Random Early Detection for Congestion Avoidance in Wired Networks: A Discretized Pursuit Learning-Automata-Like Solution

Sudip Misra, *Member, IEEE*, B. John Oommen, *Fellow, IEEE*,
Sreekeerthy Yanamandra, and Mohammad S. Obaidat, *Fellow, IEEE*

***Abstract*—In this paper, we present a learning-automata-like[1] (LAL) mechanism for congestion avoidance in wired networks. Our algorithm, named as LAL Random Early Detection (LALRED), is founded on the principles of the operations of existing RED congestion-avoidance mechanisms, augmented with a LAL philosophy. The primary objective of LALRED is to optimize the value of the average size of the queue used for congestion avoidance and to consequently reduce the total loss of packets at the queue. We attempt to achieve this by stationing a LAL algorithm at the gateways and by discretizing the probabilities of the corresponding actions of the congestion-avoidance algorithm. At every time instant, the LAL scheme, in turn, chooses the action that possesses the maximal ratio between the number of times the chosen action is rewarded and the number of times that it has been chosen. In LALRED, we simultaneously increase the likelihood of the scheme converging to the action, which minimizes the number of packet drops at the gateway. Our approach helps to improve the performance of congestion avoidance by adaptively minimizing the queue-loss rate and the average queue size. Simulation results obtained using NS2 establish the improved performance of LALRED over the traditional RED methods which were chosen as the benchmarks for performance comparison purposes.**

***Index Terms*—Average queue size, discretized pursuit learning, queue loss, random early detection (RED), stochastic learning automata (LA).**

I. Introduction

ONE OF THE main advantages that wired networks offer is their higher degrees of reliability and better connection strength as compared to their wireless counterparts. However, the performance of wired networks often degrades to a great extent due to congestion in the network. The latter results in an increase in the packet loss and a corresponding significant decrease in the throughput. Although congestion cannot be curbed permanently (since it is determined by the traffic patterns and not by the traffic-routing mechanisms), its adverse effects can be minimized by decreasing the packet drops in the network. Despite the fact that the transmission control protocol (TCP) supports mechanisms such as Slow Start, Congestion Avoidance, and Fast Retransmit and Fast Recovery to decrease the effect of packet loss due to congestion, they are not very effective in curbing down congestion *per se* [4]. Consequently, we believe that alternative congestion-avoidance mechanisms are needed.

To avoid congestion in networks, researchers have advocated the use of active-queue-management (AQM) strategies, in which packets are dropped before the queue gets full. Many AQM techniques, such as the adaptive virtual queue, random early detection (RED), random exponential marking, PI controller, and the blue and stochastic blue [11] schemes, have been reported. Among these existing schemes, RED is one of the most widely used techniques in practice. Philosophically, RED is a congestion-*avoidance* algorithm. This is because it foresees (or anticipates) the congestion by monitoring the average queue size. It also avoids global synchronization by randomly choosing packets to be *marked* or *dropped* before the queue gets full. The performance of RED is known to be sensitive to its parameters such as the MAXimum threshold (MAX_{th}), the MINimum threshold (MIN_{th}), the Maximum packet-marking probability (PMP) (Max_P), and the so-called weighting factor [6], [7]. Before we proceed, we clarify how these parameters affect RED. Let Avg denote the average queue size. Then, we have the following conditions.

1) If $Avg < MIN_{th}$, then no packet drops and marks occur.
2) If $Avg > MAX_{th}$, then all the packets are marked.
3) If $MIN_{th} < Avg < MAX_{th}$, then the packets are randomly marked with a certain probability whose value varies from zero to Max_P, evaluated using (2).
4) Let p_b be an *intermediate* PMP given by

$$p_b \leftarrow Max_P \times \frac{Avg - MIN_{th}}{MAX_{th} - MIN_{th}}. \tag{1}$$

Manuscript received January 29, 2009; revised August 2, 2009. Current version published October 30, 2009. This paper was presented in part at the *Proceedings of AICCSA'09, the 2009 ACS/IEEE International Conference on Computer Systems and Applications*, Rabat, Morocco, May 2009. This work was supported in part by the Department of Science and Technology, Government of India, under Grant SR/FTP/ETA-36/08. This paper was recommended by Associate Editor G. Papadimitriou.

S. Misra is with the School of Information Technology, Indian Institute of Technology, Kharagpur 721 302, India (e-mail: sudipm@iitkgp.ac.in).

B. J. Oommen is with the School of Computer Science, Carleton University, Ottawa, ON K1S 5B6, Canada, and also with the University of Agder, 4876 Grimstad, Norway (e-mail: oommen@scs.carleton.ca).

S. Yanamandra is with the School of Information Technology, Indian Institute of Technology, Kharagpur 721 302, India, and also with Kalinga Institute of Industrial Technology, Bhubaneswar 751 024, India (e-mail: sreekeerthy46@gmail.com).

M. S. Obaidat is with the Department of Computer Science, Monmouth University, West Long Branch, NJ 07764 USA (e-mail: obaidat@monmouth.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMCB.2009.2032363

[1]The reason why the mechanism is not a *pure* LA, but rather why it yet mimics one, will be clarified in the body of this paper.

Then, the *Final* PMP p_a is evaluated as per equation as follows:

$$p_a \leftarrow p_b \times \frac{p_b}{1 - Count \times p_b} \tag{2}$$

where $Count$ denotes the number of the packets last marked.

We observe that the packets are to be marked or dropped before the queue gets full. To be more specific, in order to drop a packet, it should be marked. When the average queue size lies between the minimum and maximum thresholds, packets are marked with a certain probability, and these marked packets are not dropped. However, when the average queue size exceeds the maximum threshold, the packets are marked so that they can be dropped. Finally, it must be noted that as the value of $Count$ increases, the value of p_a also increases [6].

The primary objective of this paper is to minimize the average queue size and to reduce the packet loss at the queue. The LALRED mechanism that we have proposed in this paper, which uses a learning-automata-like (LAL) philosophy, succeeds in achieving this.

A. Contributions

The premise of this paper is that we can utilize a LAL philosophy to optimally manage the queue. The use of "learning" to do this is not entirely new; however, the method by which we have approached this is both novel to the field and to our particular application domain. In this paper, we approach the problem by stationing a LAL mechanism at the gateway. The task of the machine, as in any LA-based problem, is to choose a (locally) optimal action from a set of actions offered to it by the "Environment" in which it operates. The question now is that of determining how we can model the environment in this application domain. This is one of the contributions of this paper, because once this is done, we can use the same methodology for other LA-based solutions. However, rather than use a pure LA, we utilize what we refer to as a LAL machine, in which the estimates are essential to update probabilities so as to determine convergence—as in the recently developed family of *discretized pursuit learning* [19] schemes, but not used to choose the actions.

Thus, we can summarize the specific contributions of this paper as follows.

1) We propose LALRED, the first efficient LAL solution to solving the congestion-avoidance problem in wired networks, the performance of which has been rigorously tested through simulations when compared with RED.
2) LALRED maintains a low-average queue size, the advantage of which is mentioned in Section V.
3) Although the family of *discretized pursuit learning* schemes has been shown to be both extremely accurate and fast, its application to solve other engineering or scientific problems is not well known. We have adapted it to update the probabilities and determine the convergence, and thus, we believe that this result is pioneering when it concerns its applicability to solve a real-life complex problem.

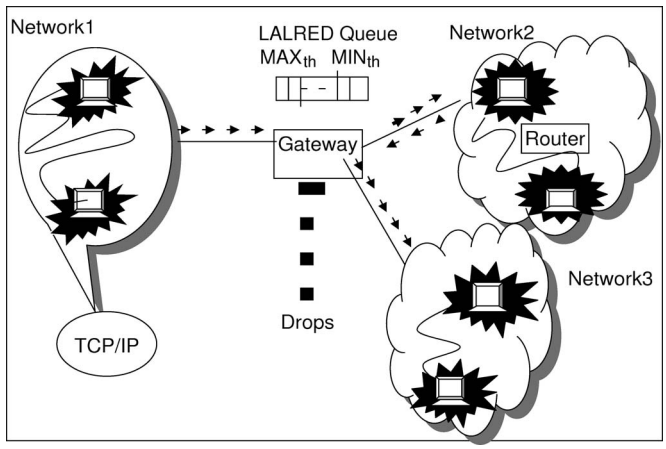


Fig. 1. Transmission of packets from one node to another node in the same or different networks through a gateway. The RED queue is used to avoid incipient congestion in the network.

II. Motivation

To motivate the problem, let us consider Fig. 1, which shows three wired networks, namely, Network1, Network2, and Network3. As shown in the figure, data are sent from Network1 to Network2 and from Network2 to Network3 in the form of packets or bytes from one node to another through the gateway. When congestion occurs in the network, the accepted protocol drops packets at the gateway. RED helps in avoiding congestion. As the average queue size increases (i.e., as it exceeds the maximum threshold), the system drops more packets. It is thus clear that it is advantageous that congestion be minimized for effective communication in the transmission channel. Of course, the optimal situation is that congestion is avoided altogether. However, since this may not be feasible, if it occurs, it should be controlled by the network protocol.

Window-based protocols for flow control are those in which an upper bound on the unacknowledged data sent from the sender to the receiver is set. The flow-control mechanism in the popular transport protocol, TCP, is influenced by the maximum window size allowed by the receiver. This policy permits the sender to send new packets only after receiving the acknowledgment from the receiver for the previous packet. In the late 1980s, three congestion-control algorithms, namely, Slow Start, Congestion Avoidance, and Fast Retransmit and Fast Recovery, were proposed. TCP's congestion-control mechanism defines the following three variables: Congestion_Window (*cwnd*), Slowstart_Threshold, and Advertised_Window (*awnd*). The purpose of *awnd* is to limit the sender from completely using the resources of the receiver. *cwnd*, on the other hand, is used to limit the sender from transmitting more data than what the network can support. The minimum of the quantities *cwnd* and *awnd* is set as the window size of the sender [4]. It should be noted that, despite using these algorithms, it is not possible to completely eliminate congestion in the network. Consequently, researchers sought out congestion-control algorithms to specifically handle this.

To minimize the number of packet drops, it is mandatory that the probability of marking a packet to be dropped is minimized, which implies the minimization of the average queue

size. Thus, the rationale behind our LAL-based approach is to minimize the average size of the queue and that using a LAL mechanism.

III. LA

The functionality of a LA can be described in terms of a sequence of repetitive feedback cycles in which the automaton interacts with the environment. During a cycle, the automaton chooses an action, which triggers a response from the environment, a response that can be either a reward or a penalty. The automaton uses this response and the knowledge acquired in the past actions to determine which is the next action. By learning to choose the optimal action, the automaton adapts itself to the environment. Excellent references that survey the field are the books by Lakshmivarahan [8], Najim and Poznyak [15], Narendra and Thathachar [16], and a recent special issue of the journal IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS, PART B [17].

The learning paradigm, as modeled by LA, has found applications in systems that posses incomplete knowledge about the environment in which they operate [8], [15]–[17]. A variety of applications[2] that use LA have been reported in the literature. They have been used in game playing, pattern recognition, object partitioning, parameter optimization and multiobjective analysis, telephony routing and path planning [12]–[14], [22], and priority assignments in a queuing system. They have also been used in statistical decision making, distribution approximation, natural language processing, modeling biological learning systems, string taxonomy, graph partitioning, distributed scheduling, network protocols (including conflict avoidance) for LANs, photonic LANs, star networks and broadcast communication systems, dynamic channel allocation, tuning PID controllers, assigning capacities in prioritized networks, map learning, digital filter design, controlling client/server systems, adaptive signal processing, vehicle path control, and the control of power systems and vehicle suspension systems.

The beauty of incorporating LA in any particular application domain is, indeed, the elegance of the technology. Essentially, LA is utilized exactly as one would expect—by interacting with the "Environment"—which is the system from which the LA learns. For example, in parameter optimization, the LA chooses a parameter, observes the effect of the parameter in the control loop, and then updates the parameter to optimize the objective function, where this updating is essentially achieved by the LA's stochastic updating rule. The details of how this is achieved in the various application domains involve modeling the "actions" and transforming the system's outputs so that they are perceived to be of a reward or penalty flavor. This is where the ingenuity of the researcher comes into the picture—this is often a thought-provoking task. In the first LA designs, the transition and the output functions were time invariant, and for this reason, these LAs were considered "fixed-structure" automata [16], [17]. Tsetlin, Krylov, and Krinsky presented notable examples of this type of automata. Later, Vorontsova and Varshavskii introduced a class of stochastic automata (SA) known in literature as variable-structure SA (VSSA). In the definition of a VSSA, the LA is completely defined by a set of actions (one of which is the output of the automaton), a set of inputs (one of which is usually the response of the environment), and a learning algorithm T. Within the context of this paper, the VSSA [8], [15]–[17] operates on a vector, called the *action probability vector* $\mathbf{P}(t)$, where

$$\mathbf{P}(t) = [p_1(t), \ldots, p_r(t)]^{\mathrm{T}}$$

where $p_i(t)(i = 1, \ldots, r)$ is the probability that the automaton will select the action α_i at the time t

$$p_i(t) = Pr\left[\alpha(t) = \alpha_i\right],\ i = 1, \ldots, r, \text{ and it satisfies}$$

$$\sum_{i=1}^{r} p_i(t) = 1 \text{ for all } 't'.$$

Note that the algorithm $T : [0, 1]^r \times A \times B \to [0, 1]^r$ is an updating scheme, where $A = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$, $2 \leq r < \infty$, is the set of output actions of the automaton and B is the set of responses from the environment. Thus, the updating is such that

$$\mathbf{P}(t + 1) = T\left(\mathbf{P}(t), \alpha(t), \beta(t)\right) \tag{3}$$

where $\beta(t)$ is the response that the LA receives from the environment.

If the mapping T is chosen in such a manner that the Markov process has absorbing states, the algorithm is referred to as an *absorbing algorithm*. Families of VSSA that posses absorbing barriers have been studied in the literature [8], [15], [16]. Ergodic VSSA have also been reported and extensively studied [8], [15], [16]. These VSSA converge in distribution, and thus, the asymptotic distribution of the action probability vector has a value that is independent of the corresponding initial vector. Thus, while ergodic VSSA are suitable for nonstationary environments, automata with absorbing barriers are preferred in stationary environments.

During the initial years of research in the field of LA, these updating rules worked with the *continuous* probability space. In practice, the relatively slow rate of convergence of these algorithms constituted a limiting factor in their applicability. In order to increase their speed of convergence, the concept of discretizing the probability space was first introduced in [25] and later extensively studied in [1]–[3], [9], [10], [18], [20], [21], [25]. This concept is implemented by restricting the probability of choosing an action to a finite number of values in the interval [0, 1]. Following the discretization concept, many of the continuous VSSA have been discretized.

The family of estimator algorithms are characterized by the use of the estimates for each action. The change of the probability of choosing an action is based on its current estimated mean reward and possibly on the feedback of the environment. The environment determines the probability vector indirectly,

[2] *The applications listed here are few, and the actual bibliographic citations are omitted due to space limitations*. Indeed, this relatively new field has been "exploding." It has recently been enhanced by a spectrum of applications in computer science and engineering—from areas as diverse as the design of data structures, to the implementation of automatic navigation methods.

through the calculation of the reward estimates for each action. Even when the chosen action is rewarded, there is a possibility that the probability of choosing another action is increased.

For the definition of an estimator LA, a vector of reward estimates $\hat{\mathbf{D}}(t)$ must be introduced. Hence, [23], [26], the state vector $\mathbf{Q}(t)$ is defined as

$$\mathbf{Q}(t) = \left\langle \mathbf{P}(t), \hat{\mathbf{D}}(t) \right\rangle$$

where $\hat{\mathbf{D}}(t) = [\hat{d}_1(t), \ldots, \hat{d}_r(t)]$ can be calculated using the following:

$$\hat{d}_i(t) = \frac{W_i(t)}{Z_i(t)}, \qquad \text{for } i = 1, 2, \ldots, r$$

where $W_i(t)$ is the number of times the ith action has been rewarded up to the time t and $Z_i(t)$ is the number of times the ith action has been chosen up to the time t.

Thathachar and Sastry [23], [26] have shown that the estimator algorithms exhibit a superior speed of convergence when compared with the nonestimator algorithms. In 1989, Oommen and Lanctôt [9], [10] introduced discretized versions of the estimator algorithms and have shown that the discretized estimator algorithms are even faster than their continuous counterparts.

For the sake of this paper, we shall concentrate on the family of Pursuit algorithms, characterized by the fact that they pursue the action that is currently estimated to be the optimal action. In the continuous versions, they pursue it in a continuous space, and in the discretized versions, by changing the probabilities in discrete steps. The $\mathrm{DP_{RI}}$ scheme is briefly described as follows.

This scheme, which was introduced by Lanctôt and Oommen [9], is based on the reward–inaction learning "philosophy" and is, thus, denoted by $\mathrm{DP_{RI}}$. The differences between the discrete and continuous version of the Pursuit algorithm occur only in the updating rules for the action probabilities. The discrete Pursuit algorithm makes changes to the probability vector $\mathbf{P}(t)$ in discrete steps, whereas the continuous version uses a continuous function to update $\mathbf{P}(t)$. Being a reward–inaction algorithm, the action probability vector $\mathbf{P}(t)$ is updated only when the current chosen action is rewarded. If the current action is penalized, the action probability vector $\mathbf{P}(t)$ remains unchanged. When the chosen action is rewarded, the algorithm decreases the probability for all the actions that do not correspond to the highest estimate, by the smallest step size Δ, where if N is the so-called resolution parameter, $\Delta = 1/rN$. In order to keep the sum of the components of the vector $\mathbf{P}(t)$ equal to unity, the $\mathrm{DP_{RI}}$ increases the probability of the action with the highest estimate by an integral multiple of the smallest step size Δ.

Oommen and Lanctôt proved that the $\mathrm{DP_{RI}}$ is *ε-optimal* in all stationary random environments.

They also performed simulations of the $\mathrm{DP_{RI}}$ in some benchmark environments, and the results have been compared against the results of a continuous reward–penalty version, the $\mathrm{CP_{RP}}$ algorithm. The results have shown that, in some difficult environments, the $\mathrm{DP_{RI}}$ requires only 50% of the number of iterations required for its continuous version. In a ten-action environment, the $\mathrm{DP_{RI}}$ algorithm required 69% of the iterations required by the $\mathrm{CP_{RP}}$ [9], [10].

IV. LALRED Algorithm

Many schemes that improve the performance of RED by tuning the different parameters, such as control-parameter settings [27], varying the rate of change of the average queue size [5], and improving the response time when RED recovers from congestion [28] have been proposed lately.

In spite of the existence of a number of schemes to avoid congestion, the primary motivation behind this paper was to design a methodology which helps to lower the packet loss due to congestion by adaptively discretizing the drop types based on the maximal component of the reward estimates vector explained in Section III. Our approach should not be construed as "competing" with the existing congestion-avoidance algorithms mentioned earlier. Rather, we try to show how the use of the discretized pursuit learning approach can help in effectively minimizing the average queue size and packet loss at the gateways of the networks.

Before we proceed with our LAL-based solution, we have to clearly indicate the "Actions" which the environment has to offer (which the LAL scheme has to choose from). In our approach, we advocate the following four actions, based on the packet drop type, as follows.

1) *Forced_Drop:* This action is chosen when the average queue size is above the maximum-threshold set for the queue or when the queue is full [24].
2) *Minimum_Exceed:* This action is chosen when the average queue size exceeds the minimum threshold or it transitions from an empty queue state to a nonempty queue state.
3) *Unforced_Drop:* This action is chosen when the average queue size lies between the minimum threshold and the maximum threshold. For an unforced drop, the arriving packet is always dropped.
4) *No_Drop*: This action is chosen when the average queue size lies below the minimum threshold.

By virtue of the nature of RED, we can state that the actions *Forced_Drop, Minimum_Exceed, Unforced_Drop*, and *No_Drop* in LALRED are mutually exclusive. This is because of the following reasons.

1) *Forced_Drop* occurs when $Avg > MAX_{th}$.
2) *Minimum_Exceed* occurs when $MIN_{th} < Avg < Max_{th}$ and when Avg just crosses MIN_{th}.
3) *Unforced_Drop* occurs when $MIN_{th} < Avg < MAX_{th}$.
4) *No_Drop* occurs when $Avg < MIN_{th}$.

The mutually exclusive nature of the actions is because the earlier four cases are themselves mutually exclusive.

The rationale behind our approach is as follows. First of all, we station a LAL machine, which makes its decisions based on a LALRED strategy (see Fig. 2, where we consider two networks: Network 1 and Network 2). Unlike a true LA scheme, LALRED does not use an "action probability" vector

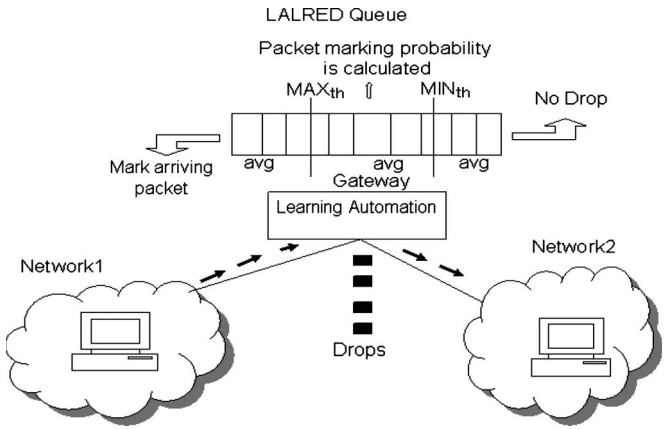


Fig. 2. Example of the transmission of packets from one network to another using a LAL machine placed at the gateway.

to *choose* the actions. Thus, it is not appropriate to consider our scheme to be within the LA family. Rather, at every time instant, we maintain a maximum-likelihood estimate of how profitable a particular action has been, and this is essentially inferred by examining the "estimate vector" stored by the Pursuit algorithm. Thus, the scheme intelligently chooses the randomly arriving packets to mark, drop, or not drop, based on the current average queue size in the network. A locally optimal action is chosen (from this set of possible actions) by the machine at every iteration. This is enabled by the continuous interaction of the automaton, stationed at the gateway, with the environment.

The algorithm also maintains and updates a probability vector analogous to the "action probability" vector used by LA. What then is the use of this vector, if truly, it is not used to choose the actions? Indeed, it is used to control the convergence accuracy of the mechanism. For example, let us suppose that we have used the DP_{RI} action probability updating scheme discussed in Section III to update the probability vector. Unlike the DP_{RI}, the action chosen is not based on the "action probability" vector but rather based on the maximal component of the reward estimates vector. The response generated by the environment signifies whether the action chosen is to be given a "reward" or a "penalty." Indeed, when the action *No_Drop* is chosen by the LAL system, it gets a reward; otherwise, it gets a penalty. When an action chosen by the automaton is rewarded by the environment, the probabilities corresponding to all the actions are updated, and so, more specifically, the probability corresponding to the rewarded action is increased, and the probabilities of the other actions are decreased by a single step-size Δ. If the action chosen is penalized, since the scheme is of a reward–inaction flavor, the action probabilities are unchanged. This process iterates until an action probability converges to unity (i.e., the vector becomes a unit vector).

Consider now the effect of the step size Δ. If Δ is large, the action probabilities will take large steps, and thus, the scheme will converge to a unit vector more rapidly, although the convergence could be less accurate. On the other hand, if Δ is small, the changes will be more conservative and measured, and thus, it will take a larger number of iterations for the scheme to converge to a unit vector, leading to a more accurate convergence. We, thus, have the traditional speed-accuracy conflict, and our experimental results show that we can choose a value of Δ that is small enough to guarantee an accurate-enough convergence.

We establish through rigorous simulation studies that LALRED can help minimize the average queue size and the packet loss rate significantly. Although LALRED is effective in reducing the queue size, it is not as effective in improving the throughput. Indeed, in case of the throughput, the LALRED algorithm does not outperform RED significantly. The main objective of LALRED is to optimize the average queue size. Our position is that we have to do this to both avoid congestion in the network and maximize the number of packets sent with respect to time. The formal algorithm LALRED is as follows.

ALGORITHM LALRED

Input: Set of actions $\{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$, where
α_1: Forced_Drop
α_2: Minimum_Exceed
α_3: Unforced_Drop
α_4: No_Drop.

Output: The output is the DROPTYPE action chosen by the automaton.

Parameters: m: Index of the maximal component of $\hat{\mathbf{D}}(t)$, $\hat{d}_m(t) = \max_{i=1,2,\ldots,r}\{\hat{d}_i(t)\}$
$W_i(t)$: The number of times the ith action has been rewarded up to the time t, with $1 \leq i \leq r$
$Z_i(t)$: The number of times the ith action has been chosen up to the time t, with $1 \leq i \leq r$
N: The resolution parameter
$\Delta := 1/rN$ is the smallest step size

Method

Initialization

$p_i(t) = 1/r$, for $1 \leq i \leq r$
Step 0: Initialize $\hat{\mathbf{D}}(t)$ by picking each action a small number of times
Initially, choose an action α_m based on the average initial queue size

Repeat

Step 1: The feedback β of the environment is given to the action α_m chosen by the machine. When the action *No_Drop* is chosen, it gets a reward $(\beta = 0)$; otherwise, it gets a penalty $(\beta = 1)$.
Step 2: Update $\mathbf{P}(t)$ according to the following equations:
For all $j \neq m$,
If $\beta(t) = 0$ and $p_m(t) \neq 1$

$$p_j(t+1) = \max\{p_j(t) - \Delta, 0\}$$

$$p_m(t+1) = 1 - \sum_{\forall j \neq m} p_j(t+1)$$

Else

$$p_j(t+1) = p_j(t) \qquad \text{for all } 1 \leq j \leq r.$$

Step 3: Update $\hat{\mathbf{D}}(t)$ according to the following:

$$W_m(t+1) = W_m(t) + (1 - \beta(t))$$
$$Z_m(t+1) = Z_m(t) + 1$$
$$\hat{d}_m(t+1) = \frac{W_m(t+1)}{Z_m(t+1)}$$
$$\left.\begin{array}{l} W_j(t+1) = W_j(t) \\ Z_j(t+1) = Z_j(t) \\ \hat{d}_j(t+1) = \hat{d}_j(t) \end{array}\right\} \text{ for all } j \neq m.$$

Step 4: Choose an action based on the maximal component of $\hat{\mathbf{D}}(t)$.
Step 5: Drop the packets based on the action chosen by the automaton.
Step 6: Repeat Steps 1–5 until some component $p_m(t) \rightarrow 1$
End Repeat
END ALGORITHM LALRED

V. Experimental Details

In this section, we present the details of the three sets of simulation experiments that were conducted to evaluate the performance of LALRED. We describe the network topology and the performance metrics that were used in the experiments, and then, we present the results obtained from the simulations.

A. Simulation Results

The performance evaluation of our proposed algorithm, LALRED, as compared with the traditional RED algorithm, was performed using the NS2 simulator (version NS-2.29) and TCP traffic. The algorithms were, initially, simulated for a network of six nodes and then for another with 100 nodes. The simulation results reported in this paper are for an ensemble average of ten runs.

The performance plots of LALRED and RED are presented in the following sections. The simulation results show that the performance of LALRED is superior by virtue of its low-average queue size and its packet loss. Both of these decrease significantly using our approach.

B. Performance Study

The following metrics were used to comparatively evaluate the performances of RED and LALRED.

1) **Queue Size**: This metric quantifies the size of the queue. It is measured in terms of the number of packets or bytes.
2) **Queue Lost**: The queue-lost metric helps to evaluate the number of packets lost at the gateway due to congestion in the network. As the number of packets lost increases, the delay for a receiver to receive the message also increases.

C. Results

Three sets of experiments were conducted to establish the performance of LALRED. In the first set, the instantaneous queue size, the average queue size, and the number of all the packets transmitted were plotted in a network having six nodes but having different bandwidth and delay. In the second set of

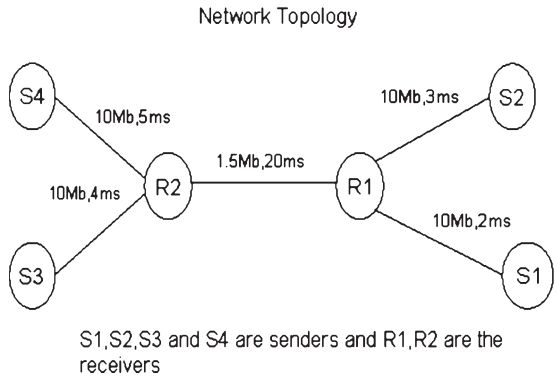


Fig. 3. Topology of the network having six nodes used in the first two experiments.

TABLE I
PARAMETERS USED FOR EXPERIMENTAL SET 1

Parameter	Value
Bandwidth	10Mb, 1.5Mb
Delay	20, 5, 4, 3, 2 (all in ms)
Queue weight	0.002 / -1
Threshold	5
Maximum threshold	15
Maximum current window size	16
Queue limit	100
Packet size	1,000 bytes
Maximum probability of marking the packets	1.0
Simulation time	10.0s

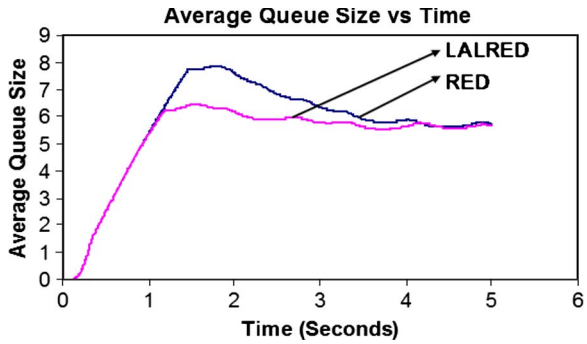


Fig. 4. Graphs of the *average queue size* for RED and LALRED for experimental set 1.

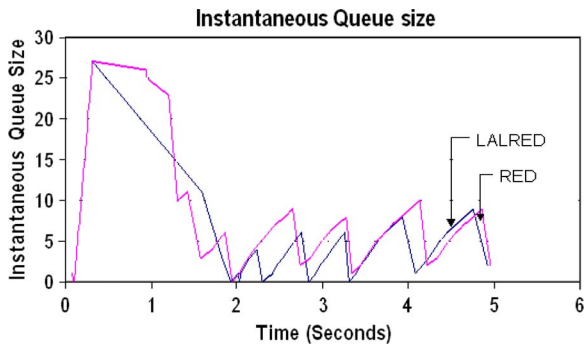


Fig. 5. Graphs of the *instantaneous queue size* for RED and LALRED for experimental set 1.

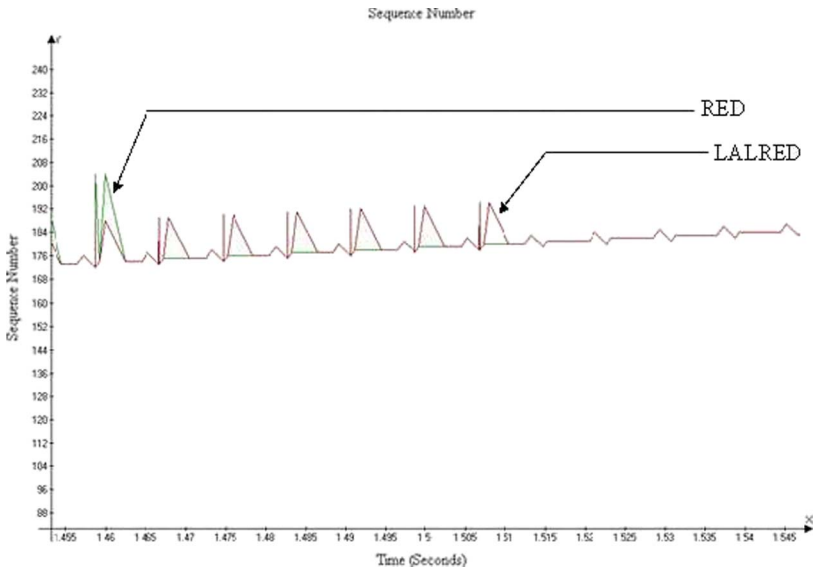


Fig. 6. Graphs of the *sequence number* for RED and LALRED for experimental set 1.

experiments, the queue lost and the queue size were determined and compared in a network having six nodes. The bandwidth and delay in this case were different from those characterizing the network used in the first set of experiments. The third set of experiments were conducted with a network having 100 nodes, and the queue lost and the queue size were analyzed.

Experimental Set 1: The network topology that we used in this set of experiments is shown in Fig. 3, which also clearly displays the senders and the receivers. The figure also clearly shows the delays and bandwidths of all the links. This topology is also the one used in the second set of experiments, except that, as mentioned later, in that case, the delays and bandwidths of all the links were specified as per Fig. 3 and Table I. With regard to the queues maintained between R1 and R2, we note that the queues between (S1, R1), (S2, R1), (S3, R2), and (S4, R2) were of the type "DropTail," i.e., the packets were dropped from the *tail* of the queue.

Some of the configuration parameters that we used in this set of experiments are given in Table I. In the figure and table, *Bandwidth* denotes the bandwidth of the links in the network, *Delay* denotes the propagation delay on a link, and the *Queue limit* denotes the maximum limit on the size of the queue. The *Threshold* and the *Maximum threshold* parameters denote the predefined minimum and the maximum threshold values set for a queue. The *Number of Flows* denotes the total number of flows among all pairs of nodes. The *Simulation time* parameter denotes the duration for which the simulation experiments are run. The *Packet size* denotes the size of the packets transmitted. The results obtained are given as follows.

Average and Instantaneous Queue Sizes: Figs. 4 and 5 show the variation of the average and instantaneous queue sizes and the queue lost of RED and LALRED. It should be noted that, generally speaking, as the average queue size increases, the number of packet drops also increases, because in such a case, whenever the average queue size exceeds the maximum threshold, the packet drops increases. Moreover, the average queue size is used instead of the instantaneous queue size

TABLE II
PARAMETERS USED FOR EXPERIMENTAL SET 2

Parameter	Value
Bandwidth	10Mb
Delay	2ms
Queue limit	25
Threshold	5
Maximum threshold	15
Number of Nodes	6
Number of Flows	10
Simulation time	20.0s
Packet size	1,000 bytes
Maximum probability of marking packets	0.5
Queue type	DropTail or RED

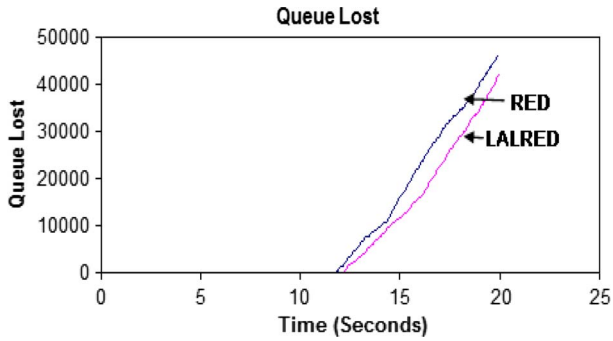


Fig. 7. Graphs of the *queue lost* for RED and LALRED for experimental set 2.

to attenuate transient congestion. Hence, the goal of a good algorithm should be to keep the average queue size minimal. From the figure, it can be observed that, from time $t = 1$ to $t = 4$, the average queue size of LALRED is lesser than that of RED. This small queue size can be attributed to the low delay in the network. The mean of the average queue size of LALRED calculated for this plot is 5.024037, whereas that of RED is 5.442317. Thus, LALRED's average queue size is 7.68% lesser than that of RED's. The instantaneous queue size for LALRED is also almost always less than that for RED.

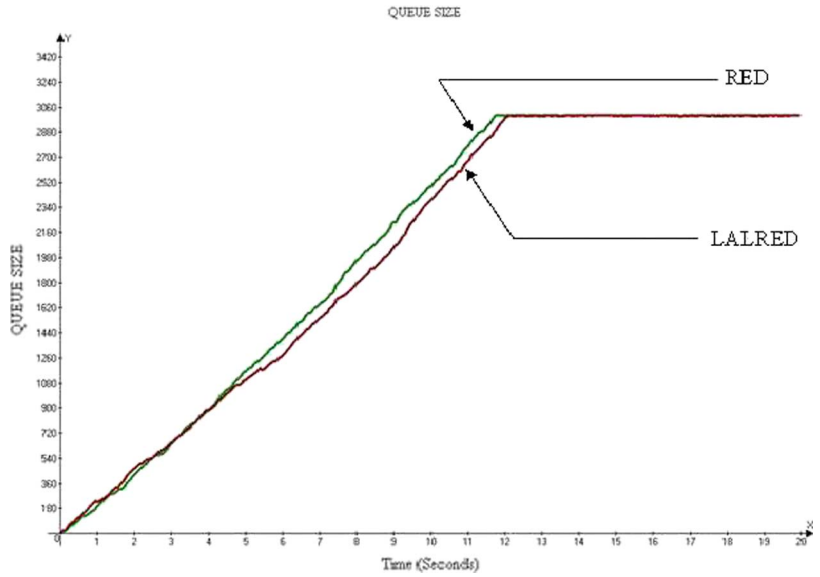


Fig. 8. Graphs of the *queue size* for RED and LALRED for experimental set 2.

Sequence Number: Fig. 6 shows the variation of the *sequence* number of the packets sent for both RED and LALRED. This index, the *sequence* number, signifies the number of packets acknowledged as having been received by the receiver. It thus denotes the number of the last packet sent. From the plot, we see that, almost uniformly and on an average, the *sequence* number associated with LALRED is greater than that for RED. Hence, we can infer that LALRED is superior to RED in terms of the number of packets that are acknowledged.

Experimental Set 2: In experimental set 2, we considered a network having a set of six nodes, whose topology is the same as that used in experimental set 1. The links present in the network had a bandwidth of 10 Mb and a delay of 2 ms. In this set of experiments, the quantities measured were the queue lost and the queue size. The important simulation parameters that were used in this experimental set are summarized in Table II.

Queue Lost: As queue lost signifies the number of packets lost with respect to time, an increased value for this index would imply an increased difficulty in reassembling the original message at the destination. This would thus increase the delay of transmission in the network. For an ideal scenario, the queue-lost value should be zero. Fig. 7 shows the comparison of queue lost of the RED and LALRED schemes. From the figure, it is shown that the curve of LALRED is almost always lower than that of RED. This signifies that the queue lost for RED is greater than that of LALRED. For instance, it can be observed that the value of the queue lost for RED at 19.95 s is 46 100 B. On the other hand, the queue lost for LALRED at the same time instant is 42 100 B. This implies a decrease of 8.676% for LALRED over RED.

Queue Size: The size of the queue should be optimized to such an extent that both the packet loss at the queue and the delay are minimized. Fig. 8 shows the variation of queue size for both RED and LALRED. From the figure, it is again shown that the curve corresponding to LALRED is always lower than that of RED. Thus, the queue size at any instant, for example, 5.1499 s, in case of RED, is 1194, whereas that of LALRED is 1136. The general observation is that the queue size of LALRED is generally smaller than that of RED.

TABLE III
PARAMETERS USED FOR EXPERIMENTAL SET 3

Parameter	Value
Queue weight	0.002/-1
Threshold	5
Maximum threshold	15
Bandwidth	2Mb
Delay	1ms
Number of flows	530
Maximum current window size	16
Queue-limit	3000
Packet size	1000 bytes
Maximum probability of marking packet	0.5
Simulation time	20.0
Number of nodes	100

Experimental Set 3: In experimental set 3, we tested the performance of LALRED and RED when the number of nodes is increased. In this set of experiments, we used a network with 100 nodes. The links between the nodes had a bandwidth of 2 Mb and a delay of 1 ms. The simulation parameters set in this set of experiments are summarized in Table III.

Queue Size: Fig. 9 shows the variation of queue size for RED and LALRED when the *Queue Type* is DropTail. The experimental results show that the average queue size of LALRED is marginally lesser than that of RED, by about 0.12%. This is much smaller than what was observed in experimental set 1.

Queue Lost: Figs. 10 and 11 show the performance results corresponding to the queue-lost metric. From the plots in the figures, it can be observed that the queue lost in case of RED is more than the queue lost in case of LALRED. For example, the value of queue lost at 19.95 s of RED is about 2 086 000, whereas the same index for LALRED is about 2 063 880. Again, the superiority is not as marked in this experimental set as compared to that in experimental set 1.

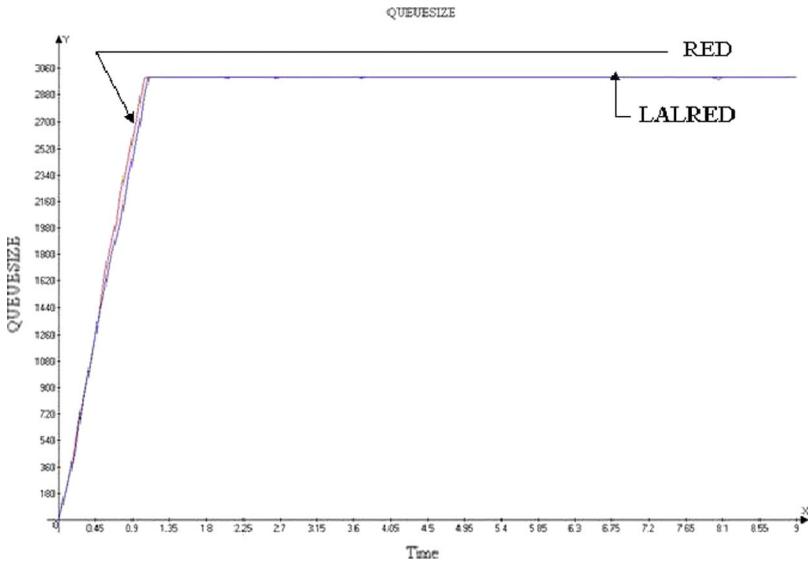


Fig. 9. Queue sizes of RED and LALRED for experimental set 3 when the *queue type* is DropTail.

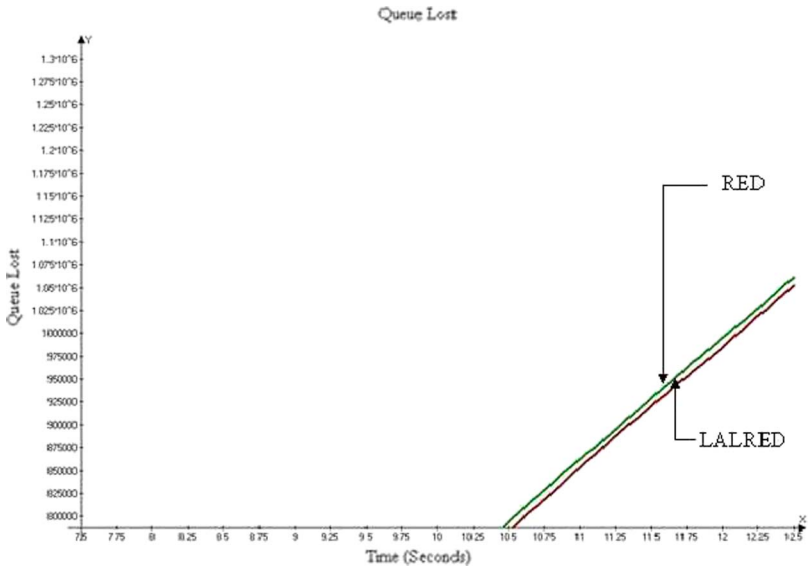


Fig. 10. Graphs of the *queue lost* of RED and LALRED for experimental set 3 when the *queue type* is DropTail.

VI. Conclusion

In this paper, we have devised a LAL mechanism for congestion avoidance in wired networks. Our algorithm, LALRED, is not a pure LA family because it does not use the action probability vector to choose the action at any given time instant. Rather, it uses the so-called estimate vector maintained by the family of Pursuit algorithms and updates the probability vector using a discretized philosophy so as to move toward convergence. LALRED is founded on the principles of the operations of existing RED congestion-avoidance mechanisms, augmented with a LAL philosophy, and it aims to optimize the value of the average size of the queue used for congestion avoidance and to consequently reduce the total loss of packets at the queue. Simulation results obtained using NS2 establish the improved performance of LALRED over the traditional RED methods which were chosen as the benchmarks for performance comparison purposes. From these, we infer the following results.

1) The number of packets lost at the gateway using LALRED is lower as compared to that using RED. LALRED reduces the packet drops at the gateway.
2) The average queue size maintained when using LALRED is lower as compared to that using RED. The average queue size is proportional to the dropping probability at the gateway.
3) Using LALRED, more packets are acknowledged to the sender.
4) For more complex subnets, it may happen that the transfer of packets between the nodes takes place more than in less complex networks. However, our observation is that the average throughput of LALRED is less than that of the conventional RED. Furthermore, the average queue size

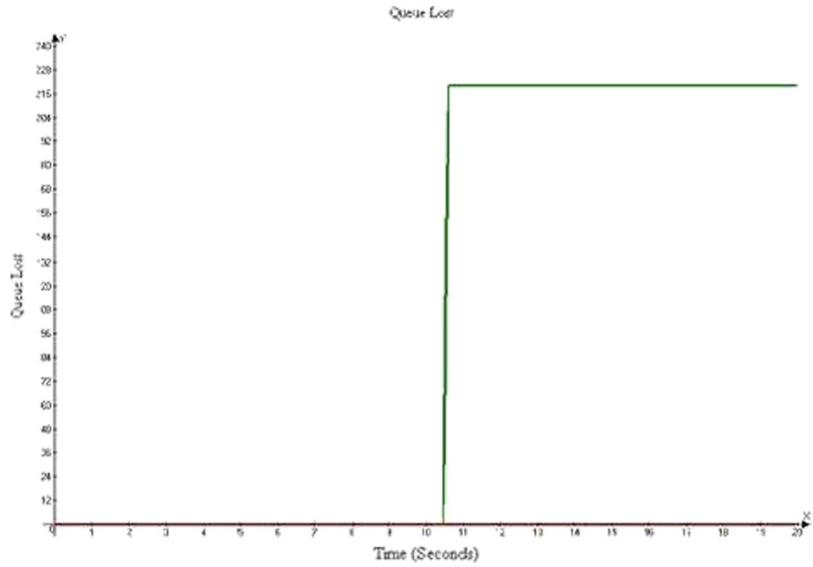


Fig. 11. Graphs of the *queue lost* of RED and LALRED for experimental set 3 when the *queue type* is RED.

and the number of packets lost at any particular instant of time are less than that of normal RED. Our position is that, for even networks with 100 nodes, the LALRED scheme is superior to the RED.

We suggest a few ideas for future research by which LALRED can be enhanced.

1) The question of using the action probability vector and the estimate vector to choose the actions is the first promising avenue for further research. This is not going to be so trivial because the action probability vector may recommend the choice of an action contrary to the choice recommended by reward estimate vector. This research is currently being undertaken.
2) It would be interesting to see how LALRED performs in environments characterized by nodes that are mobile.
3) An entirely new approach along the lines of thought used in LALRED or a modified LALRED needs to be proposed for congestion avoidance in both infrastructure-based and infrastructureless wireless networks.
4) The scalability of LALRED for use in networks having a large number of nodes needs to be improved.
5) The performance of LALRED in fault-prone environments would also need detailed investigation.

REFERENCES

[1] M. Agache, "Families of estimator-based stochastic learning algorithms," M.S. thesis, School Comput. Sci., Carleton Univ., Ottawa, ON, Canada, 2000.

[2] M. Agache and B. J. Oommen, "Continuous and discretized generalized pursuit learning schemes," in *Proc. 4th SCI*, S. Andraddttir, K. J. Healy, D. H. Withers, and B. L. Nelson, Eds., Orlando, FL, Jul. 2000, pp. VII:270–VII:275.

[3] M. Agache and B. J. Oommen, "Generalized pursuit learning schemes: New families of continuous and discretized learning automata," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 32, no. 6, pp. 738–749, Dec. 2002.

[4] J. Antila, *TCP Performance Simulations Using NS2*, Last accessed on Feb. 3, 2009. [Online]. Available: http://www.netlab.tkk.fi/j˜mantti3/pubs/special study.pdf

[5] R. Fengyuan, R. Yong, and S. Xiuming, "Enhancement to RED algorithm," in *Proc. 9th IEEE Int. Conf. Netw.*, Oct. 2001, pp. 14–19.

[6] S. Floyd and V. Jacobson, "Random Early Detection gateways for congestion avoidance," *IEEE/ACM Trans. Netw.*, vol. 1, no. 4, pp. 397–413, Aug. 1993.

[7] D. Katabi and M. Handley, "Enhancing Internet congestion control using explicit and precise feedback," in *Proc. Oxygen Workshop*, Jul. 2001, pp. 1–2.

[8] S. Lakshmivarahan, *Learning Algorithms Theory and Applications*. New York: Springer-Verlag, 1981.

[9] J. K. Lanctôt, "Discrete estimator algorithms: A mathematical model of computer learning," M.S. thesis, Dept. Math. Stat., Carleton Univ., Ottawa, ON, Canada, 1989.

[10] J. K. Lanctôt and B. J. Oommen, "Discretized estimator learning automata," *IEEE Trans. Syst., Man, Cybern.*, vol. 22, no. 6, pp. 1473–1483, Nov./Dec. 1992.

[11] *Active Queue Management*, Last accessed on Feb. 3, 2009. [Online]. Available: http://en.wikipedia.org/wiki/active queue management

[12] S. Misra and B. J. Oommen, "GPSPA: A new adaptive algorithm for maintaining shortest path routing trees in stochastic networks," *Int. J. Commun. Syst.*, vol. 17, no. 10, pp. 963–984, Dec. 2004.

[13] S. Misra and B. J. Oommen, "An efficient dynamic algorithm for maintaining all-pairs shortest paths in stochastic networks," *IEEE Trans. Comput.*, vol. 55, no. 6, pp. 686–702, Jun. 2006.

[14] S. Misra and B. J. Oommen, "Fault-tolerant routing in adversarial mobile Ad Hoc networks: An efficient route estimation scheme for non-stationary environments," *Telecommun. Syst. J.*, 2009, to be published.

[15] K. Najim and A. S. Poznyak, *Learning Automata: Theory and Applications*. Oxford, U.K.: Pergamon, 1994.

[16] K. S. Narendra and M. A. L. Thathachar, *Learning Automata: An Introduction*. Englewood Cliffs, NJ: Prentice-Hall, 1989.

[17] M. S. Obaidat, G. I. Papadimitriou, and A. S. Pomportsis, "Learning automata: Theory, paradigms, and applications," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 32, no. 6, pp. 706–709, Dec. 2002.

[18] B. J. Oommen, "Absorbing and ergodic discretized two-action learning automata," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-16, no. 2, pp. 282–293, Mar./Apr. 1986.

[19] B. J. Oommen and M. Agache, "Continuous and discretized pursuit learning schemes: Various algorithms and their comparison," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 31, no. 3, pp. 277–287, Jun. 2001.

[20] B. J. Oommen and J. P. R. Christensen, "ϵ-optimal discretized reward–penalty learning automata," *IEEE Trans. Syst., Man, Cybern.*, vol. 18, no. 3, pp. 451–457, May/Jun. 1988.

[21] B. J. Oommen and E. R. Hansen, "The asymptotic optimality of discretized linear reward–inaction learning automata," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-14, no. 3, pp. 542–545, May/Jun. 1984.

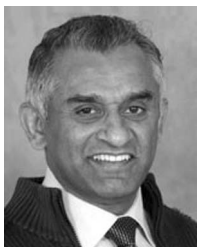
[22] B. J. Oommen, S. Misra, and O.-C. Granmo, "Routing bandwidth guaranteed paths in MPLS traffic engineering: A multiple race track learning approach," *IEEE Trans. Comput.*, vol. 56, no. 7, pp. 959–976, Jul. 2007.

[23] P. S. Sastry, "Systems of learning automata: Estimator algorithms applications," Ph.D. dissertation, Dept. Elect. Eng., Indian Inst. Sci., Bangalore, India, Jun. 1985.
[24] *NS2 Network Simulator*, Last accessed on Feb. 3, 2009. [Online]. Available: http://www.isi.edu/nsnam/ns/
[25] M. A. L. Thathachar and B. J. Oommen, "Discretized reward–inaction learning automata," *J. Cybern. Inf. Sci.*, vol. 2, no. 1, pp. 24–29, Spring 1979.
[26] M. A. L. Thathachar and P. S. Sastry, "A class of rapidly converging algorithms for learning automata," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-15, no. 1, pp. 168–175, Jan./Feb. 1985.
[27] R. Verma, A. Iyer, and A. Karandikar, "Active queue management using adaptive RED," *J. Commun. Netw.*, vol. 5, no. 3, pp. 275–281, Sep. 2003.
[28] B. Zheng and M. Atiquzzaman, "Low pass filter/over drop avoidance (LPF/ODA): An algorithm to improve the response time of RED gateways," *Int. J. Commun. Syst.*, vol. 15, no. 10, pp. 899–906, Dec. 2002.

Sudip Misra (M'09) received the B.S. degree from the Indian Institute of Technology, Kharagpur, India, the M.S. degree from the University of New Brunswick, Fredericton, NB, Canada, and the Ph.D. degree in computer science from Carleton University, Ottawa, ON, Canada.

He has several years of experience working in the academia, government, and private sectors in research, teaching, consulting, project management, architecture, software design, and product engineering roles. He was with Cornell University, Ithaca, NY; Yale University, New Haven, CT; Nortel Networks, Canada; and the Government of Ontario, Canada. He is currently an Assistant Professor with the School of Information Technology, Indian Institute of Technology. He is the author/Editor of a large number of scholarly research papers and books. His current research interests include algorithm design and engineering for telecommunication networks, software engineering for telecommunication applications, and computational intelligence and soft-computing applications in telecommunications.

Dr. Misra was the recipient of six research paper awards in different conferences. He was also the recipient of several academic awards and fellowships such as the (Canadian) Governor General's Academic Gold Medal at Carleton University and the University Outstanding Graduate Student Award in the doctoral level at Carleton University. In 2008, he was conferred The National Academy of Sciences, India—Swarna Jayanti Puraskar. He is the Editor-in-Chief of two journals and is an Associate Editor or an editorial board member of a dozen others published by Elsevier, Springer, Wiley, IOS Press, etc. He is an Editor of six books in the areas of wireless ad hoc networks, wireless sensor networks, wireless mesh networks, communication networks and distributed systems, network reliability and fault tolerance, and information and coding theory, published by reputed publishers such as Springer and World Scientific. He was invited to chair several international conference/workshop programs and sessions. He was also invited to deliver keynote/invited lectures in over 15 international conferences in the U.S., Canada, Europe, Asia, and Africa.



B. John Oommen (F'03) was born in Coonoor, India, on September 9, 1953. He received the B.Tech. degree from the Indian Institute of Technology, Madras, India, in 1975, the M.E. degree from the Indian Institute of Science, Bangalore, India, in 1977, and the M.S. and Ph.D. degrees from Purdue University, West Lafayettte, IN, in 1979 and 1982, respectively.

In the 1981–1982 academic year, he joined the School of Computer Science, Carleton University, Ottawa, ON, Canada, where he is currently a Full Professor and, since July 2006, has been awarded the honorary rank of Chancellor's Professor, which is a lifetime award. He is the author of more than 315 refereed journal and conference publications. His research interests include automata learning, adaptive data structures, statistical and syntactic pattern recognition, stochastic algorithms, and partitioning algorithms.

Dr. Oommen is a Fellow of the IAPR. He has been on the editorial board of the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS, and *Pattern Recognition*.

Sreekeerthy Yanamandra received the B.S. degree in computer science and engineering from Kalinga Institute of Industrial Technology, Bhubaneswar, India.

She is currently an Assistant Software Engineer with Tata Consultancy Services, India. Her current research interests include computer networks and learning systems.

Mohammad S. Obaidat (F'05) received the M.S. and Ph.D. degrees in computer engineering with a minor in computer science from The Ohio State University, Columbus.

He is an internationally well-known Academic, Researcher, and Scientist. He was a Faculty Member with the City University of New York, New York. He is currently a Full Professor of computer science with the Department of Computer Science, Monmouth University, West Long Branch, NJ, where he was previously the Chair of the Department of Computer Science and the Director of the Graduate Program. He has received extensive research funding. He has served as a Consultant for several corporations and organizations worldwide. In 2002, he was the Scientific Advisor for the World Bank/UN Workshop on Fostering Digital Inclusion. During the 2004/2005 academic year, he was on sabbatical leave as the Fulbright Distinguished Professor and Advisor to the President of Philadelphia University, Philadelphia, PA (Dr. Adnan Badran who became in April 2005 the Prime Minister of Jordan). He has made pioneering and lasting contributions to the multifacet fields of computer science and engineering. He has authored or coauthored six books and over 420 refereed scholarly journal and conference articles. His research interests include wireless communications and networks, modeling and simulation, performance evaluation of computer systems and telecommunications systems, security of computer and network systems, high-performance computing/computers, applied neural networks and pattern recognition, security of e-based systems, and speech processing.

Dr. Obaidat is the Editor of many scholarly journals, including being the Editor-in-Chief of the *International Journal of Communication Systems* (John Wiley). He is also an Editor of the IEEE WIRELESS COMMUNICATIONS. Recently, he was the recipient of the Distinguished Nokia Research Fellowship and the Distinguished Fulbright Award. He has guest-edited numerous Special Issues of scholarly journals such as the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS, IEEE WIRELESS COMMUNICATIONS, IEEE SYSTEMS JOURNAL, *Elsevier Performance Evaluation, SIMULATION: Transactions of Society for Modeling and Simulation International (SCS)*, *Elsevier Computer Communications Journal*, *Journal of C&EE*, *Wiley Security and Communication Network Journal*, *Wiley International Journal of Communication Systems*, and among others. He has served as the Steering Committee Chair, Advisory Committee Chair, Honorary Chair, and Program Chair of many international conferences. He is the Founder of the International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS) and has served as the General Chair of SPECTS since its inception. He was the recipient of a recognition certificate from the IEEE. Between 1994 and 1997, he has served as Distinguished Speaker/Visitor of the IEEE Computer Society. Since 1995, he has been serving as an Association for Computing Machinery (ACM) Distinguished Lecturer. He is currently also an SCS Distinguished Lecturer. He is the Founder of the SCS Distinguished Lecturer Program and its current Director. Between 1996 and 1999, he was an IEEE/ACM Program Evaluator of the Computing Sciences Accreditation Board/Commission. Between 1995 and 2002, he has served as a member of the board of directors of the Society for Computer Simulation International. Between 2002 and 2004, he has served as Vice President of Conferences of the Society for Modeling and Simulation International SCS. Between 2004–2006, he served as Vice President of Membership of SCS. Between 2006–2009, he served as the Senior Vice President of SCS. He is currently the President of SCS. He has been invited to lecture and give keynote speeches worldwide. He was the recipient of the Best Paper Award in the IEEE AICCSA 2009 International Conference for one of his recent coauthored papers. In 2009, he was the recipient of the McLeod Founder's Award for Distinguished Service to the Profession. He is a Fellow of the Society for Modeling and Simulation International SCS.