Economides, A.A.: **Learning automata routing in connection-oriented networks.**
International Journal of Communication Systems, Vol. 8, No. 4, pp. 225-237, 1995.

LEARNING AUTOMATA ROUTING IN CONNECTION-ORIENTED NETWORKS

Anastasios A. Economides

University of Macedonia
Thessaloniki 54006, GREECE

Tel # +3031-891799
Fax # +3031-891292
economid@macedonia.uom.gr

20 March 1995

Abstract

Learning automata are used at the source nodes of a connection-oriented network to dynamically route newly arriving virtual calls to their destination.

First, two new learning automata are introduced. Then, these two learning automata, as well as the well-known L learning automaton and the deterministic shortest-path algorithms are used in a simulation program to route virtual calls. The more frequent the updating and the more recent network state information used, the better the performance.

In the sequence, the virtual link length is developed as a function of both the number of packets and the number of virtual calls at the network link. This virtual link length is used in the learning automata routing algorithm and is showed via simulation to be superior to the minimum packet delay or shortest-queue-type link length, usually used in real networks. Thus, in connection-oriented networks, not only the packet but also the virtual call traffic characteristics should be used in the routing decisions.

Furthermore, when the network state information is out-of-date, or when there are few virtual calls and each one carries a large number of packets, then the virtual link length should be based more on the number of virtual calls than on the number of packets at this link. On the other hand, when the network state information is current and there are many virtual calls and each one carries a small number of packets, then the virtual link length should be based more on the number of packets than on the number of virtual calls at this link.

Key-words: connection-oriented networks, learning automata routing, simulation, virtual calls, virtual circuit networks.

1 INTRODUCTION

Dynamic routing in connection-oriented or virtual circuit networks is the selection for every newly arriving virtual call of the currently best path from its source to its destination node. Dynamic routing may be implemented according to two ways:

1) In *deterministic routing*, the selection of a route from source to destination is done deterministically. If our network model is correct and the network state does not change drastically from the moment we measure it until we make the routing decisions, then we may act with confidence (in a deterministic way) that our routing decisions are correct. One possible rule to achieve the optimal routing is with a weighted round-robin fashion. Another possible rule is to select a route if the cost of this route is less than the cost of all other possible routes.

2) In *probabilistic routing*, the selection of a route from source to destination, is done probabilistically. Note, that deterministic routing is a special case of probabilistic routing (with probability 1). Solving the routing problem in computer networks, we only find (if we can) the "solution" to an approximation of the real problem. The underlying assumptions of the model (e.g. independent exponential distributions), or even other management problems that are not explicitly considered, affect the "solution". Thus, instead of using a definitive decision (by completely trusting the optimality conditions and the measurements) a probabilistic one may be used favoring some action.

In this paper, we propose such a probabilistic routing approach based on learning automata algorithms. These are adaptive control algorithms for highly uncertain systems [20]. They select probabilistically an action and then update their action probabilities according to the outcome of the selected action. If the outcome is favorable, then the probability of the selected action increases, otherwise it decreases. So, instead of deterministically choosing an action, learning automata choose it with very high probability. Note, that if we appropriatelly calibrate the step size of these learning automata algorithms, then they may choose an action deterministically (with probability 1).

The greatest potential of the learning automata methodology is that it permits the control of very complex dynamic systems. Even when little information is available, they act to minimize the effects of future system changes. Learning automata have been applied to routing problems like telephone routing [1, 21, 22, 24, 28, 29], datagram routing [5, 11, 14, 15, 18, 26, 27], and virtual circuit routing [10, 8]. British Telecom has already developed a learning automata routing system for use in their long distance network [4].

Glorioso, Grueneich & Dunn [14] use learning automata for routing. If a call is successful through a path, the probability of using the same path is increased, otherwise it is decreased. Simulation results show the ability of learning automata to route the traffic around the destroyed portions of the network and provide an improved grade of service. Glorioso, Grueneich & McElroy [15] equalize the loading on the network links using learning automata routing. They demonstrate the ability of learning automata to adapt to a loss of network facilities and to operate with uncertain network facilities.

Narendra, Wright & Mason [24] use the M automaton for telephone call routing. An action corresponds to a sequence of alternate paths to be attempted. The penalty probability of each action is updated according to the success or blocking of the telephone call. At overload conditions, learning automata routing results in a lower blocking probability and lower node congestion compared to fixed rule alternate routing.

Narendra & Thathachar [22] introduce two new models of nonstationary random environments. When an action is performed, its penalty probability increases while the penalty probabilities of the other actions decrease. They prove that the L_{R-P} automaton operating in such environment tends to equalize the expected penalty probabilities. Simulation of telephone call routing confirms the equalization of the blocking probabilities.

Chrystall & Mars [5] route messages over the outgoing links using learning automata at every network node. The delay experienced by every message is used to update the probability of selecting the same link again. They point out that the L_{R-I} automaton attempts to equalize the average path delays, while the L_{R-P} automaton attempts to equalize the accumulated path delays.

Srikantakumar & Narendra [28] analyze nonstationary environments where the penalty probabilities are functions of the action probabilities. They prove that the L_{R-P} automaton equalizes the penalty rates, while the $L_{R-\epsilon P}$ automaton equalizes the penalty probabilities. For telephone call routing, the L_{R-P} automaton attempts to equalize the rates of the call blocking probabilities, while the $L_{R-\epsilon P}$ automaton attempts to equalize the call blocking probabilities. Further simulation results by Narendra & Mars [21] verify the behavior predicted by the theoretical analysis.

Mason [18] proposes new learning automata algorithms to route packets in packet switched networks. The routing parameters are updated according to the packet delays on the links. Simulation results show the superiority of the learning automata routing over fixed and random routing.

Akselrod and Langholz [1] show via simulation that learning automata routing performs better than fixed routing for telephone networks. Furthermore, they introduce a penalty function of the number of calls on each trunk group to represent its load.

Nedzelnitsky and Narendra [26] propose a new model for nonstationary enironments whose penalty probabilities depend on the previous penalty probabilities and on the probabilities of the performed actions. Then they simulate routing in datagram packet switched network by learning automata at every network node. The SL_{R-I} and $SL_{R-\epsilon P}$ automata equalize the delays, while the SL_{R-P} automaton equalizes the delay rates.

Zgierski & Oommen [29] show via object-oriented simulation the superiority of learning automata telephone call routing to fixed rule and random routing in terms of call blocking probability. Their simulation environment uses any of the M-automaton, several linear learning automata, the corresponding discrete learning automata and their absorbing versions.

Economides & Silvester [11] propose learning automata for routing packets in a network with unreliable links. The routing probabilities are updated according to the success or failure of packet transmission, or according to the marginal packet delays over the unreliable links. Learning automata are also used to estimate the link error rates.

Economides, Ioannou & Silvester [10] propose learning automata for routing virtual calls. The routing probabilities are updated according to the unfinished work on the paths. Simulation results show equalization of the unfinished work on the paths. Furthermore, two other extensions on the learning automata updating algorithms are proposed: the multiple response learning automata (which are analyzed mathematically in [9]), and the virtual updating learning automata.

In this paper, we extend our results on introducing learning automata [10] at the source nodes of the network to dynamically route newly arriving virtual calls to their destination. In section 2, we introduce two new learning automata and propose the learning automata routing of virtual calls in connection-oriented networks. In section 3, we simulate virtual circuit networks and use the two new learning automata, as well as the L learning automaton and the deterministic algorithm for routing newly arriving virtual calls. We find that the more frequent the updating and the more recent information used, the better the performance.

In section 4, we develop a new measure of the link load, the virtual link length, which is a function of both the number of packets and the number of virtual calls at this link. We use it to update the learning automaton routing algorithm, that probabilistically routes every newly arriving virtual call. We show via simulation that this virtual link length is superior to the minimum packet delay or shortest-queue-type link length, usually used in real networks [2, 3, 12, 13, 16, 17, 19, 25, 30]. Furthermore, when the network state information is out-of-date, or when there are few virtual calls and each one carries a large number of packets, then the number of virtual calls should weight more in the virtual link length than the number of packets. On the other hand, when the network state information is current, and there are many virtual calls and each one carries a small number of packets, then the number of packets should weight more in the virtual link length than the number of virtual calls. Finally, in section 5, we summarize the results.

2 LEARNING AUTOMATA AS ROUTERS

In this section, we apply three learning automata algorithms to the routing problem in virtual circuits networks [8]. In these networks, a call set-up packet, which may be part of the first packet of a message, initiates the establishment of a virtual circuit from source to destination. All other packets belonging to this message follow the same route which remains fixed for the duration of the call [6, 7].

The formulation of the routing problem can be done either on the link flow space or on the path flow space. In future high speed computer communication networks, the transmission delay will be extremely low and we will not want to spend extra time in network management decisions inside the network. Therefore, the computationally intensive processes, such as the network management decisions, will be transferred outside of the network either to the source or to the destination node. With this in mind, we formulate the routing problem on the path flow space, which means that the control decisions will be done at the source nodes. In this way, we also avoid loops, since the virtual calls will follow a previously determined loop free path.

Since network conditions change very rapidly, the minimum length path at a time instant may not be the same at the next time instant. Also, the information about the network state is always obsolete and inacurrate. Therefore the routing decisions should not overreact and immediately send a new virtual call along the minimum length path to its destination. The system management decisions should fast track the current network state but without introducing instability.

The proposed dynamic virtual call routing algorithms are based on a *Probabilistic Selection of the Minimum Length Path* idea [10]. Instead of using a definitive decision as to where to send a newly arriving virtual call, we vary the routing probabilities favoring the minimum length path.

Every source node $[s.]$ has a learning automaton, for every destination node $[.d]$, that routes newly arriving virtual calls at node $[s.]$ and destined for node $[.d]$. These learning automata operate asynchronously and base their decisions on the current network state. The actions, $a(n)$, of each automaton are to select some particular path $\pi[sd]$ to the destination node $[.d]$.

The automaton selects action $a(n) = a_{\pi[sd]}$ with probability $P_{\pi[sd]}(n)$. Action $a(n)$ becomes input to the environment. If this results in a favorable outcome for the network performance, then the probability $P_{\pi[sd]}(n)$ is increased (rewarded) by $\Delta P_{\pi[sd]}(n)$ and the $P_{p[sd]}(n), \ \ \forall p[sd] \neq \pi[sd]$, are decreased by $\Delta P_{p[sd]}(n)$. Otherwise, if an unfavorable outcome happens, then the $P_{\pi[sd]}(n)$ is decreased (penalized) by $\Delta P_{\pi[sd]}(n)$ and the $P_{p[sd]}(n), \ \ \forall p[sd] \neq \pi[sd]$ are increased by $\Delta P_{p[sd]}(n)$.

The simplest information that someone can measure and transfer about the network state is the *packet delay* through each path from source to destination. Measurements of the packet delay are also used in the ARPANET routing [13, 16, 17, 30] as well as in the Internet routing [19, 25].

In the deterministic shortest-path algorithm, we send a newly arriving virtual call along the minimum packet delay path. However, in the three learning automata algorithms proposed in the next paragraphs, instead of using a definitive decision as to where to send a newly arriving virtual call, we vary the path routing probabilities favoring the minimum delay path. Note that the deterministic shortest-path algorithm is a special case of the learning automata algorithms, since by suitably tuning the parameters, we can select the minimum length path with probability 1.

The values for the reward and penalty parameters in the learning automata should be chosen by experimentation for specific network topology, number of paths between source-destination pairs, traffic characteristics, information about the network state, updating time interval and other variables. For uniformity across all three learning automata algorithms, we wanted to use the same reward and penalty parameters. After a lengthy experimentation, we found that the values of reward parameter = 0.2 and penalty parameter = 0.8 achieve good performance for all three learning automata algorithms. However, as we show in section 3, there exist other values for these parameters in each one of the three learning automata algorithms that result in better performance.

2.1 L

The first algorithm uses the well known L learning automaton [23, 20] with reward parameter $\alpha = 0.2$ and penalty parameter $\beta = 0.8$. When an action is attempted at time n, the L automaton increases at time $n+1$ the action's probability by an amount proportional to one minus its value at n for a favorable response and decreases it by an amount proportional to its value at n for an unfavorable response. In our case, if the selected path has the minimum packet delay at the next iteration, then we increase the probability of selecting it again, otherwise we decrease it. More specifically:

Let path $\pi[sd]$ is selected for the v^{th} virtual call.
Update the probabilities at time instances n until the $(v+1)^{th}$ virtual call arrives:

$$
\begin{array}{ll}
If\ T_{\pi[sd]}(n) \leq \min\limits_{p[sd]}\{T_{p[sd]}(n)\},\ then & \\
\quad P_{\pi[sd]}(n+1) = P_{\pi[sd]}(n) + 0.2 * [1 - P_{\pi[sd]}(n)] & \\
\quad P_{p[sd]}(n+1) = P_{p[sd]}(n) - \ 0.2 * P_{p[sd]}(n) & \forall\ p[sd] \neq \pi[sd] \\
else & \\
\quad P_{\pi[sd]}(n+1) = P_{\pi[sd]}(n) - 0.8 * P_{\pi[sd]}(n) & \\
\quad P_{p[sd]}(n+1) = P_{p[sd]}(n) + 0.8 * \left[1 - P_{p[sd]}(n)\right] & \forall\ p[sd] \neq \pi[sd]
\end{array}
$$

Select the path for the $(v+1)^{th}$ virtual call probabilistically according to $P_{p[sd]}(v+1)\ \ \forall p[sd]$.

Thus, let path $\pi[sd]$ is selected for the v^{th} virtual call. Then, we measure the average packet delay over the selected path $\pi[sd]$, $T_{\pi[sd]}$, as well as over the other paths $p[sd]$, $T_{p[sd]}$, during an updating time interval $(n, n+1]$. If the delay over the selected path is less that the delay over the other paths, then we increase the probability of selecting the same path again, $P_{\pi[sd]}$, and decrease the probability of selecting any other path, $P_{p[sd]}$. Otherwise, if the delay over the selected path is not the minimum, then we decrease the probability of selecting the same path again, $P_{\pi[sd]}$, and increase the probability of selecting any other path, $P_{p[sd]}$. This updating process is repeated until the next $v + 1$ virtual call arrives. At that moment, this new virtual call is routed probabilistically, according to the current routing probabilities, along a path.

2.2 *MRL*

The second algorithm uses the S-model Multiple Response Linear (MRL) learning automaton. The norms of behavior for the Q-model MRL learning automaton are investigated in [8, 9]. The idea for the MRL learning automata is to use different adaptation rates for different environment responses. We consider two response and penalty regions for the algorithm and the functions that define these regions are linear functions with parameter 2. When the selected path gives very good performance (very small delay), then we increase the probability of the selected path very fast. When the selected path gives almost good performance (small delay), then we increase the probability of the selected path slowly. Correspondingly, when the selected path gives very bad performance (very large delay), then we decrease the probability of the selected path very fast. When the selected path gives almost bad performance (large delay), then we decrease the probability of the selected path slowly. Here, we take as reward parameters $\alpha^1 = 0.8$ (excellent choice), $\alpha^2 = 0.2$ (good choice, but not excellent) and penalty parameters $\beta^2 = 0.8$ (bad choice), $\beta^1 = 1$ (very bad choice). More specifically:

Let path $\pi[sd]$ is selected for the v^{th} virtual call.

Update the probabilities at time instances n until the $(v+1)^{th}$ virtual call arrives:

$$
\begin{aligned}
&If\ T_{\pi[sd]}(n) \leq \min_{p[sd]}\{T_{p[sd]}(n)/2\},\ then\\
&\quad P_{\pi[sd]}(n+1) = P_{\pi[sd]}(n) + 0.8 * [1 - P_{\pi[sd]}(n)]\\
&\quad P_{p[sd]}(n+1) = P_{p[sd]}(n) - 0.8 * P_{p[sd]}(n) \qquad\qquad \forall\ p[sd] \neq \pi[sd]
\end{aligned}
$$

$$
\begin{aligned}
&If\ \min_{p[sd]}\{T_{p[sd]}(n)/2\} < T_{\pi[sd]}(n) \leq \min_{p[sd]}\{T_{p[sd]}(n)\},\\
&\quad P_{\pi[sd]}(n+1) = P_{\pi[sd]}(n) + 0.2 * [1 - P_{\pi[sd]}(n)]\\
&\quad P_{p[sd]}(n+1) = P_{p[sd]}(n) - 0.2 * P_{p[sd]}(n) \qquad\qquad \forall\ p[sd] \neq \pi[sd]
\end{aligned}
$$

$$
\begin{aligned}
&If\ \min_{p[sd]}\{T_{p[sd]}(n)\} \leq T_{\pi[sd]}(n) \leq \min_{p[sd]}\{2 * T_{p[sd]}(n)\},\\
&\quad P_{\pi[sd]}(n+1) = P_{\pi[sd]}(n) - 0.8 * P_{\pi[sd]}(n)\\
&\quad P_{p[sd]}(n+1) = P_{p[sd]}(n) + 0.8 * \left[1 - P_{p[sd]}(n)\right] \qquad \forall\ p[sd] \neq \pi[sd]
\end{aligned}
$$

$$
\begin{aligned}
&If\ \min_{p[sd]}\{2 * T_{p[sd]}(n)\} \leq T_{\pi[sd]}(n),\\
&\quad P_{\pi[sd]}(n+1) = P_{\pi[sd]}(n) - 1 * P_{\pi[sd]}(n)\\
&\quad P_{p[sd]}(n+1) = P_{p[sd]}(n) + 1 * \left[1 - P_{p[sd]}(n)\right] \qquad \forall\ p[sd] \neq \pi[sd]
\end{aligned}
$$

Select the path for the $(v+1)^{th}$ virtual call probabilistically according to $P_{p[sd]}(v+1)\ \ \forall p[sd]$.

The algorithm works as follows: let path $\pi[sd]$ is selected for the v^{th} virtual call. Then, we measure the average packet delay over the selected path $\pi[sd]$, $T_{\pi[sd]}$, as well as over the other paths $p[sd]$, $T_{p[sd]}$, during an updating time interval $(n, n+1]$.

If the delay over the selected path is smaller than half of the minimum delay over the other paths, then we rapidly increase the probability of selecting the same path again, $P_{\pi[sd]}$, and rapidly decrease the probability of selecting any other path, $P_{p[sd]}$. If the delay over the selected path is larger than half the minimum but smaller than the minimum delay over the other paths, then we slowly increase the probability of selecting the same path again, $P_{\pi[sd]}$, and slowly decrease the probability of selecting any other path, $P_{p[sd]}$.

Otherwise, if the delay over the selected path is larger than the minimum but smaller than double the minimum delay over the other paths, then we slowly decrease the probability of selecting the same path again, $P_{\pi[sd]}$, and slowly increase the probability of selecting any other path, $P_{p[sd]}$. If the delay over the selected path is larger than double the minimum delay over the other paths, then we rapidly decrease the probability of selecting the same path again, $P_{\pi[sd]}$, and rapidly increase the probability of selecting any other path, $P_{p[sd]}$.

This updating process is repeated until the next $v+1$ virtual call arrives. At that moment, this new virtual call is routed probabilistically, according to the current routing probabilities, along a path.

2.3 *SDL*

Finally, the third algorithm uses the State Dependent Linear (SDL) learning automaton [8]. The idea for the SDL learning automaton is to make the reward and penalty parameters functions of the difference of the average delay of the selected path and the maximum average delay of the other paths between this source-destination. We use the exponential function in order to emphasize the difference in the delays. Then, the smaller the delay of a path, the more probable its selection. More specifically:

Let path $\pi[sd]$ is selected for the v^{th} virtual call.

Update the probabilities at time instances n until the $(v+1)^{th}$ virtual call arrives:

$$If\ T_{\pi[sd]}(n) \le \min_{p[sd]}\{T_{p[sd]}(n)\},\ then$$

$$P_{\pi[sd]}(n+1) = P_{\pi[sd]}(n) + 0.2 * (1 - e^{[T_{\pi[sd]}(n) - \max_{p[sd]} T_{p[sd]}(n)]}) * [1 - P_{\pi[sd]}(n)]$$

$$P_{p[sd]}(n+1) = P_{p[sd]}(n) - 0.2 * (1 - e^{[T_{\pi[sd]}(n) - \max_{p[sd]} T_{p[sd]}(n)]}) * P_{p[sd]}(n)$$

$$\forall\ p[sd] \ne \pi[sd]$$

$$else$$

$$P_{\pi[sd]}(n+1) = P_{\pi[sd]}(n) - 0.8 * e^{[T_{\pi[sd]}(n) - \max_{p[sd]} T_{p[sd]}(n)]} * P_{\pi[sd]}(n)$$

$$P_{p[sd]}(n+1) = P_{p[sd]}(n) + 0.8 * e^{[T_{\pi[sd]}(n) - \max_{p[sd]} T_{p[sd]}(n)]} * \left[1 - P_{p[sd]}(n)\right]$$

$$\forall\ p[sd] \ne \pi[sd]$$

Select the path for the $(v+1)^{th}$ virtual call probabilistically according to $P_{p[sd]}(v+1)\ \ \forall p[sd]$.

So, let path $\pi[sd]$ is selected for the v^{th} virtual call. Then, we measure the average packet delay over the selected path $\pi[sd]$, $T_{\pi[sd]}$, as well as over the other paths $p[sd]$, $T_{p[sd]}$, during an updating time interval $(n, n+1]$.

If the delay over the selected path is smaller than the delay over the other paths, then we increase the probability of selecting the same path again, $P_{\pi[sd]}$, and decrease the probability of selecting any other path, $P_{p[sd]}$. If the difference of the delay of the selected path minus the maximum delay over the other paths, $T_{\pi[sd]} - \max\limits_{p[sd]} T_{p[sd]}$, is large, this means that the selected path has very good performance. In this case, the function $exp[T_{\pi[sd]} - \max\limits_{p[sd]} T_{p[sd]}]$ approaches the 0. Therefore, the probability for the selected path, $P_{\pi[sd]}$, increases very fast. If this difference is small, this means that the selected path has marginally good performance. In this case, the function $exp[T_{\pi[sd]} - \max\limits_{p[sd]} T_{p[sd]}]$ approaches the 1. Therefore, the probability for the selected path, $P_{\pi[sd]}$, increases slowly. The probabilities for the other paths decrease accordingly.

Otherwise, if the delay over the selected path is not the minimum, then we decrease the probability of selecting the same path again, $P_{\pi[sd]}$, and increase the probability of selecting any other path, $P_{p[sd]}$. If the difference of the delay over the selected path minus the maximum delay over the other paths, $T_{\pi[sd]} - \max\limits_{p[sd]} T_{p[sd]}$, is large, this means that the selected path has not so bad performance. In this case, the function $exp[T_{\pi[sd]} - \max\limits_{p[sd]} T_{p[sd]}]$ approaches the 0. Therefore, the probability for the selected path, $P_{\pi[sd]}$, decreases slowly. If this difference is small, this means that the selected path has very poor performance. In this case, the function $exp[T_{\pi[sd]} - \max\limits_{p[sd]} T_{p[sd]}]$ approaches the 1. Therefore, the probability for the selected path, $P_{\pi[sd]}$, decreases fast. The probabilities for the other paths increase accordingly.

This updating process is repeated until the next $v+1$ virtual call arrives. At that moment, this new virtual call is routed probabilistically, according to the current routing probabilities, along a path.

3 SIMULATION

In this section, we compare the performance of the deterministic shortest-path and the three learning automata algorithms (see previous section) via simulation. In an arbitrary topology network, we consider two specific paths to be available between a given source-destination pair. We assign a learning automaton at the source node to route newlly arriving virtual calls through either the first or the second path. Path # 1 has seven links each one with effective service rate 1. Path # 2 has seven links with effective service rates 1, 0.5, 2, 2, 2, 0.5 and 1. Once a path is selected for a new virtual call, all packets belonging to this virtual call are transmitted through this path. Upon arrival, each packet is sent through the selected path for the virtual call it belongs to. Then the packet is transmitted link-by-link to the destination.

We consider Poisson distributed virtual call arrivals, Poisson distributed packet arrivals in a virtual call and geometrically distributed number of packets in a virtual call. Then the virtual call duration (lifetime) is exponentially distributed. Finally, the packet service requirement is exponentially distributed with mean $1/\mu = 1$. For the traffic characteristics, we consider two cases:

i) 30/2/40: the mean interarrival time of virtual calls is $1/\gamma = 30$, the interarrival time of packets in a virtual call is $1/r = 2$ and the mean virtual call duration is $1/\delta = 40$.

ii) 50/5/200: the mean interarrival time of virtual calls is $1/\gamma = 50$, the interarrival time of packets in a virtual call is $1/r = 5$ and the mean virtual call duration is $1/\delta = 200$.

For measuring the path delay and updating the probabilities, we consider two cases:

i) $\underline{1}$: at every packet departure from the network through a path, the destination sends to the source the delay of this last packet through this path.

ii) $\underline{50}$: at every 50th packet departure from the network through a path, the destination sends to the source the average packet delay of these 50 last packets through this path.

The source node keeps and updates the information about the delay of its paths to the destination. The information about the delay of a path is updated every time a packet arrives at the destination through this path. However, this updating is not done immediately, but we assume that this information becomes available to the source node after a feedback delay. We assume that no extra traffic is created for transferring this feedback information to the source node (it is either piggybacked on regular packets or uses a different channel). We consider two cases for the feedback delay:

30/2/40	1 instant	1 obsolete	50 instant	50 obsolete
deterministic	50.59 ±0.89	63.59 ±1.28	55.38 ±0.93	61.97 ±1.36
L automaton	50.27 ±1.15	61.29 ±1.36	57.37 ±0.88	61.44 ±1.25
MRL automaton	50.64 ±0.73	61.27 ±1.63	61.15 ±0.92	64.04 ±1.36
SDL automaton	48.92 ±0.51	62.52 ±1.04	57.37 ±1.14	60.60 ±1.46

50/5/200	1 instant	1 obsolete	50 instant	50 obsolete
deterministic	46.79 ±1.75	57.84 ±1.92	60.52 ±2.21	68.30 ±2.23
L automaton	45.35 ±1.45	54.85 ±2.31	61.43 ±1.76	65.43 ±1.77
MRL automaton	43.25 ±1.45	56.45 ±2.13	62.05 ±3.16	65.67 ±2.52
SDL automaton	46.22 ±1.36	57.45 ±2.17	60.81 ±1.79	67.24 ±1.68

Table 1: The average packet delay ± error (95% confidence interval) for deterministic, Linear automaton, Multiple Response automaton and State Dependent automaton based routing.

i) *instantaneous* information, when the feedback delay is 7 time units. In this case, we assume that the feedback information has higher priority over other packets and does not wait in queues.

ii) *obsolete* information, when the feedback delay is 60 time units. In this case, we assume that the feedback information is piggybacked on regular packets and is transferred back to the source node.

Updating the information of a path asynchronously at packet departure instances has an undesirable characteristic. If a path becomes unattractive for routing packets through it, then we may not route any more packets through it. However, our information about its length remains the same, although after some time this path may become idle. We have overcome this problem by sending a probe packet through a path that has not been used for 100 time units. In this way, we may update our information about its delay.

In Table 1 and Figures 1-8, we show the simulation results for the average packet delay for 10,000 virtual calls.

For given learning parameters $\alpha = 0.2$ and $\beta = 0.8$, all four algorithms perform similarly, although the learning automata algorithms achieve better performance. Furthermore, the learning automata have more flexibility, since we can calibrate their learning parameters depending on the particular network topology and traffic characteristics. By suitably tuning the reward and penalty parameters, the learning automata give improved performance.

For example, in the case of 30/2/40, the L automaton with $\alpha = 0$ and $\beta = 0.6$ achieves an average packet delay of 45.03 (when 1 instant), 52.42 (when 1 obsolete), 54.40 (when 50 instant), 57.99 (when 50 obsolete). These delays are much smaller than those in Table 1.

In the case of 50/5/200, the MRL automaton with $[\alpha^1 = 0.8, \alpha^2 = 1, \beta^2 = 1, \beta^1 = 1]$ achieves an average packet delay of 57.14 (when 50 instant) and with $[\alpha^1 = 0.8, \alpha^2 = 0.2, \beta^2 = 1, \beta^1 = 1]$ achieves an average packet delay of 64.43 (when 50 obsolete). Again, these delays are much smaller than those in Table 1.

Another important result is that the more frequent we update the algorithms and the more recent network state information we have, the better the performance. Thus, the best performance is achieved when the router knows the delay experienced by every packet as it traverses a particular path, as soon as possible.

4 VIRTUAL LINK LENGTH

In this section, we develop the virtual link length, a new measure for the link length in virtual circuit networks. We use it in the learning automata routing algorithm to route newly arriving virtual calls. Then, we show via simulation its superiority over the minimum packet delay or shortest-queue routing.

Let a link ij with service rate C_{ij}. We propose as link length $l_{ij}(n)$ at time n a convex combination of its current length $l_{ij}^{current}(n)$ and its expected length in the future $l_{ij}^{future}(n)$. In this way, we base our decisions not only on the current network state, but also on the estimated future network state. We call it *virtual link length* and define it as [8]:

$$l_{ij} = \epsilon * l_{ij}^{current}(n) + (1 - \epsilon) * l_{ij}^{future}(n)$$

We may consider as current length $l_{ij}^{current}(n) = \dfrac{1 + N_{ij}(n)}{\mu C_{ij}}$, a linear function of the number of packets on link ij, N_{ij}. This current link length is the estimated packet delay on this link right now. We may consider as future length $l_{ij}^{future}(n) = \dfrac{1+V_{ij}(n)}{\mu C_{ij}}$, a linear function of the number of virtual calls on link ij, V_{ij}. This future link length is the estimated packet delay on this link in the near future.

Then the virtual length of link ij is

$$l_{ij} = \epsilon * \frac{1+N_{ij}(n)}{\mu C_{ij}} + (1-\epsilon) * \frac{1+V_{ij}(n)}{\mu C_{ij}} \quad 0 \leq \epsilon \leq 1$$

A related measure to the virtual link length is the *unfinished work* [10]:

$$U_{ij}(n) = \frac{1+N_{ij}(n)}{\mu C_{ij}} + \frac{r}{\delta} * \frac{1+V_{ij}(n)}{\mu C_{ij}}$$

where the future link length is weighted by r/δ: the average number of packets in a virtual call. The unifinished work represents the average delay due to both the current packets waiting to be transmitted and the packets that are expected to arrive (due to the current open virtual calls) and be transmitted.

The virtual length of a path $\pi[sd]$ is $l_{\pi[sd]}(n) = \sum_{ij \in \pi[sd]} l_{ij}(n)$.

The routing decisions are done by a L algorithm with reward parameter $\alpha = 0.2$ and penalty parameter $\beta = 0.8$. If the selected path has the minimum virtual length at the next iteration, then we increase the probability of selecting it again, otherwise we decrease it.

Let path $\pi[sd]$ is selected for the v^{th} virtual call.

Update the probabilities at time instances n until the $(v+1)^{th}$ virtual call arrives:

$$\begin{array}{ll}
If\ l_{\pi[sd]}(n) = \min_{p[sd]}\{l_{p[sd]}(n)\},\ then & \\
\quad P_{\pi[sd]}(n+1) = P_{\pi[sd]}(n) + 0.2 * [1 - P_{\pi[sd]}(n)] & \\
\quad P_{p[sd]}(n+1) = P_{p[sd]}(n) - \ 0.2 * P_{p[sd]}(n) & \forall\ p[sd] \neq \pi[sd] \\
else & \\
\quad P_{\pi[sd]}(n+1) = P_{\pi[sd]}(n) - 0.8 * P_{\pi[sd]}(n) & \\
\quad P_{p[sd]}(n+1) = P_{p[sd]}(n) + 0.8 * \left[1 - P_{p[sd]}(n)\right] & \forall\ p[sd] \neq \pi[sd]
\end{array}$$

Select the path for the $(v+1)^{th}$ *virtual call probabilistically according to* $P_{p[sd]}(v+1) \quad \forall p[sd]$.

Next, we investigate the effect of the parameter ϵ on the average packet delay.

We consider the same network as that of the previous section. The mean packet service requirement is $1/\mu = 1$. The total packet arrival rate is $r * \gamma/\delta = 4/5$ (i.e. 4 packets per 5 time units). Two cases that achieve this rate are the following:

i) 5/50/200: the mean interarrival time of virtual calls is $1/\gamma = 5$, the mean interarrival time of packets in a virtual call is $1/r = 50$ and the mean virtual call duration is $1/\delta = 200$.

ii) 50/5/200: the mean interarrival time of virtual calls is $1/\gamma = 50$, the mean interarrival time of packets in a virtual call is $1/r = 5$ and the mean virtual call duration is $1/\delta = 200$.

For measuring the path length and updating the path probabilities, we consider two cases:

i) <u>1</u> : the current number of packets at each link is sent to the source at every packet departure from that link.

ii) <u>50</u> : the average number of packets at each link during the last 50 time units is sent to the source at every 50th packet departure from that link.

The source node keeps and updates the information about the virtual lengths of its paths to the destination. The information about the virtual length of a path is updated every time a packet arrives at the destination through this path. However, this updating is not done immediately, but we assume that this information becomes available to the source node after a feedback delay. We assume that no extra traffic is created for transferring this feedback information to the source node (it is either piggybacked on regular packets or uses a different channel). We consider two cases for the feedback delay:

i) <u>*instantaneous*</u> information, when the feedback delay is 7 time units. In this case, we assume that the feedback information has higher priority over other packets and does not wait in queues.

ii) <u>*obsolete*</u> information, when the feedback delay is 60 time units. In this case, we assume that the feedback information is piggybacked on regular packets and is transferred back to the source node.

In Table 2 and Figure 9, 10, we show the simulation results for the average

5/50/200	1 instant	1 obsolete	50 instant	50 obsolete
$\epsilon = 0.2$	104.22 ±4.50	102.20 ±5.51	133.30 ±5.98	129.71 ±4.92
$\epsilon = 0.4$	59.61 ±3.31	59.97 ±3.06	78.49 ±2.75	73.94 ±2.38
$\epsilon = 0.6$	46.98 ±2.43	46.12 ±1.79	60.88 ±1.67	56.81 ±1.53
$\epsilon = 0.8$	39.77 ±1.05	42.68 ±1.25	64.12 ±2.05	77.94 ±3.33
$\epsilon = 1$	37.19 ±1.22	50.66 ±2.06	104.38 ±4.36	126.66 ±4.45
path delay	55.97 ±3.98	97.02 ±8.79	106.41 ±8.03	121.67 ±8.15

50/5/200	1 instant	1 obsolete	50 instant	50 obsolete
$\epsilon = 0$	73.01 ±3.89	69.24 ±5.15	125.73 ±13.88	100.86 ±7.56
$\epsilon = 0.2$	36.70 ±0.98	37.20 ±0.83	51.43 ±1.66	51.50 ±1.77
$\epsilon = 0.4$	34.39 ±1.05	37.23 ±1.42	64.44 ±1.69	68.90 ±1.71
$\epsilon = 0.6$	34.85 ±1.05	39.41 ±1.10	76.96 ±1.50	85.60 ±1.10
$\epsilon = 0.8$	35.29 ±0.88	41.59 ±1.11	83.39 ±1.19	92.28 ±2.13
$\epsilon = 1$	37.02 ±1.02	44.15 ±0.99	86.26 ±2.34	97.26 ±3.46
path delay	45.35 ±1.45	54.85 ±2.31	61.43 ±1.76	65.43 ±1.77

Table 2: The average packet delay ± error (95% confidence interval) for different values of the parameter ϵ, when we use as path length the sum of the virtual link lengths $l_{ij} = \epsilon * \dfrac{1 + N_{ij}}{\mu C_{ij}} + (1 - \epsilon) * \dfrac{1 + V_{ij}}{\mu C_{ij}}$, or the path delay.

packet delay for 10,000 virtual calls.

An important observation made in the previous section is also repeated here: the more frequent we update the learning automaton algorithm and the more recent network state information we have, the better the performance.

We also notice that a proper value for the parameter ϵ should be experimentally selected for best performance. Using only the number of virtual calls on each link ($\epsilon = 0$) as the link length is very inefficient (actually, for the case 5/50/200, the average network delay becomes extremely high and we do not even show it). Also, it is not always best to use only the number of packets on each link ($\epsilon = 1$) as the link length.

For comparison, we also show the average network delay, when we use the packet delay on a path as the path length. We remark that using both the number of packets and virtual calls is much better than using the packet delay on the path.

In case, we update the learning automaton infrequently and have obsolete state information, then it seems better to weight more (ex. $\epsilon < 1/2$) the number of virtual calls, V_{ij}, than the number of packets, N_{ij}, in the virtual link length. Then, the routing decisions depend more on l_{ij}^{future} than on $l_{ij}^{current}$.

When the interarrival time of virtual calls is very short $1/\gamma = 5$, virtual calls arrive very frequently into the network. On the average, there are $\gamma/\delta = 40$ virtual calls, each one carries $r/\delta = 4$ packets, so there are 160 packets into the network. Thus, it is important to weight properly the dependency of the routing algorithm onto the number of packets and virtual calls. Table 2 shows that the routing decisions should be based more on the number of packets on each link than on the number of virtual calls on each link. If at every packet departure, we know the current network state instantaneously, then it is better to base the routing decisions only (100%) on the current number of packets. If at every packet departure, we know the network state after a feedback delay, then it is better to base the routing decisions at 80% on the number of packets and at 20% on the number of virtual calls. If at every $50th$ packet departure, we know the network state either instantaneously or after a feedback delay, then it is better to base the routing decisions at 60% on the number of packets and at 40% on the number of virtual calls.

However, if we increase the interarrival time of virtual calls at $1/\gamma = 50$, the number of virtual calls plays a more important role in the routing decisions. In this case, on the average, there are $\gamma/\delta = 4$ virtual calls, each one carries $r/\delta = 40$ packets, so there are 160 packets into the network. Here, we have fewer virtual calls, but the impact of each one on the network performance is greater than in the previous case. Thus, we weight the number of virtual calls more than previously. This is shown in Table 2. If at every packet departure, we know the current network state instantaneously, then it is better to base the routing decisions at 40% on the number of packets and at 60% on the number of virtual calls. If at every packet departure, we know the network state after a feedback delay, then it is better to base the routing decisions at 20% on the number of packets and at 80% on the number of virtual calls. If at every $50th$ packet departure, we know the network state either instantaneously or after a feedback delay, then it is better to base the routing decisions at 20% on the number of packets and at 80% on the number of virtual calls.

Note also, that although the traffic characteristics 5/50/200 and 50/5/200 give the same packet arrival rate, the overall average packet delay is different. It is obvious, that using only the number of virtual calls or only the number of packets as a measure for the traffic in connection-oriented networks (as it is done in real networks [2, 3, 12, 13, 16, 17, 19, 25, 30]) is inefficient. The proposed virtual link length incorporates both the number of packets and the number of virtual calls on the link and provides much better performance.

5 CONCLUSIONS

In this paper, we use learning automata at the source nodes of a connection-oriented network to dynamically route newly arriving virtual calls to their destination. First, we introduce the MRL and the SDL learning automata. We use these two new learning automata, as well as the well-known L learning automaton and the deterministic shortest-path algorithms in a simulation program to route virtual calls. We find that the more frequent the updating and the more recent information used, the better the performance.

Then, we develop a new measure for the load on a link, called the virtual link length, which is a function of both the number of packets and the number of virtual calls at this link. Instead of using the packet delay as a link length, we propose the use of the virtual link length in the learning automata routing. We show via simulation that this virtual link length is superior to the minimum packet delay or shortest-queue-type link length, usually used in real networks [2, 3, 12, 13, 16, 17, 19, 25, 30]. Using the virtual link length in the routing decisions results in smaller average packet delay than using only the number of packets, or only the number of virtual calls, or the packet delay. Consequently, incorporating both the packet and the virtual call traffic characteristics into the routing decisions is important for improved network performance.

Furthermore, when the routing algorithm is updated infrequently based on obsolete network state information, then the information about the number of virtual calls is more reliable than the information about the number of packets at the link. Therefore, in this case, the virtual link length should be based more on the number of virtual calls than on the number of packets at this link. Finally, when there are few virtual calls and each one carries a large number of packets, the impact of a new virtual call on the performance of the links that it will use is large. Again, in this case, the virtual link length should be based more on the number of virtual calls than on the number of packets at this link. On the other hand, when there are many virtual calls and each one carries a small number of packets, the impact of a new virtual call on the performance is small. In this case, the virtual link length should be based more on the number of packets than on the number of virtual calls at this link.

References

[1] B. Akselrod and G. Langholz. A simulation study of advanced routing methods in a multipriority telephone network. *IEEE Trans. on Systems, Man, and Cybernetics, Vol. SMC-15, No. 6*, pp.730-736, Nov./Dec. 1985.

[2] F. Amer and Y.-N. Lien. A survey of hierarchical routing algorithms and a new hierarchical hybrid adaptive routing algorithm for large scale computer communication networks. *Proc. IEEE ICC '88*, pp. 999-1003, 1988.

[3] P. Brown, J. Roumilhac, and P. Bonnard. A study of the TRANSPAC routing algorithm. *Teletraffic Science for New Cost-Effective Systems, Networks and Services*, ITC-12, M. Bonatti (editor), pp. 1033-1039, Elsevier Science Publ. 1989.

[4] P. Chemouil, M. Lebourges, and P. Gauthier. Performance evaluation of adaptive traffic routing in a metropolitan network: a case study. *Proc. IEEE Globecom '89*, pp. 314- 318, 1989.

[5] M.S. Chrystall and P. Mars. Adaptive routing in computer communication networks using learning automata. *Proc. of IEEE Nat. Telecomm. Conf.*, pp. A3.2.1-7, 1981.

[6] A. A. Economides, P. A. Ioannou, and J. A. Silvester. Dynamic routing and admission control for virtual circuit networks. *Journal of Network and Systems Management*, Vol 2, No 2, 1995.

[7] A. A. Economides, P. A. Ioannou, and J. A. Silvester. Adaptive virtual circuit routing. *Computer Networks and ISDN Systems*, Vol. 27, 1995.

[8] A.A. Economides. A unified game-theoretic methodology for the joint load sharing, routing and congestion control problem. *Ph.D. Dissertation, University of Southern California, Los Angeles*, August 1990.

[9] A.A. Economides. Multiple response learning automata. *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 26, No. 1, pp. 153-156, 1996.

[10] A.A. Economides, P.A. Ioannou, and J.A. Silvester. Decentralized adaptive routing for virtual circuit networks using stochastic learning automata. *Proc. of IEEE Infocom 88 Conference*, pp. 613-622, IEEE 1988.

[11] A.A. Economides and J.A. Silvester. Optimal routing in a network with unreliable links. *Proc. of IEEE Computer Networking Symposium*, pp. 288-297, IEEE 1988.

[12] M. Epelman and A. Gersht. Analytical modeling of GTE TELENET dynamic routing. *Teletraffic Issues in an Advanced Information Society ITC-11*, M. Akiyama (editor), Elsevier Science Publ. 1985.

[13] M. Gerla. Controlling routes, traffic rates, and buffer allocation in packet networks. *IEEE Communications Magazine*, Vol. 22, No. 11, pp. 11-23, Nov. 1984.

[14] R.M. Glorioso, G.R. Grueneich, and J.C. Dunn. Self organization and adaptive routing for communication networks. *EASCON '69 Record*, pp.243-250, 1969.

[15] R.M. Glorioso, G.R. Grueneich, and D. McElroy. Adaptive routing in a large communication network. *Proc. 9th Symposium on Adaptive Processes*, pp. XV.5.1-XV.5.4, 1970.

[16] W.-N. Hsieh and I. Gitman. Routing strategies in computer networks. *IEEE Computer*, pp. 46-56, June 1984.

[17] A. Khanna and J. Zinky. The revised ARPANET routing metric. *Proc. Communication Architectures and Protocols*, pp. 45-56, ACM 1989.

[18] L.G. Mason. Equilibrium flows, routing patterns and algorithms for store-and-forward networks. *Large Scale Systems*, Vol. 8, pp. 187-209, 1985.

[19] J. Moy. OSPF: Next generation routing comes to TCP/IP networks. *LAN Technology*, pp. 71-79, April 1990.

[20] K. Narendra and M.A.L. Thathacher. *Learning Automata: An Introduction.* Prentice Hall, 1989.

[21] K.S. Narendra and P. Mars. The use of learning algorithms in telephone traffic routing - a methodology. *Automatica, Vol. 19, No. 5*, pp. 495-502, 1983.

[22] K.S. Narendra and M.A.L. Thathachar. On the behavior of a learning automaton in a changing environment with application to telephone traffic routing. *IEEE Trans. on Systems, Man and Cybernetics*, Vol. SMC-10, No. 5, May 1980.

[23] K.S. Narendra and M.A.L. Thathachar. Learning automata : A survey. *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. SMC-4, No. 4, pp. 323-334, July 1974.

[24] K.S. Narendra, E.A. Wright, and L.G. Mason. Application of learning automata to telephone traffic routing and control. *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. SMC-7, No.11, pp. 785-792, Nov. 1977.

[25] Th. Narten. Internet routing. *Proc. Communication Architectures and Protocols*, pp. 271-282, ACM 1989.

[26] O.V.Jr. Nedzelnitsky and K.S. Narendra. Nonstationary models of learning automata routing in data communication networks. *IEEE Tr. on Systems, Man and Cybernetics*, Vol. SMC-17, No. 6, pp. 1004-1015, Nov./Dec. 1987.

[27] P.R. Srikantakumar. Adaptive routing in large communication networks: Probabilistic study. *Proc. IEEE Conf. on Decision and Control*, pp. 398-401, 1981.

[28] P.R. Srikantakumar and K.S. Narendra. A learning model for routing in telephone networks. *SIAM J. Control and Optimization*, vol-20, no 1, Jan. 1982.

[29] J.R. Zgierski and B.J. Oommen. SEAT: an object-oriented simulation environment using learning automata for telephone traffic routing. *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-24, No. 2, pp. 349-356, February 1994.

[30] J. Zinky, G. Vichniac, and A. Khanna. Performance of the revised routing metric in the ARPANET and MILNET. *Proc. MILCOM*, pp. 219-224, IEEE 1989.

Anastasios A. Economides was born and grew up in Thessaloniki, Greece. He received the Diploma degree in Electrical Engineering from Aristotle University of Thessaloniki, in 1984. After receiving a Fulbright and a Greek State Fellowship, he continued for graduate studies at the United States. He received a M.Sc. and a Ph.D. degree in Computer Engineering from the University of Southern California, Los Angeles, in 1987 and 1990, respectively. During his graduate studies, he was a research assistant performing research on routing and congestion control. He is currently an Assistant Professor of Informatics at the University of Macedonia, Thessaloniki. His research interests are in the area of Performance Modeling, Optimization and Control of High-Speed Networks. He is a member of IEEE, ACM, INFORMS, EPY, TEE.