A NEW ADAPTIVE AND INTELLIGENT TRAFFIC SHAPER FOR HIGH SPEED NETWORKS

A. A. Safavi, Ph.D.
Department of Electrical Engineering
School of Engineering, Shiraz University
Shiraz, I. R. of Iran
Corresponding Author: safavi@shirazu.ac.ir

I. Shames, M.S.
Department of Electrical Engineering
School of Engineering, Shiraz University
Shiraz, I. R. of Iran

N. Najmaei, M.S.
Department of Electrical Engineering
School of Engineering, Shiraz University
Shiraz, I. R. of Iran

M. Zamani, M.S.
Department of Electrical Engineering
School of Engineering, Shiraz University
Shiraz, I. R. of Iran

Abstract - In this paper, the advantage of reinforcement learning to develop a new traffic shaper is invoked in order to obtain a reasonable utilization of bandwidth while preventing traffic overload in other parts of the network. This leads to a reduction in the total number of packet droppings in the whole network. The method is implemented in a novel proposed intelligent simulation environment. Keeping dropping probability low while injecting as many packets as possible into the network, in order to utilize the available bandwidth, shows satisfactory behavior in simulation environment. On the other hand, the results show that the system can perform well even in situations that have not been previously introduced to the system.

Keyword: Network, Traffic Control, Traffic Shopping, Intelligent Systems, Reinforcement learning.

INTRODUCTION

Advances in high speed data transmission result in development of many new interactive and real-time applications such as teleconferencing, distributed computation, etc. Due to the heterogeneity of the sources of these new applications in terms of data types (i.e. The applications require video, audio and data transmission.), the 'best-effort' Quality of Service (QoS) of current packet-switched networks such as Internet does not provide good support for continuous transmission of data for these applications. Therefore, one needs new effective congestion control methods to meet QoS requirements. These include admission control, traffic enforcement and shaping, and scheduling schemes at the intermediate switches. In the past several years, the control community has tried to systematically model communication networks in order to thoroughly analyze their structure, properties and behaviors. This has led to the development of several algorithms for congestion control, for example.

In a resource sharing packet network, admission control and scheduling schemes are insufficient to provide QoS guarantees. This is due to the fact that the users may, inadvertently or otherwise, attempt to exceed the rates specified at the time of connection establishment [1]. The traffic policing schemes proposed in the literature include mainly Leaky Bucket (LB), Token Bucket (TB), Jumping Window (JW), Moving Window (MW), Exponential Weighted Moving Average (EWMA) and associated variations. A performance comparison among these schemes shows that the TB and the EWMA are the most promising mechanisms to cope with short-term fluctuations and hence are suited for policing burst traffic [1]. Several improvements of the LB have been proposed for increasing utilization in an ATM environment [2,3,4]. Traffic enforcement schemes supervise the source streams to check whether their characteristics conform to the declared values throughout the connection. Various schemes have been studied from the point of view of their capability to smooth the burstiness in the source. Traffic Shaping, on the other hand, conditions the input stream so that the characteristics are amenable to the scheduling mechanisms to provide the required QoS guarantees [1,11]. Although, one may imply the other, there are subtle differences. The former checks the conformance to the declared values whereas the latter shapes it to be more agreeable to the scheduling policies.

There have been previous attempts to achieve an adaptive traffic shaper [1,5], but they are highly dependent on traffic measurement. In previous work [12], the authors introduced an intelligent traffic shaper in which a typical Q-learning was used to perform the traffic shaping action. In that work, the traffic shaper was topologically dependent and thus unable to adapt itself with undesired bandwidth changes, etc. In this paper, a topologically independent adaptive and intelligent traffic shaper is proposed which learns the best parameters for a token bucket at any arbitrary node of the network, at any given time, no matter how the medium bandwidth changes. In addition, this new method uses no external tool to measure traffic parameters.

In the following part, the proposed system model is given. Then, theoretical framework of the learning agent is thoroughly discussed. In the fourth section, remarks are made on using the networks infrastructure in order to get feedback information without overloading the transmission lines with extra feedback packets and a reward evaluator is proposed for the learning of the agent. The proposed simulation framework is discussed thereafter. Simulation results and concluding remarks are given in the last two parts of the paper.

THE SYSTEM MODEL

In this proposed traffic policing scheme, routers are categorized into two main groups of Network Routers (NR) and Source Routers (SR) [12]. The SRs are those routers that are connected to end nodes and NRs are those that are indirectly connected to each end node and act as connections between subnets (see Figure 1).

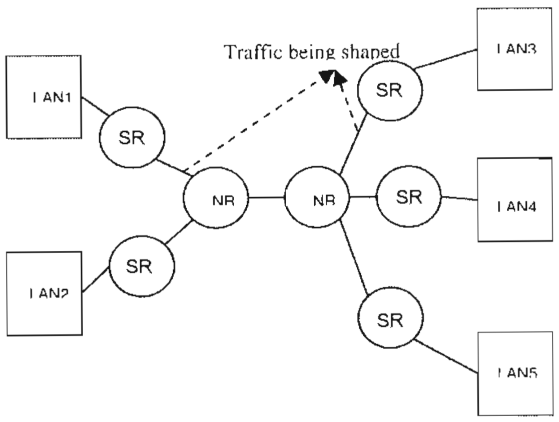


Figure 1: The network model under consideration.

Each SR is characterized with a token bucket and buffer length of l (bits) and token generation rate of g (bits/s). In this case, the source can only inject a complete data packet into the network if there are enough tokens for a complete packet transmission, and tokens are discarded if the token bucket overflows [1,5,12]. In case of bursty generation of packets, if the total number of bits is larger than the tokens, then only the first few packets whose lengths are smaller or equal to the total number of tokens can pass, and the others should wait for the time when enough tokens are generated for them. The NRs are responsible for generating information for the SRs which can adjust their parameters in line with the goal of minimization of the packet loss probability in the network. To achieve this goal, the parameters of the token bucket should be chosen in a way to make the loss as small as possible. Therefore, a flexible mechanism for choosing the token bucket parameters, g, is desirable [12]. To do so, an intelligent system is designed which learns the best g for the token bucket in each state of the network. In this scheme, the action, a, determines g of each SR, and can take a value between 0 and 100. The value of a is related to g as,

$$g = a \times g_{max} \tag{1}$$

where g_{max} is determined by

$$g_{max.i} = W_j \tag{2}$$

with Wj defined as the bandwidth of the medium connected to the jth port of the router in which the traffic is shaped.

Network states are determined by two parameters: packet dropping percentage sensed by ith SR at time t, namely $p_{t,i}$, and used buffer size to maximum buffer size ratio at the sending port of SR_i connecting to NR_m, namely $b_{t,im}$. The reward r_{t+1} is determined by how effective was the action, a, at time t in changing the state from a worse one to a better one. The reward takes a value between 0 to 1. This procedure is discussed in the following sections. Figure 2 depicts a block diagram of the proposed system.

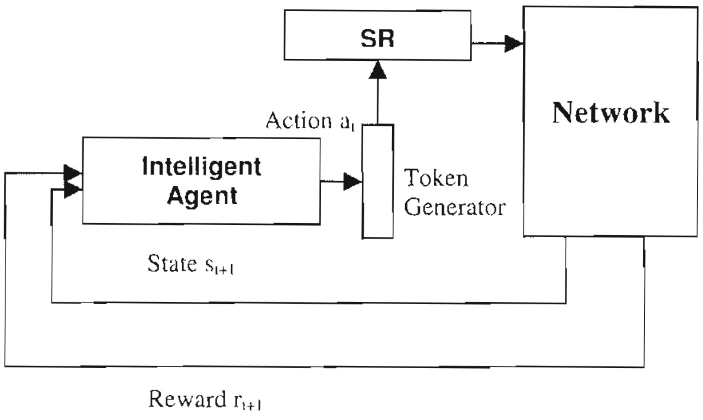


Figure 2: Block diagram of the proposed system.

THEORETICAL FRAMEWORK

Modem reinforcement learning research uses the formal framework of Markov decision processes (MDPs) [6]. In this framework, the agent and environment interact in a sequence of discrete time steps, $t = 0, 1, 2, 3, \ldots$. On each step, the agent perceives the environment to be in a state, s_t, and selects an action, $\mathbf{a}_t$. In response, the environment makes a stochastic transition to a new state, s_{t+1}, and emits a numerical reward, $r_{t+1} \in [0,1]$. The agent seeks to maximize the reward it receives in the long run. For example, the most common objective is to choose each action $\mathbf{a}_t$ so as to maximize the expected discounted return,

$$E\left(\sum_{t=0}^{\infty} \gamma^t r_t\right) \tag{3}$$

where γ is a discount-rate parameter, $0 \leq \gamma \leq 1$.

Reinforcement learning methods attempt to improve the agent's decision making policy over time. Formally, a policy is to map the states to actions or to probability distributions over actions. The policy is stored in a relatively explicit fashion so that appropriate responses can be generated quickly in response to unexpected states. The policy is thus what is sometimes called a "universal plan" in artificial intelligence, a "control law" in control theory, or a set of "stimulus-response associations" in psychology. The value of being in a state s under policy π can be defined as the expected discounted return starting in that state and following policy π. The function that maps all states to their values is called the "state-value function" for the policy π and denoted as V^{π},

$$V^{\pi}(s) = E_{\pi}\left(\sum_{t=0}^{\infty} \gamma^t r_{t+1} \mid s = s_t\right) \tag{4}$$

The values of states define a natural ordering of policies. Policy π is said to be better than or equal to policy π' if and only if $V^{\pi}(s) \geq$'3d $V^{\pi'}(s)$ for all states $\mathbf{s}$. For infinite MDPs, there are always one or more policies that are better than or equal to all others. These are the optimal policies, all of which share the same value function.

The simplest reinforcement learning algorithms apply directly to the agent's experience interacting with the environment and change the policy in real time. For example, Watkins'

Q-Learning algorithm [7,8], one of the simplest reinforcement learning algorithms, uses the experience of each state transition to update each element of a table. This table, denoted **Q**, has an entry **Q(s, a)** for each pair of state s and action **a**. Upon the transition $s_t \rightarrow s_{t+1}$, having taken action $\mathbf{a}_t$ and received reward r_{t+1}, this algorithm performs the update as below:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \left\lfloor r_{t+1} + \gamma \max Q(s_{t+1}, a_t) - Q(s_t, a_t) \right\rfloor \tag{5}$$

The algorithm described above is only the simplest reinforcement learning methods. More sophisticated methods implement **Q** not as a table, but as a trainable parameterized function such as an artificial neural network. This enables generalization between states which can greatly reduce learning time and memory requirements. In this proposed method, the latter type of Q-Learning is invoked due to the large size of action space [9]. Here, the action is set the value of ***a*** which can take a value between **0** and ***100*** and is calculated as described earlier in this paper. These actions are related to generation rate, ***g***, in each SR by Equation 1.

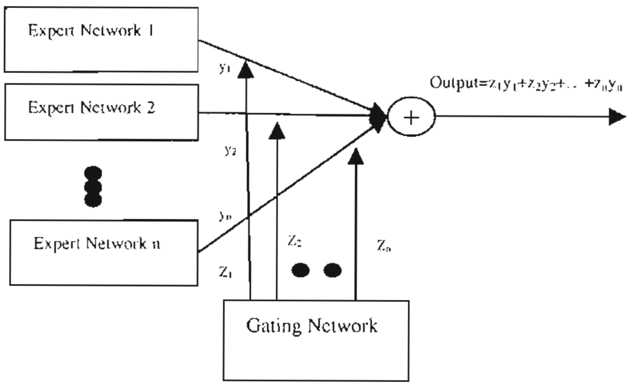


Figure 3: Architecture of expert networks and gating network (from [9]).

Figure 3 shows several expert and one gating networks in a modular architecture [9]. The expert network architectures are equivalent, and in fact can be used as stand-alone networks trained to learn the whole task. When used as part of a bigger modular network, each competes to learn a sub-task instead of learning the whole task. In Figure 3, y_i, $i = 1, 2, \ldots, n$ denotes the outputs of the expert networks. The gating network has the same number of output units as the number of expert networks. The variable z_i, $i = 1, 2, \ldots, n$ denotes the outputs of each output unit of the gating network. The values of z_i are positive and sum to one. The output of the entire network is determined by:

$$output = \sum_{i=0}^{n} z_i y_i \tag{6}$$

In this way, the gating network determines how much each expert network should contribute to the final output. The weights of a neural network being trained via Q-learning are modified so as to maximize $V^{\pi}(s)$, the discounted cumulative reinforcement, in Equation 3.

Given a state $\mathbf{s}_t$ and an action **a**, the system moves into a new state s_{t+1} and gets feedback reinforcement $\mathbf{r}_{t+1}$ from the environment. The Q-function can be learned by the following steps [9]:

1. Input vector is fed into the expert networks and the gating network.
2. The gating network selects **i**th expert network for that particular input state according to the output value.
3. Let **u** be the current value of **Q(s**$_t$**, a)** output by the **i**th expert network.
4. Let **u'** be the **i**th expert network's predicted value $r_{t+1} + \gamma \max_a Q(s_{t+1}, a_t)$.
5. Update the weights of the **i**th expert network to improve Q-function by back-propagating the temporal difference error $u'-u$.

The gating network is able to learn both a decomposition of the whole task and the control of each sub-task. This network takes the same input as what the expert networks take. The last layer of this network computes the weighted sums of the outputs of the hidden units. The weighted sum of the **j**th unit is denoted as $\mathbf{q}_j$:

$$q_j = \sum_{i=1}^{m} x_i w_{ji} \tag{7}$$

where $\boldsymbol{m}$ is the number of hidden units, and w_{ij} is the weight connecting output node $\boldsymbol{j}$ and hidden unit output $\boldsymbol{x}_i$. Because the outputs of gating network have to sum to one, the "softmax" activation function [9] is used in the second layer of the network to meet this constraint. The $\boldsymbol{i}$th output node is denoted as z_i:

$$z_i = \frac{e^{q_i}}{\sum_{j=1}^{n} e^{q_j}} \tag{8}$$

where $\boldsymbol{n}$ is the number of output units. During the training, the weights of the expert networks and the gating network are updated at the same time using the back-propagation with TD-error. This will lead to the maximization of the Q-function as time evolves.

STATES AND REWARD CALCULATIONS

To determine the current state of the system two parameters are used. The first is packet dropping percentage sensed by $\boldsymbol{i}$th SR at time $\boldsymbol{t}$, $\boldsymbol{p}_{t,i}$. The second is the used buffer size to maximize buffer size ratio at the sending port of SR$_i$, connecting to NR$_m$, $\boldsymbol{b}_{t,im}$. Calculation of the latter is rather straightforward but for the former some problems arise. First, how can one measure the parameter value? and second, after measurement, how can one transfer this feedback information to the place that it can be used? To answer the first problem, it is assumed that the network supports the Explicit Congestion Notification (ECN) mechanism [10] which has been proposed as a solution for conveying the congestion signaling rapidly and explicitly to TCP senders. This mechanism is utilized for estimating $\boldsymbol{p}_{t,i}$ at SR$_i$, because the ECN mechanism marks packets instead of dropping them as a means of signaling congestion. This mechanism marks packets as dropped with a probability proportional to network congestion.

Ideally, this feedback information should be conveyed from where the packet marking has happened to the SR where it is utilized. However, it is impossible to communicate directly with these routers without the aid of any additional signaling protocol, because current IP networks do not have any signaling architecture for this feedback information. Hence, one has to find a way to convey the information to the SR. The TCP ACK packet can serve as a good transporter for this purpose [11]. If TCP receivers receive a packet which is marked as dropped, they simply extract the flag from the IP header (Unused two-bit subfield in the IPv4 Type-Of-Service (TOS) field or IPv6 Traffic Class (TC) field) and copy them into the unused field in the TCP header of ACK packet in order to feed them back to the TCP senders. Because the feedback information consists of only two one-bit flags, this does not create a great deal of overhead. The SR$_i$ router checks the ACK packets and counts how many of them are marked and calculate $p_{t,i}$ at the end of each duty cycle of the network,

$$p_{t,i} = \frac{Number\quad of\quad marked\ \ packets}{Total\ \ number\quad of\quad pockets} \tag{9}$$

Having these two parameters and feeding them into a state detector, the current state of the network can be determined. These states are shown in Figure 4.

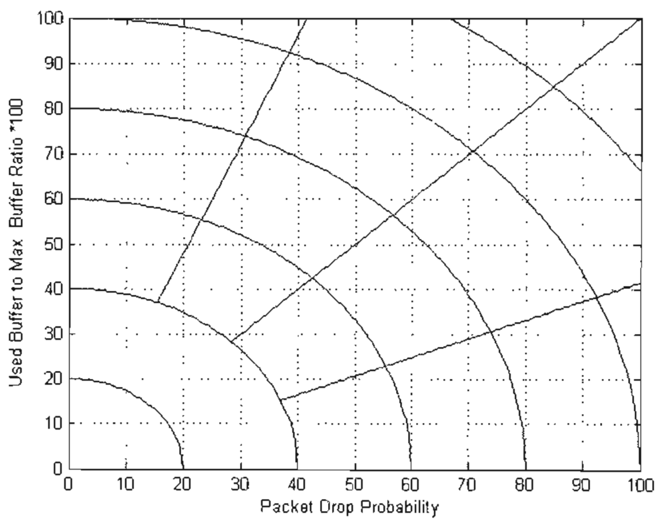


Figure 4: State plane of the network.

To calculate reward one may follow the simple rule

$$r_{t+1} = \begin{cases} \dfrac{d_t - d_{t+1}}{d_t} & d_{t+1} < d_t \\ 0 \quad d_{t+1} \le d_t \end{cases} \tag{10}$$

where d is calculated as below:

$$d_t = \sqrt{p_{t,i}^2 + b_{t,im}^2} \tag{11}$$

Here $p_{t,i}$, is dropping percentage that is sensed by SR_i in which the traffic is shaped at time t, and $b_{t,jm}$ is current buffer size to maximum buffer size ratio at ith Source Router which is connected to mth Network Router with the link jm.

SIMULATION FRAMEWORK

To achieve the goal of designing a precise model of the system capable of simulating real dynamic behavior of the real network systems, It is assumed that mediums, LANs, routers and packets are as entities that exist in large high speed networks. Each unit contains its own essential sub-units.

Each unit has its own responsibilities, behaviors, parameters, limitations, etc. This defines the nature of that unit. For example, routers contain different numbers of ports where each port has its own buffer size, and token generation unit with responsibilities such as routing, sending packets, receiving packets, etc. Taking into consideration the resource restrictions, it is tried to split each unit behaviors into separate modules and define them in functions which are as much independent as possible and do not encounter lack of essential resources. This independency of modules (with essential interactions among them) provides the user with the chance to change the specific behaviors of each unit only by redesigning the desired module and without the need to redesign the whole model. This feature helps simulate desired network under different circumstances such as different routing algorithms, different traffic shaping policies and other aspects. Besides, a great access to desired details of each unit and modules will become possible. On the other hand, new developments and inventions can be verified through a simple task of reconsideration in a small number of modules.

- SIMULATION METHODOLOGY

Different methods can be used in simulation of high speed networks. Discrete Event Simulation (both time driven and event driven simulations) can be proposed for this matter [14,15,16]. On the other hand, process emulation method may also be used, but it may inherit the non-suitable performance due to overhead associated with creating a large number of processes. Then communication among them becomes prohibitive as the model size increases [16].

The invoked method in this paper for simulating large high speed networks is an intelligent agent-based approach. This method will not encounter any prohibitions due to overhead associated with creating processes and communicating among them. Besides, it omits the need for a large number of events that are listed in event lists, which can become prohibitive in large scale systems that may have thousands or more numbers of events scheduled in the event-list.

Both Discrete Event Simulation and intelligent agent-based approach will be studied in the following.

- -Discrete Event Simulation

Typically, performance modeling involves the simulation of different system states which are represented by the presence or absence of countable units, such as jobs, requests, processes, packets, users or errors. A new state can only be entered through the execution of an event that modifies one or more of these units. Here, the state change involves a specific number of discrete events and, therefore, this type of simulation is generally referred to as Discrete Event Simulation [14,15].

This method can be implemented in two ways, time driven method and event driven method. In time driven method, one has a set of events ***E*** that contains all plausible events that can be executed in the current time. If ***E*** is null, it means there is no plausible event for the present time. In each time ***t*** an event ***e*** will be chosen from set of ***E*** and will be executed. Each event has some consequences that can be appeared as new events in the set of ***E***. In a large scale network, where thousands of packets are transmitting between their sources and destinations, set ***E*** can contain large number of events and it has to choose between them on a random basis. On the other hand, it requires complex programming of the system to enable it to know all of the future events and their desired time of execution.

In the event driven method, events are scheduled for the various future instances of time at which they will be executed. Each event ***e*** has a time of ***t(e)*** at which the event has to be executed. At the trigger of the clock, all of the events with the occurrence time equal to present will be executed and the execution of each event can result in generation of new events which will be scheduled as well and they will be placed in the *event list*. This method has the same problems of resource restrictions and complexity in the programming as the time driven method [14,15,16].

In the event driven method, the scheduling aspect system results in new consideration in designing of the system. Two methods of post-scheduling and pre-scheduling have been proposed for this matter and the consequence of both of them is the complexity of the program (for more information refer to [14,15,16]).

- - Intelligent Agent Based Method

In the used method here, each entity was designed as an intelligent unit with each essential behavior, responsibility, parameter, restriction, etc. As a consequence, each unit is completely capable of handling its tasks and needs. In a higher level, we designed a Network class for this system in a way that it has relationship with all of the entities within the system and it takes the responsibility of communicating between them. In another word, it plays the role of the spinal cord of the system.

There is no need for memory-consuming event lists and complex programming. Each unit has complete ability to render the processes and in case that it needs any information from the other entities (i.e Relationships that really exist in real network systems.) The information can be achieved through the Network class of the system. When different units generate packets, the responsibility of passing on these packets between units is on the network class but it does not mean that the network class does the routing task. All the decisions are taken by the units themselves and routers decide and determine the path of the packet. The units decide on what to do with the packet (depending on their defined responsibilities and policies). Then they hand the packets to the network class and direct it to hand packets over which of its neighbors. The next neighbor determines the next step for the packet and again delivers it to the network class to pass it on.

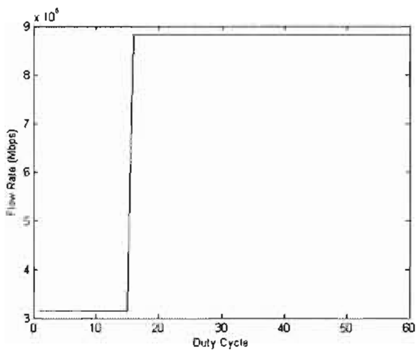
In this approach, one has avoided the memory-consuming lists of the other methods and limited the memory usage only to the memory that is used for saving the units information. Other tasks are all done by the intelligent and well-defined units. This specialized manner helps make fundamental changes in the behavior of the system (routing algorithms, traffic shaping policies and etc.) through simple changes in the specialized functions of each class of units.

SIMULATION RESULTS

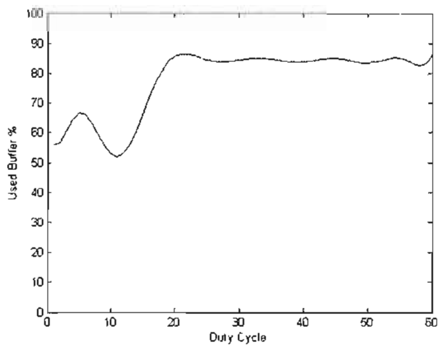
In this section some simulation results are provided. The simulation environment was briefly explained in the previous section and is also available from an earlier work [13]. To perform the simulations, an intelligent traffic shaper (Introduced in the third section) was added to the SR routers object. In the followings, simulation results from three different scenarios are presented.

- EXPERIMENT 1

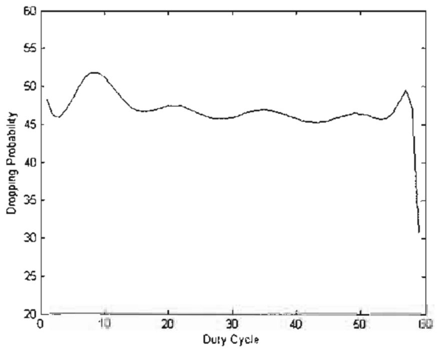
In this experiment, the performance of the method is studied in reaction to 3 types of burstiness that may happen in a network. At first, the system reaction to sudden rise in input flow rate of Source Routers is studied (i.e. These are connected directly to the end nodes.). In the second test, the system behavior to a flow rate increase-decrease is monitored. At last, the system reaction to a sudden decrease in input flow rate of Source Router is observed. In each case $p_{t,i}$ is measured and results are depicted in Figures 5 to 7. The results show that the system can keep dropping percentage low while it keeps the used buffer size as small as possible. The simulated network is presented in Figure 8.



(a)

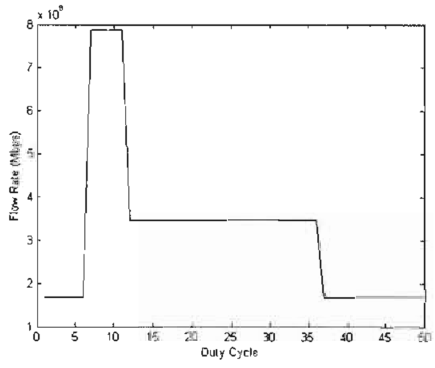


(b)

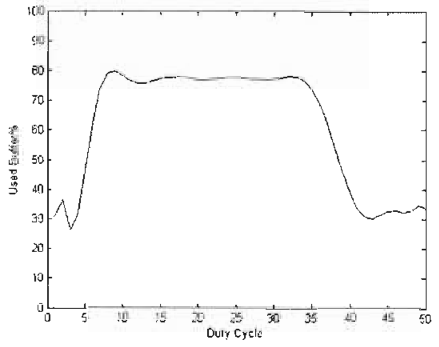


(c)

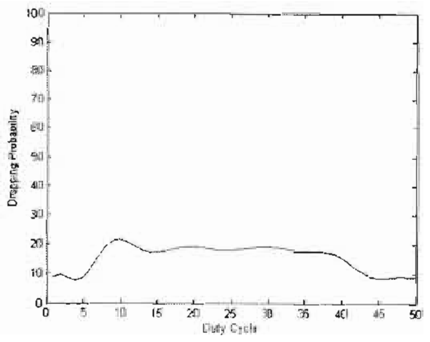
Figure 5: (a) Input Flow rate to SR_1, (b) Used Buffer of SR_1, (c) Dropping probability at NR_1.



(a)



(b)



(c)

Figure 6: (a) Input Flow rate to SR_1, (b) Used Buffer of SR_1, (c) Dropping probability at NR_1.

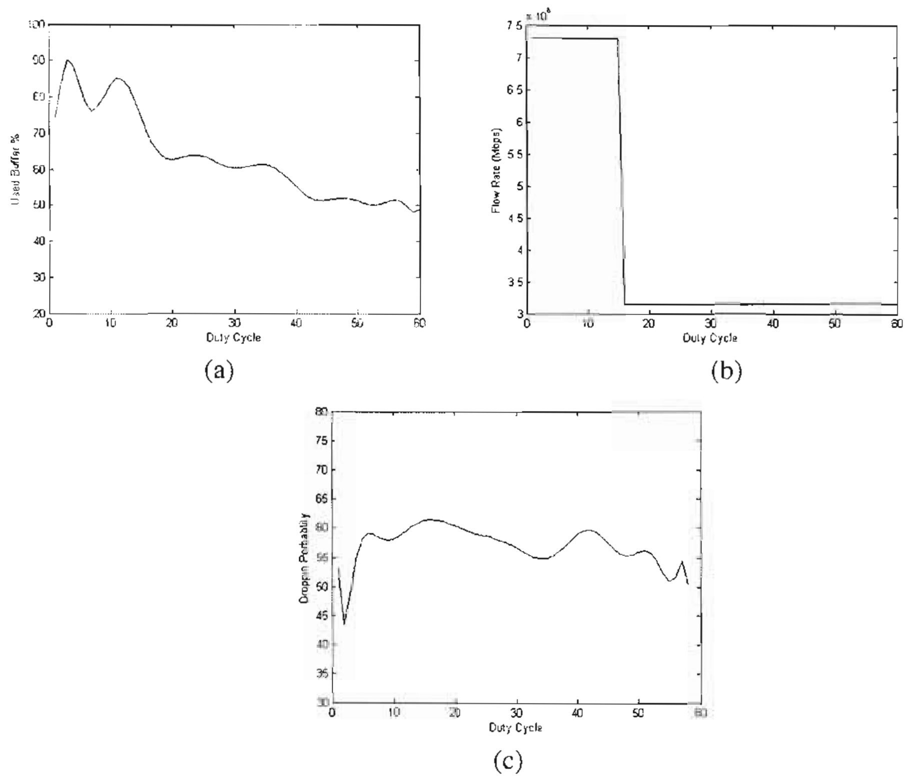


Figure 7: (a) Input Flow rate to SR₁, (b) Used Buffer of SR₁, (c) Dropping probability at NR₁.

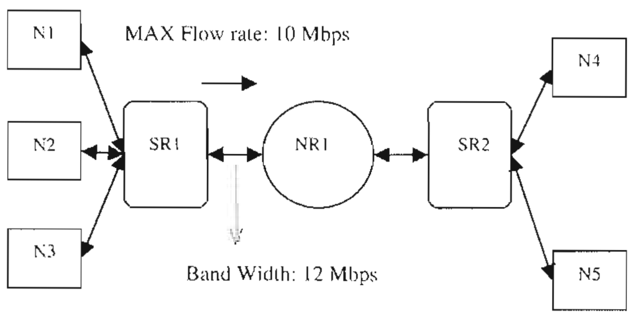
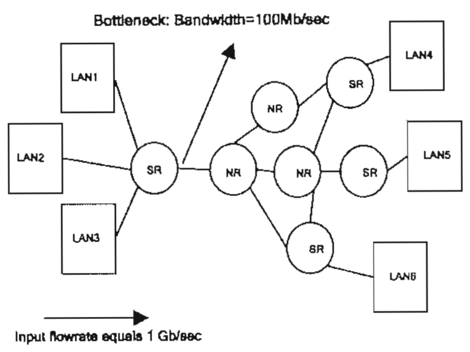


Figure 8: The simulated network that was used in experiment 1.

- EXPERIMENT 2

In the previous experiment and [12], it has been shown that the proposed traffic shaper works in a reasonable manner in those networks that has no bottle neck. Here, the performance of the traffic shaper is studied at the presence of a bottle neck in a network like the one which is depicted in Figure 9. The response to a random input flow rate is presented in Figure 10. As it can be seen from the figures, although bottleneck exists, the intelligent traffic shaper has managed to keep the discarding probability as low as possible, while utilizing available bandwidth in a reasonable fashion.


Bottleneck: Bandwidth=100Mb/sec
LAN1
LAN2
LAN3
LAN4
LAN5
LAN6
SR
NR
Input flowrate equals 1 Gb/sec
Figure 9: Network tested in experiment 2.

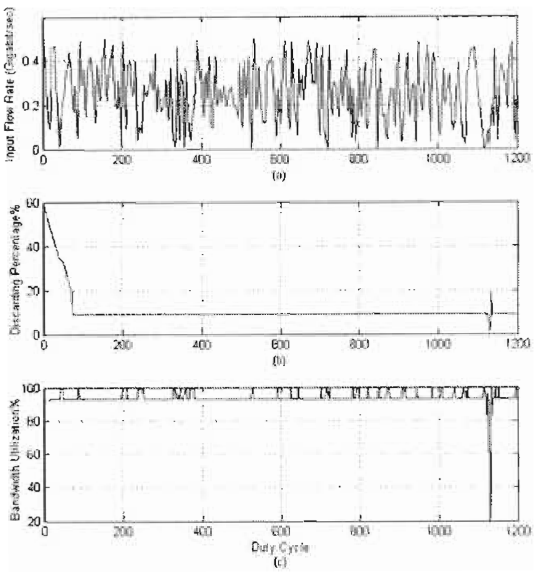


Figure 10: (a) Input Flowrate, (b) Discarding Probability Percentage, (c) Free Bandwidth Utilization.

- EXPERIMENT 3

In this experiment, the effect of a medium change on the performance of the traffic shaper is investigated. To do so, the intelligent shaper is trained for the environment depicted in Figure 11. Then, the medium indicated by arrow is replaced with a medium with maximum bandwidth of 100 Mb/sec. Discarding percentages in the network and available bandwidth utilization in the discussed medium to two different input flowrates are presented in Figures 12 and 13.

As it is clear from these figures, one can see that the traffic shaper shapes the traffic regardless of the difference of the test environment from its original training conditions.

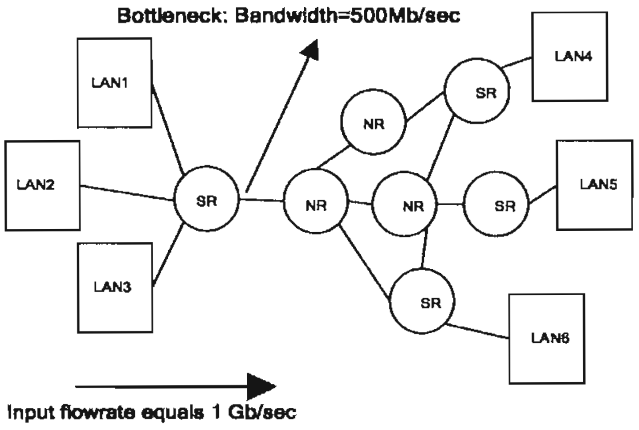

Bottleneck: Bandwidth=500Mb/sec
LAN1
LAN2
LAN3
LAN4
LAN5
LAN6
SR
NR
Input flowrate equals 1 Gb/sec

Figure 11: Network Used for Training in Experiment 3.

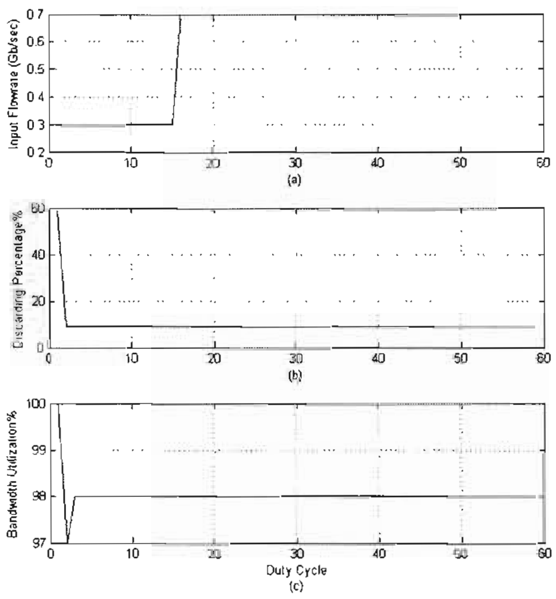


Figure 12: (a) Input Flowrate, (b) Discarding Probability Percentage, (c) Free Bandwidth Utilization.

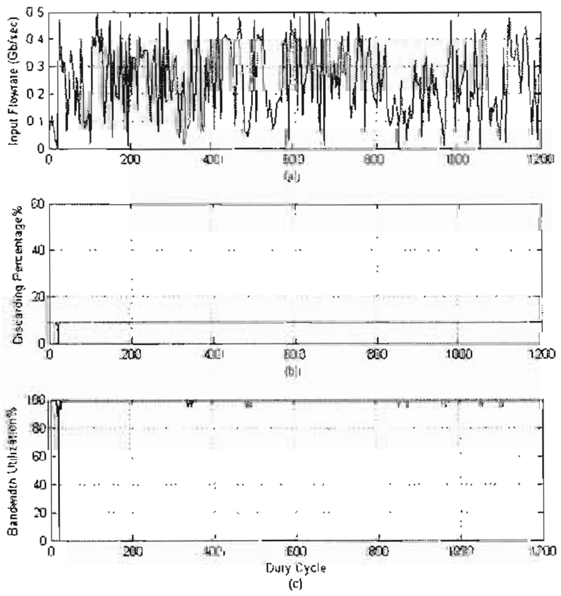


Figure 13: (a) Input Flowrate, (b) Discarding Probability Percentage,
(c) Free Bandwidth Utilization.

CONCLUSIONS

A new adaptive and intelligent traffic shaper was proposed in this paper. The aim was to obtain a reasonable utilization of bandwidth while preventing traffic overload in other parts of the network and as a result, reducing the total number of packet droppings in the whole network. The intelligent agent trains to learn an appropriate value for token generation rate of a Token Bucket at various states of the network. The results show that the system can keep the dropping percentage low while it keeps the used buffer size as small as possible. Besides, it has been shown that the proposed traffic shaper can perform well in those new scenarios that have not been previously introduced to. Implementing the method on a real network will be done in future studies.

REFERENCES

[1] Anderson, C. W. and Hong, Z., "Reinforcement Learning with Modular Neural Networks for Control." *Department of Computer Science Colorado State University*, Available: at <www.cs.colostate.edu/~anderson/pubs>.

[2] Bala, K., et al., "Congestion Control for High Speed Packet Switched Networks." *Proc. IEEE INFOCOM*, pp. 520-526, 1990.

[3] Cassandras C.G. *Discrete Event Systems – Modeling and Performance.* Richard D. Irwin Inc., Boston, 1993.

[4] Eckberg, A. E., et al., "Bandwidth Management: A Congestion Control Strategy for Broadband Packet Networks-Characterizing the Throughput-Burstiness Filter." *Comput. Networks ISDN Systems*, Vol. 20, pp. 415-423, 1990.

[5] Fujimoto R. M., *"Parallel Discrete Event Simulation." Communications of ACM*, Vol. 33, No. 10, 1990.

[6] Kaelbling, L. P., et al., "Reinforcement Learning: A Survey." *Journal of Artificial Intelligence Research*, Vol. 4, pp. 237-285, 1996.

[7] Mikler A. R., et al., *An Object-Oriented Approach to Simulating Large Communications Networks.* Iowa State University, 1998.

[8] Najmaei, N. el al., "An Object Oriented Approach to Simulation of High Speed Networks." *Technical Report, Dept. of Electrical Engineering*, Shiraz University, 2006.

[9] Park, E. C., and Choi, C. H., "Adaptive Token Bucket Algorithm for Fair Bandwidth Allocation in DiffServ Networks." *School of Electrical Engineering and Computer Science, Seoul National University, Seoul, KOREA*, csl.snu.ac.kr/publication.

[10] Radhakrishnan, S., et al., "Flexible Traffic Shaper for High Speed Networks: Design and Comparative Study with Leaky Bucket." *Computer Networks and ISDN Systems,* Vol. 28, pp. 453-469, 1996.

[11] Ramakrishnan, K. and Floyd, S., "A Proposal to Add Explicit Congestion Notification (ECN) to IP." *RFC 2481*, 1999.

[12] Shames, I., et al., "A New Intelligent Traffic Shaper for High Speed Networks." *Proc. ICMLA'06*, Orlando, Florida, December 2006.

[13] Sidi, M., et al., "Congestion Control through Input Rate Regulation." *Proc. GLOBECOM'89*, Dallas, Texas, pp. 1764-1768, November 1989.

[14] Tan, T. Y., et al., "Adaptive Resource Negotiation Based Control for Real Time Applications." *Computer Communications,* Vol. 24, pp. 1283-1298, 2001.

[15] Watkins, C., "Learning with Delayed Rewards." *Ph.D. Thesis, Cambridge University Psychology Department*, 1989.

[16] Watkins, C. J. C. H., and Dayan, P., "Q-learning." *Machine Learning*, Vol. 8, pp. 279-292, 1992.