# An Adaptive Network Routing Strategy with Temporal Differences

**Yván Túpac Valdivia, Marley M. Vellasco, Marco A. Pacheco**

ICA: Applied Computational Intelligence Laboratory
Pontifícia Universidade Católica do Rio de Janeiro
Rio de Janeiro - BRAZIL
{yvantv, marley, marco}@ele.puc-rio.br

**Abstract**

This paper describes the TD-Routing, an adaptive algorithm for packet routing, based on the *Temporal Differences* TD(λ) method, and compares its performance with other routing strategies: *Shortest Path Routing*, Bellman-Ford and the Q-Routing. High and low network traffic conditions are considered. In contrast with other algorithms that are also based on Reinforcement Learning (RL), the TD-Routing is able to discover good policies for situations that present a reduction in network traffic. The performance of the proposed algorithm was evaluated within a benchmark network configuration of 16 nodes with different traffic conditions in different topologies. The simulations demonstrate that the TD-Routing outperforms other RL-based algorithms in terms of learning speed and adaptability.

**Keywords:** Reinforcement Learning, Packet Routing, Network Communication Networks

## 1. Introduction

One of the major problems for most communications networks lies in defining an efficient packet routing policy. The simplest routing policy uses the *Shortest Path Routing* algorithm [Floyd62] [Dijkstra59] and consists of sending the packet via the path that offers the minimum number of hops. This policy is not optimal for all network traffic conditions because it generates popular routes that can be flooded rapidly with packets, under high traffic conditions. In this condition, long packet queues are formed, resulting in an increasing in routing time. Routing strategies that are adaptive to network load conditions try to find alternative routes when the current routing performance decreases.

Basically, the routing problem can be summarized by the following question: *Which is the optimal route for a packet, given an origin and a destination, under the network's current conditions?*

The solution to this problem may be approached in two ways [Tanenbaum96]:

**Routing defined in the source node:** In the first approach, a packet generated at the origin, comprises all the necessary routing information. The easiest way to achieve this type of routing is to use a central observer, which is able to collect general information related to the status of the network.

Based on that information, optimal routes for all possible destinations are calculated and sent to all the nodes such that, in the moment that the packet is sent from its origin, it is carrying all its routing information.

**Routing determined at each node:** In this approach, each node in the network makes local routing decisions based just on local information, i.e., information on the status of the network that has been obtained from the neighbor nodes. In this case, a node receives a packet and has to decide the neighbor that must receive the packet. The routing problem becomes a local problem: *Which neighbor*

*should be used to send the packet in order to make it reach its destination as quickly as possible under the network's current conditions*? In this case, the information required for each node is [Tanenbaum96]:

- An estimate for each neighbor about how long it would take for the packet to reach its destination when it is sent via that neighbor,
- An heuristic procedure which is capable of making decisions based on the previously estimated information,
- A way of updating routing information that can keep up with the status of the network;
- A mechanism to transport the information to the one node to all the others on network.

## 2. Routing as a Reinforcement Learning Task

Routing policies aim to optimize the average overall routing time based on local or global information. If local information is used, there is no training signal for a direct evaluation of the performance of policy before the packet has reached its final destination, so it becomes difficult to apply *supervised learning* techniques [Littman93][Haykin98]. Furthermore, it is difficult to determine how a routing decision made on a single node may influence the network's overall performance. This fact fits into the *temporal credit assignment problem* [Watkins89]. *Reinforcement Learning* [Sutton98] deals with the routing problem in spite of the limitations presented by the lack of input/output patterns and by the temporal credit assignment problem.

In a Reinforcement Learning (RL) model, an agent interacts with its environment. This interaction occurs as a result of the agent's capacity to perceive current environment conditions and to select the actions that should be performed in that environment. An action that is carried out changes the environment, and the agent is aware of these changes through a scalar reinforcement signal supplied by the environment. The objective of a RL system is to learn the mapping of states into actions. The scalar reinforcement value obtained from a reinforcement function represents the system's immediate objective. The agent makes use of value functions to represent the quality of the actions in the long-term. The decision to take a specific action when it faces a given status depends on a policy that represents the agent's behavior [Sutton98]. Basically, a RL system may be represented by the diagram in Figure 1.

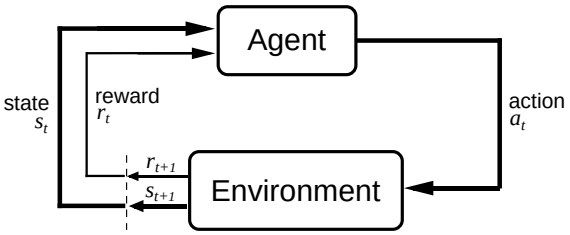


**Figure 1. Basic Reinforcement Learning system**

Therefore, a Reinforcement Learning system involves the following elements: an agent, an environment, a reinforcement function, an action, a state, a value function, and a policy. In order to obtain a network routing model usefully, it is possible to make that the following network's elements correspond to the basic elements of a RL system [Tupac00]

| Network Elements | RL System | |
|---|---|---|
| Network | Environment | |
| Each network node | Agent | |
| Delay in links and nodes | Reinforcement Function | $r_t$ |
| Estimate of total routing time for each target node | Value Function for each target node | $V_d(s_t)$ |
| Action of sending a packet | Action | $a(s_t)$ |
| Node through which the packet passes in time *t* | State in time *t* | $s_t$ |
| Local routing decision | Policy | $\pi$ |

**Table 1. Correspondences between a RL system and network elements**

Thus, the network may be modeled as a set of RL agent nodes, that interact among themselves and carry out a learning process based on the information obtained from these same nodes. The learning process is planned in the following manner: each node stores the value $V_t(s_t)$ that corresponds to the value function that estimates the routing time of a packet from the node $x$ (state $s_t$) to a $d$ destination node (end state $s_F$). When the node $x$ decides about the route of a packet, it sends this via the neighbor that presents the shortest estimated time to reach the destination. With the value obtained from the elected neighbor, the $x$ node updates its own value function by applying a learning algorithm. The diagram in Figure 2 illustrates this process.

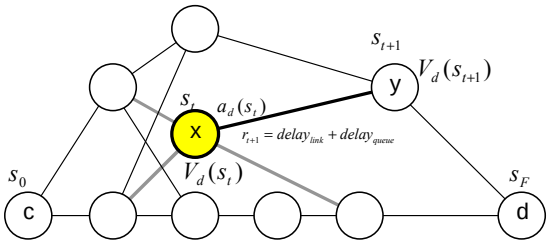


**Figure 2. Network routing process viewed as a Reinforcement Learning Task**

Let us consider a packet $P(c,d)$ whose origin is at $c = s_0$ and whose destination is at $s_F = d$. The origin state is $s_0$, the current state is $s_t = x$, and the end state is $s_F$. In its current state $s_t$, the node has a value $V(s_t)$, which represents the estimated routing time until it reaches $d$ destination node. As soon as the packet is sent from a node, reinforcement values are immediately received from the selected neighbor node, applying the following expression:

$$r_{T+1} = delay_{link}^{s_t,s_{t+1}} + delay_{BW}^{s_t,s_{t+1}} + delay_{queue}^{s_{t+1}} \quad (1)$$

that represents the time delay due to the distance $delay_{link}$ and to the link's bandwidth $delay_{BW}$ at the link $s_t$ - $s_{t+1}$, which are static values, and the delay in the queue $delay_{queue}$ of the selected node $s_{t+1}$, which is a dynamic value. These values are collected and utilized for updating the general value function. In order to standardize the symbol notations, let us $s_t$, $s_{t+1}$, $s_0$, $s_F$ and $r_{t+1}$ to be $x$, $y$, $c$, $d$ and $r$ respectively. For each update step, the difference $\delta_d$ is obtained among the successive states, which, according to the Temporal Differences method [Sutton88], corresponds to the following expression:

$$\delta_d = r + \gamma V_d(y) - V_d(x) \quad (2)$$

where $r$ is the reinforcement value obtained from (1), and $\gamma$ is a learning constant called the *discount parameter*.

## 3. Decision-making process

The decision-making process consists of selecting the neighbor through which the packet will be sent. Since the objective of routing is to make possible that the packets arrive to their destination as soon as possible, a *greedy policy* [Bellman57] is employed. Therefore, the decision consists of sending the packet via the neighbor node that presents the shortest end-to-end time delay estimate, so that it can proceed to state $s_{t+1}$. Greedy policies apply an intense exploration of the routes that have been found.

Occasionally, routes are selected applying non-greedy policies with the purpose of inserting a exploring factor for finding new alternatives. The algorithm makes a random decision on the neighbor by applying a given exploring ratio called $T_a$.

### 3.1. Updating Routing Information

Once the routing decision was made and the packet was already sent from node $x$ to the selected node $y$, it is necessary to update the routing information on node $x$ from node $y$ and the other neighborhoods nodes.

Although the updating process on TD(λ) algorithm is computationally more expensive its updating process, it is faster than the updating process of basic *Temporal Differences* algoritm, TD(0) [Sutton88], because it is applied *Eligibility Traces* [Singh96] and the λ parameter. For routing as a RL task *Eligibility Traces* mean how visited is a node when a greedy policy is applied. This parameter is applied for calculating the amount of updating from elected and non-elected nodes. The λ parameter ($0 < \lambda < 1$) means how long far can the node looking into network. If λ tends to 0, then the node can view just the next possibly states (as the basic TD(0) algorithm), but, if λ tends to 1, the node's vision is extended toward non-immediate future states.

For updating the value functions, the following equation is applied:

$$V_d(x) = V_d(x) + \alpha \delta_d e_d(x) \quad (3)$$

where, $\alpha$, is a learning rate, $\delta_d$ the temporal difference obtained from (2), and $e_d(x)$ is the *eligibility trace* which, is also updated decreasing and tends to zero if the related state is not elected and therefore, not visited. The *eligibility traces* updating is performed as follow:

$$e_d(x) = \begin{cases} 0 & \text{Non - greedy selection} \\ e_d(x) + 1 & \text{Elected node} \\ \lambda \gamma e_d(x) & \text{Non elected node} \end{cases} \quad (4)$$

In this case, the λ parameter weights the updating contribution from the non-selected nodes.

The algorithm to be applied reads as follows [Tupac00]:

```
Initialize V(x) randomly, e(x)=0 for
every node on network
Initialize x (select an initial node)
Repeat
  Select action a_t, according to the
  policy π (greedy policy)
  Once action a_t is carried out
```

```
    Observe the reinforcement r_t
    and the next node y
    δ_d = r_t + γ * V_d(y) - V_d(x_t)
       e_d(y)  = e_d(y) + 1
  For all the neighboring nodes:
    V_d(x) = V_d(x) + α * δ_d *
    e_d(x)
    e_d(x)  = γ * λ * e_d(x)
  y is the current node (new state s_t+1)
Until d = current node, where d is the
destination node (end state)
```

## 4. Performance Evaluation

For the experiments performed in this work, we have used a program that simulates the network structure and the traffic of packets. This program was developed with the purpose of evaluating the Q-Routing [Boyan94] algorithm. The program considers the following aspects, with the corresponding parameters:

**Topology:** Configuration of nodes, links, processing time for nodes, and link delays due to distance. The layout is loaded from a file.

**Packets:** Generation times, initial node, destination node, and current node.

**Traffic:** Rate of creation of new packets and *Poisson distribution* for packet generation.

**Report:** Specification of the interval for presentation of results such as: average routing time, average length of routes, number of packets successfully routed, and activity at each link.

In order to make the program more realistic, a few characteristics were added, namely: link delay due to bandwidth, possibility of different processing times, and delays due to distance and bandwidth on a single network. In packet generation, random destinations are used and the packets are inserted in random nodes. The rate at which new packets are created, which is expressed in packets per unit of time (u.t.), is called *network load level*. The arrival of several packets at a single node is carried out in FIFO structures.

For each step of the simulation, each node removes the first packet in its queue and, immediately after it has analyzed the destination of the packet, makes the routing decision to send it via one of its neighboring nodes. When a node receives a packet, it removes it from the network if it is the destination node; otherwise it inserts the packet in its own queue.

Routing time is defined as the time it takes the packet, from its introduction into the origin node, to its elimination at the destination node, and is measured in terms of simulation steps. The average routing time is calculated at regular intervals and corresponds to the arithmetic mean of all the packets that reach their destination during that time interval.

The average time is the main performance metric that was proposed during the learning phase; after learning occurs, this metric evaluates the quality of the current policy. The tests were performed in the network topology shown in Figure 3: 36 nodes, bandwidth 256 Kb/u.t., packet size 64 Kb each, delay due to the distance of 1 u.t. for each link, and 0.1 u.t. processing time at each node (an u.t. is an arbitrary time unit).

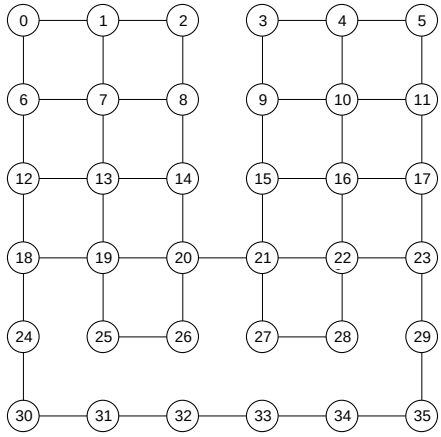


**Figure 3. The 6x6 irregular grid topology of 36 nodes used for learning tests**

Three different tests were performed: learning with low load level on the network with 1 packet per u.t., learning with high load level on the network with 12 packets per u.t. and learning with varying traffic, between 1 packet and 12 packets per u.t.

The performance of the TD-Routing was compared with the main existing algorithms, i. e., the *Shortest Path Routing* [Floyd62], Bellman-Ford algorithm [Bellman58] and the Q-Routing [Boyan94] algorithms. Q-Routing algorithm is the first routing algorithm based in Reinforcement Learning techniques, more specifically, Q--Routing is based in the Q-Learning algorithm [Watkins92]. Among the parameters that characterize the TD(λ) method, tests with the variation of the value λ, the random action rate $T_a$, and the discount rate γ, were conducted. The

best results were obtained for the following values:$\lambda = 0.2$, $T_a = 0.1$ % and $\gamma = 1.0$\$.

### 4.1. Learning with low load level and high load level

In the first test, one makes the algorithm find a good policy when the traffic of packets is relatively low, starting all the adaptive algorithms without any prior knowledge of the network's status. In this traffic pattern, the *Shortest Path Routing* algorithm is the optimal routing policy. Figure 4 presents the learning process and the convergence, herein expressed in terms of the average routing time for the algorithms in question.

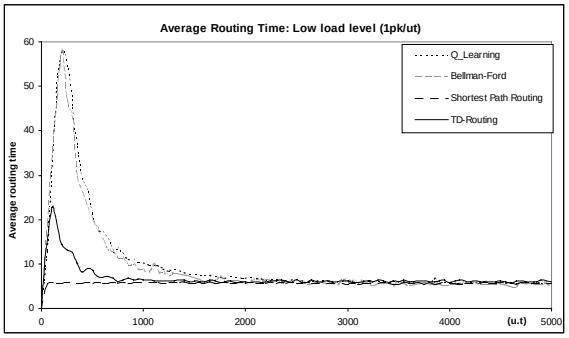


**Figure 4. Learning process for low load level (about 1 packet per u.t.)**

The second test makes use of a traffic pattern that is high enough to prevent the *Shortest Path Routing* algorithm from maintaining the minimum routing time. The adaptive algorithms succeed in finding alternative routes and converging to steady states.

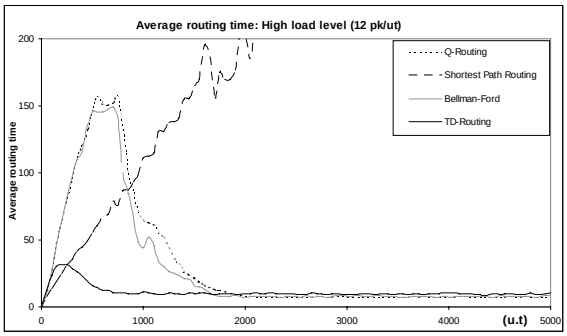


**Figure 5. Learning process for high load level (about 12 packet per u.t)**

Figure 4 shows that under low load level conditions, all algorithms succeed in converging to stable conditions with routing time values that are close to those of the *Shortest Path Routing* algorithm, which is the optimal policy in this case. Among the adaptive algorithms, the TD-Routing is the one that learns the fastest. In Figure 5, when there is a high load level on the network, it rapidly becomes jammed when the *Shortest Path Routing* algorithm is used; however, with the Bellman--Ford, the Q--Routing and the TD-Routing algorithms, the network is maintained in a steady state. The learning speed of the TD-Routing algorithm is considerably faster, allowing reaching steady states in almost half the time taken by the Bellman-Ford and Q-Routing algorithm.

### 4.2. Load Level Variation

A different test was carried out with the purpose of checking the adaptability of the algorithms to load variations, consisting of suddenly and drastically changing the load level pattern from low to high, and then returning to the low load level pattern after a period of time had elapsed. The results in terms of average routing time versus load level variations can be seen in Figure 6. The only one that did not re-adapt was the *Shortest Path Routing* algorithm.

A well-known problem in the Q-Routing algorithm was verified in this test: the algorithm does not succeed in converging to an optimal policy of minimum routes when it returns from the high to the low load level pattern, even though it has achieved a minimum route policy prior to the congestion. In Figure 7, it can be seen that as they move on to high load levels, the algorithms find new routes that are longer; in the case of the Q-Routing algorithm, when the status goes back to the low traffic level, the routes do not obtain the lengths that correspond to minimum routes.

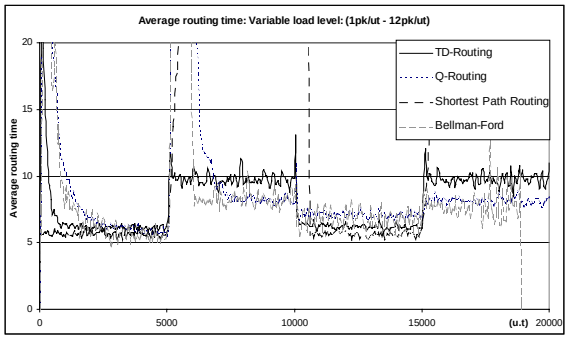


**Figure 6. Average routing time for variable load level**

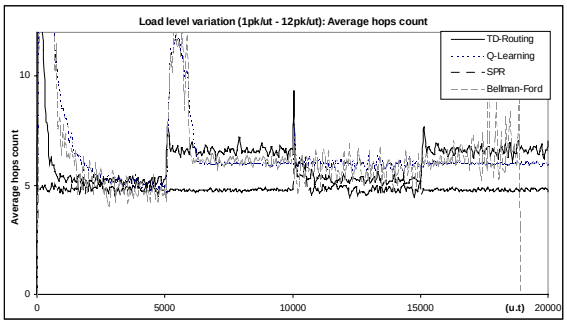


**Figure 7. Average routing lenghts for variable load level**

### 4.3. Maximum load level sustenance

Among all the others, the *Shortest Path Routing* algorithm is optimal for low load level patterns, whereas the adaptive algorithms outperform it in the case of high load levels. In order to test the potentiality of each algorithm, the network load level was gradually increased until all the algorithms reached the congestion status. In this case, the greatest load level sustenance was obtained by the Q-Routing algorithm, as can be seen in Figure 8.

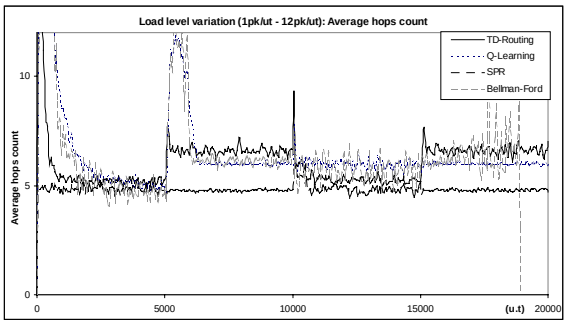


**Figure 8. Load level evaluation: the amount of packets per time unit is gradually incremented until all algorithms have reached congestion status on network**

### 4.4 Changes in the Topology

In this test, one of the links was changed so as to test adaptability to changes in network topology, this test addresses to evaluate the response to problems on link and node failures.
The adaptive algorithms succeed in performing the relearning task and redistributing the packet traffic according to the new topology, the topology changes can be seen on Figure 9.

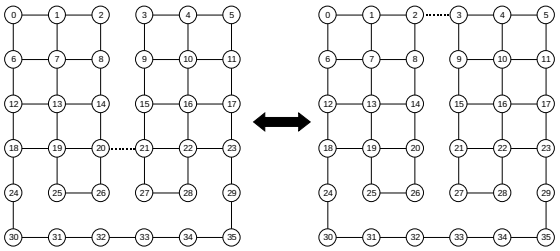

**Figure 9. Changes in the network topology**

The tests consisted of comparing the performances of the Q-routing and the TD-Routing algorithms. The learning process begins with the topology on the left-hand side of Figure 9, when the time becomes approximately 1500 u.t., the network topology changes to its counterpart on the right-hand side of Figure 9, and when time is 3000 u.t., the topology is once again in the initial state. In the first test, the load level pattern is always low and is identical to the one used in the low load level test; in the second test, the load level pattern is high and is similar to the one used in high load level tests. Figure 10 and Figure 11 present the routing times obtained with the Q-Routing and TD-Routing algorithms on low load level and high load level, respectively.

Figure 10 and Figure 11 show that the performance of the TD-Routing algorithm is better in both cases, in terms of changes in the network topology. This is due to the speed with which the TD-Routing algorithm is able to adapt to the changes in the traffic pattern. When a node is unable to route through a given link because either the link or the node is down, the value of the value function becomes very high, and the result is that specific node or link will no be longer used in the routing decisions.

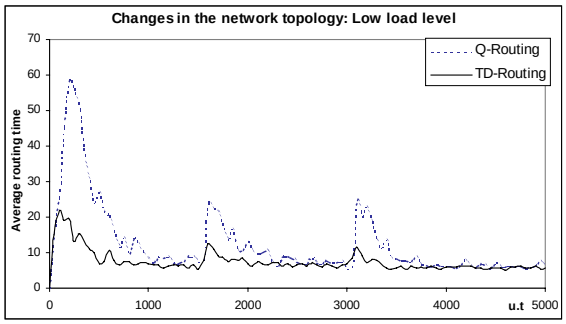


**Figure 10. Performance of the Q--Routing and TD—Routing algorithms for topology changes with a low load level**

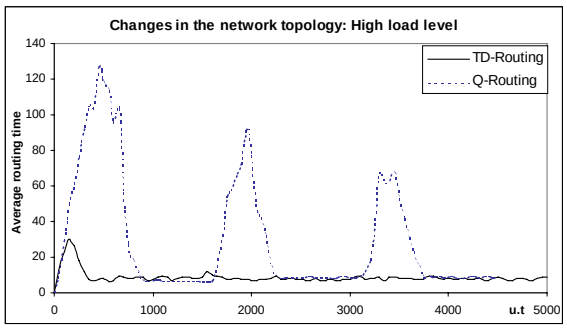


**Figura 11. Performance of the Q-Routing and TD-Routing algorithms for topology changes with a high load level**

## 5. Conclusions

The tests that were carried out clearly demonstrated how quickly the TD-Routing algorithm succeeds in adapting to changes in the traffic pattern. This is due to the algorithm uses recent information to make the routing decision. The values $V(s)$ keep estimates of the routing time from the state $s$ up to each destination. When a routing decision is made, the current node knows its neighbors' current estimated values and sends the packet through the neighbor that presents the lowest value. An overload may occur as a result of the need to obtain updated information from the, and also as a result of the information updating process itself. TD-Routing algorithm is capable of adapting to changes in the network status and its main goal is to minimize the routing time. In terms of learning speed and adaptation to load level changes, TD-Routing provides satisfactory results outperforming the *Shortest Path Routing*, Bellman-Ford and Q-Routing algorithms.

## References

[Bellman57] Bellman, R. E.: Dynamic Programming. Princeton University Press (1957), Princeton, NJ.

[Bellman58] Bellman, R. E.: On a Routing Problem. Quarterly of Applied Mathematics 16 (1958), 315--333.

[Boyan94] Boyan, J. A., Littman, M. L.: Packet Routing in Dynamical Changes Networks: A Reinforcement Learning Approach. In Advances in Neural Information Processing Systems 6 (1994), 671--678. Morgan, Kauffmann, San Francisco, California.

[Dijkstra59] Dijkstra, E. W.: A note on two problems in connection with graphs. Numer. Math. 1 (1959), 269--271.

[Floyd62] Floyd, R. W.: Algorithm 97 (Shortest Path). In Communications of the ACM vol 5(6) (1962).

[Haykin98] Haykin, S.: Neural Networks --- A Comprehensive Foundation, Mcmillan College Publishing Co. (1998).

[Littman93] Littman, M., Boyan J. A.: A distributed reinforcement learning scheme for Network Routing. In Proceedings of the First International Workshop on Applications of Neural Networks in Telecommunications (1993), 45-51. Lawrence Erlbaum, Hillsdale, New Jersey.

[Singh96] Singh, S. P., Sutton R. S.: Reinforcement Learning with Replacing Eligibility Traces. In Machine Learning 22 (1996), 123-158.

[Sutton88] Sutton, R. S.: Learning to predict by the method of temporal differences. In Machine Learning 3 (1988), 9-44.

{Sutton98] Sutton, R., Barto, A.: Reinforcement Learning: An Introduction, MIT Press, Cambridge MA (1998).

[Tanenbaum96] Tanenbaum, A. S.: Computer Networks. Englewood Cliffs, N.J., Prentice Hall. Third Edition (1996).

[Tupac00] Túpac, Y. J.: Roteamento Adaptativo em Redes de Comunicação de Dados por Reinforcement Learning. M.Sc.~Theses, Pontificia Universidade Católica de Rio de Janeiro. Brasil (2000).

[Watkins89] Watkins, C. J. C. H.: Learning from Delayed Rewards, Ph.D. thesis, University of Cambridge, England (1989).

[Watkins92] Watkins, C. J. C. H. and Dayan, P.: Q-Learning. Machine Learning 8 (1992), 279-292.