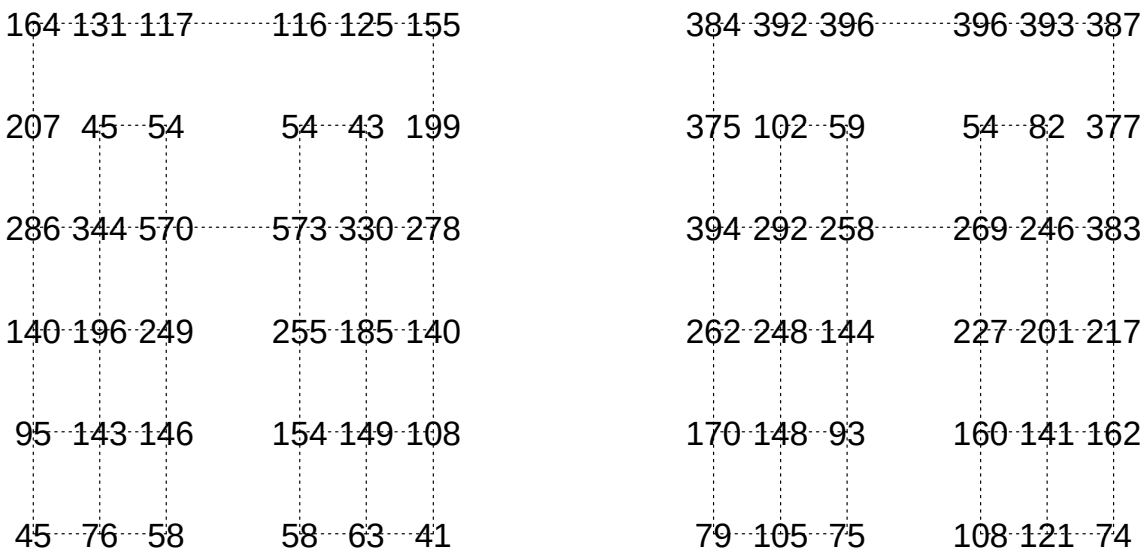

Figure 3: Policy summaries: shortest path and learned under high load

As network traffic increases, however, the shortest path routing scheme is far from optimal: it ignores bottlenecks and soon floods the network with packets. The right part of Figure 2 plots performance vs. time for the two routing schemes under high load conditions: while shortest path is unable to tolerate the packet load, our algorithm learns an efficient routing policy. The reason for the learning algorithm's success is apparent in the "policy summary diagrams" in Figure 3. These diagrams indicate, for each node under a given policy, how many of the 36×35 point-to-point routes go through that node. In the left part of Figure 3, which summarizes the

[2] In fact, it is interesting to note that the Q-learning update rule is mathematically very much like the well-known Bellman-Ford shortest paths algorithm [1, 2], except our path relaxation steps are performed asynchronously.

shortest path routing policy, two nodes in the center of the network (labelled 570 and 573) are on many shortest paths and thus become congested when network load is high. By contrast, the diagram on the right shows that our algorithm, under conditions of high load, has learned a policy which routes some traffic over a longer than necessary path (across the top of the network) so as to avoid congesting the nodes in the center of the network.

Our basic result is captured in Figure 4, which compares the performances of the shortest path policy and learned policy as the network load increases. Each point represents the median over three trials of the mean packet delivery time (after learning has settled). When the load is very low, our learning algorithm routes nearly as efficiently as the shortest path policy. As load increases, the shortest path policy leads to exploding levels of network congestion, whereas the learning algorithm continues to route efficiently. Only after a further significant increase in load does the distributed learning algorithm, too, result in congestion.

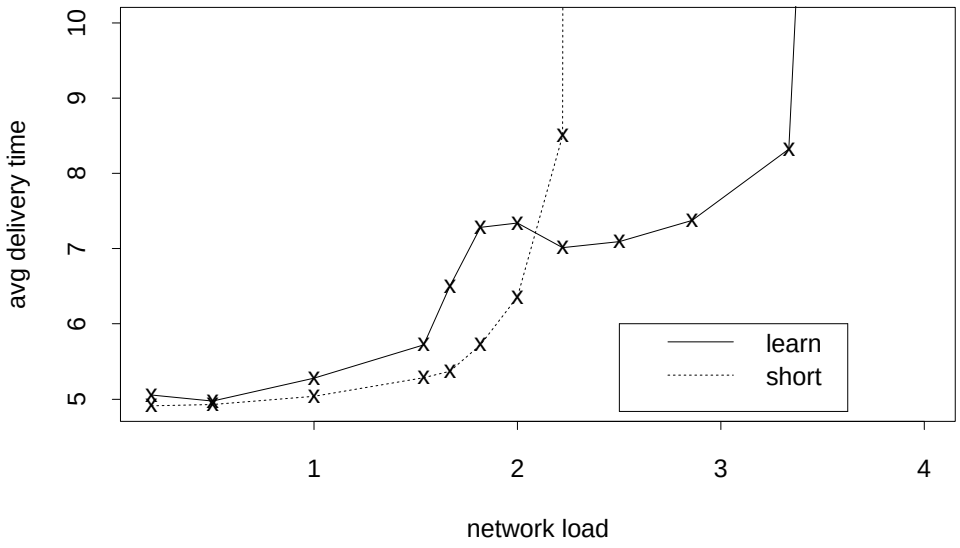

Figure 4: Delivery time at various loads

4 Conclusion

In this paper, we have exhibited a learning algorithm which, without having to know in advance the network topology and traffic patterns, and without the need for any centralized routing control system, is able to discover an efficient routing policy. Although the simulations described here are not fully realistic from the standpoint of actual telecommunication networks, we believe that reinforcement learning algorithms such as this one could be effective for self-adapting routing and other practical tasks.

References

[1] R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958.

[2] L. R. Ford, Jr. *Flows in Networks*. Princeton University Press, 1962.

[3] L.-J. Lin. *Reinforcement Learning for Robots Using Neural Networks*. PhD thesis, School of Computer Science, Carnegie Mellon University, 1993.

[4] H. Rudin. On routing and delta routing: A taxonomy and performance comparison of techniques for packet-switched networks. *IEEE Transactions on Communications*, COM-24(1):43–59, January 1976.

[5] A. Tanenbaum. *Computer Networks*. Prentice-Hall, second edition edition, 1989.

[6] G. Tesauro. Practial issues in temporal difference learning. *Machine Learning*, 8(3/4), May 1992.

[7] C. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, 1989.