Deep Blue: A Fuzzy Q-Learning Enhanced Active Queue Management Scheme

S. S. Masoumzadeh and G. Taghizadeh
Department of Computer Science
Qazvin Azad University
Qazvin, Iran
{masoumzadeh, gelareh.taghizadeh}@gmail.com

K. Meshgi and S. Shiry
Department of Computer Science
Amir Kabir University
Tehran, Iran
{meshgi, shiry}@aut.ac.ir

Abstract — **Although RED has been widely used with TCP, however it has several known drawbacks [1]. The BLUE algorithm that benefits from a different structure has tried to compensate some of them in a successful way [2]. A quick review on active queue management algorithms from the very beginning indicates that most of them tried to improve classic algorithms. Some of them use network traffic history to achieve more flexibility and prediction ability while others use algorithms such as fuzzy logic to address scalability problem and high input load. Our proposed approach benefits from both: Using fuzzy logic to deal with high input load and embedding expert knowledge into the algorithm while optimizing router decisions with reinforcement learning fed by network traffic history. We call this approach "DEEP BLUE" as is consist of an improved version of BLUE algorithm. Derived from BLUE, our algorithm uses packet drop rate and link idle events to manage congestion. Our experiments using OPNET simulator shows that this scheme works faster and more efficient than original BLUE.**

Keywords-component; Fuzzy Reinforcement Learning; Active Queue Management; OPNET Simulation

I. INTRODUCTION

Nowadays, congestion control is still an important challenge despite of quick breakthrough in network sciences. Many algorithms have been developed to address this issue, and some of them made it into real world application but still packet drop rate can be drastically uncontrollable in some situations. IETF suggests using explicit congestion notification along with one of active queue management (AQM) schemes such as RED [2]. Even though RED is widely accepted AQM schemes but it suffers from few defects which make it less responsive to the emerging need of a robust and efficient AQM algorithm, Researchers proposed several algorithms to cover up these drawbacks [1]. One of those algorithms is BLUE algorithm which benefits from different structure than RED and aims to have the advantages of RED while covering its defects. In its turn, Blue suffers from inaccurate parameter which decreases the performance of algorithm.. In this paper we propose a modification to BLUE which is called DEEP BLUE. This algorithm inherits the advantages of BLUE using its infrastructure to handle congestion. DEEP BLUE doesn't need the parameter *freeze_time* and uses learning techniques to determine step sizes. DEEP BLUE is provided with fuzzy logic and machine learning utilities to improve BLUE even more and forms a novel algorithm for active queue management.

II. RELATED WORK

A survey on recent researches, those which performed after the appearance of classic schemes [1] like RED, BLUE, and PI Controller, shows the bias of some researchers toward different creative structures. Some other researchers try to use basic structures of classic algorithms and develop intelligent approaches by adding the essence of artificial intelligence to them. Generally these researches can be divided into two major categories: The first category is the intelligent algorithms. These algorithms have different structure comparing classic ones and utilized with creative structure. Author of [3] proposed an intelligent algorithm and believes that an AQM algorithm should be self-adaptive, means that it should adjust its own parameters and maintain its efficiency at the best level. The proposed algorithm in this paper doesn't follow any classic algorithm and balances the throughput versus queuing delay trade-off with an AQM algorithm based on fuzzy logic. The scheme presented in [4] is claimed that is intelligent despite of previously fuzzy-based AQM schemes that extends RED algorithm because it designed a modern intelligent packet drop mechanism with fuzzy logic. A congestion detector is explained in [5] which not only inherits the advantages of classic schemes, but also use fuzzy logic with dynamic membership functions adjusted by PSO. Using Neuro-fuzzy to handle traffic swings and correctly detecting congestions is a modern approach which is stated in [6]. The second

category tries to improve traditional algorithms like RED, PI, etc. using intelligent tools and usually don't modify the structure. The idea beneath the fuzzy controller for RED which is explained in [7] is that linguistic knowledge is capable of implementing non-linear probabilistic drop unit. Having a better understanding of the environment, this method provides better quality of service for various traffics. In agreement with this idea, [8] highlights the effects of fuzzy logic on improving packet drop policy and proposed an algorithm called DSRED on the basis of fuzzy rules. In addition [9] use a fuzzy controller and [10] use a neuron controller to adjust maximum drop probability of RED, and both intend to maintain the average queue length near its size, i.e. to use full queue capacity in average case. Satisfactory queuing delay in RED is guaranteed in [11] by the means of learning automata. PID Controller is another classic AQM scheme which is subjected to extension in [12]. In this paper, PSO algorithm is used to adjust PID controller parameters. Similar idea is used in [13] in which these parameters are adjusted autonomously by RBF neural networks. In these two papers parameters are adjusted regarding link capacity, traffic load and transmission delay. This autonomy makes the mentioned algorithms counted as adaptive ones.

III. BLUE AQM SCHEME

Instead of using average queue length or even current queue length, BLUE directly uses the drop rate and link utilization to manage the queue. It keeps a single probability P_m to mark incoming packets. In the case of repeated drop of arrived packets due to buffer overflow, BLUE increases the P_m which can be considered as an explicit congestion notification. In contrary when the queue goes empty or become idle, this probability decreases. This mechanism enables BLUE to correctly learn the parameter necessary for congestion notification. BLUE uses three other parameters along with drop probability; The *freeze_time* parameter defines the minimum delay between two consecutive similar updates of P_m. It assures that system notifies the changes in mark probability before its next change. The second parameter is d_1 which determines the amount of change of P_m in case of overflow. Finally the parameter d_2 shows amount of decrease in P_m when the link goes idle [2]. Following code describes the pseudo-code of BLUE algorithm.

Upon packet loss event:
if ((now – last_update) > freeze_time)
$P_m = P_m + d_1$
last_update= now

Upon link idle event:
if ((now – last_update) > freeze_time)
$P_m = P_m - d_2$
last_update = now

IV. MOTIVATION

freeze_time parameter is one of the most important parameters that its different values lead to completely different results. The step size parameters d_1 and d_2 help BLUE to achieve the optimal Pm. Large steps boosts the speed of BLUE in convergence to the optimal value, however it may cause large jumps around this value. In the best case, when desired Pm has too much distance from current Pm, large steps take BLUE to the goal neighborhood, and if followed by small steps, optimal Pm can be achieved quickly. Learning traffic situations is one of the key elements in selecting d_1 and d_2 and drives algorithm to reach desired Pm very quickly

V. FUZZY Q-LEARNING

Reinforcement learning is the problem faced by an agent that has to learn behavior through trial and error interactions with a dynamic environment, Further there is a focus on on-line performance, which involves finding a balance between exploration (of uncharted territory) and exploitation (of current knowledge). Thus, reinforcement learning is particularly well suited to problems which include a long-term versus short-term reward trade-off. It has been applied successfully to various problems, including robot control, elevator scheduling, telecommunications and chess [14]. One of the most effective algorithms of this type is Q-Learning. Q-learning is a reinforcement learning technique that works by learning an action-value function that gives the expected utility of taking a given action in a given state and following a fixed policy thereafter. Q-learning is able to compare the expected utility of the available actions without requiring a model of the environment. The core of the algorithm is a simple value iteration update. For each state, *s*, from the state set *S*, and for each action, *a*, from the action set *A*, we can calculate an update to its expected discounted reward with the following expression:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(s_t, a_t) \times [r_{t+1} + \gamma \times max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (1)$$

Where r_t is an observed real reward at time t, $\alpha_t(s,a)$ are the learning rates such that $0 \leq \alpha_t(s,a) \leq 1$, and γ is the discount factor such that $0 \leq \gamma < 1$. The problem presented in this paper, will be modeled in the form of a Q-Learning optimization problem. Each QL problem consists of four design parts: Input States, Actions, Goal (or Goals), and Reinforce Signal. Input states and Actions will be described in full details in the implementation section. Fuzzy Reinforcement Learning is based on Fuzzy Inference Systems (FIS). FIS are universal approximators and they can learn by examples. The most important feature of FIS is that it can incorporate human priori knowledge into its parameters. As these parameters have clear physical meaning this will actually speedup learning. A FIS is based on a rule-base, in which each rule get an antecedent part and its corresponding

consequent. Usually the antecedent combination is not matter in tuning FIS, while the main argue is on the choice of the different linguistic terms for each fuzzy variable and the conclusion of each rule. With the method used here it is possible to tune the conclusion part of each rule in a TS-FIS which would be done over the whole possible rules in the rule-base. FQL is the fuzzy extension of Q-learning, which is an online model free optimization of a control policy. FQL uses TS-FIS to estimate the Q-value function for current state-action pair. This Q-value function along with the optimal Q-value of the state, which is calculated in the same way, will be used to compute the TD Error. Then based on the TD Error and update rule of the TD learning, the action weights will be updated towards gaining more reinforcement, as the case in Q-Learning. FQL then chooses actions based on the quality values (weights) of different actions available in the action set of each rule along with an exploration/exploitation mechanism named double e-Greedy [15-16]. As shown in the schematic diagram of Fig.1, the algorithm relies on the actions' quality values and fuzzy inference.

VI. IMPLEMENTATION

In this paper we improve BLUE by combining it with fuzzy Q-learning. We call the new algorithm DEEP BLUE. In the original BLUE algorithm, *freeze_time*, d_1, and d_2 parameters is used in each event while in the DEEP BLUE the *freeze_time* parameter no longer exists and d_1 and d_2 actions are replaced by d_{11} to d_{1n} and d_{21} to d_{2n}. Following steps is the detailed implementation of DEEP BLUE:

1- Initialize learning parameters γ and exploration rate θ

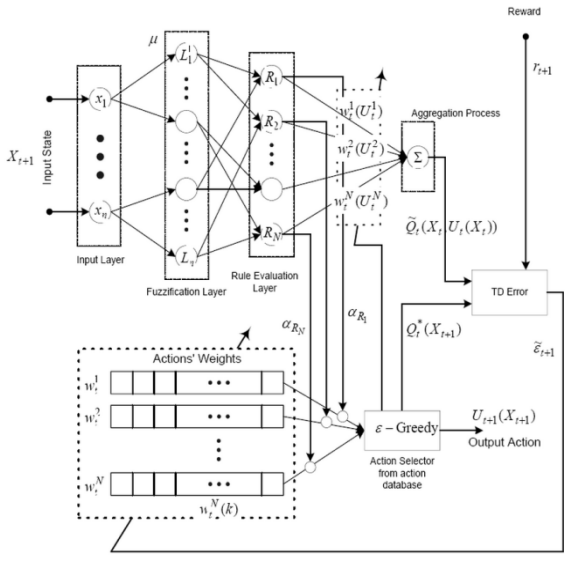


Figure 1. FQL STRUCTURE

2- Data Gathering for FQL. Construct input vector containing now-last update, current queue length, and packet drop probability (Pm) which defines current state of learning agent.

3- Fuzzification: we apply a triangular membership functions to fuzzify each input vector into low, medium and high.

4- Reinforcement Signal Generation: Reinforcement signal is a linear combination of throughput of the router and instantaneous queuing delay. These factors are inversely correlated thus the learning mechanism tends to balance their trade-off in dynamic traffic conditions in the network.

5- *Estimation of the Optimal Q-value:* The current input values fires a set of rules. The activated rules have a truth value which is calculated by Rule Evaluation. The optimal Q-value is the maximum value resulted from truth value of each rule multiplied by its best action value in the whole rule base.

$$Q^*(X_t) = \frac{\sum_{R_i \in A(X_t)} \alpha_{R_i}(X_t) \times \left[\max_{a \in U^i} w_t^i(a) \right]}{\sum_{R_i \in A(X_t)} \alpha_{R_i}(X_t)} \quad (2)$$

In this equation R_i represents a rule from rule-base, α_{Ri} is truth value of this rule, $A(X_t)$ is the set of activated rules with input X_t, and w_t is the table of Q-Learning in time *t.* These table stores values of state-action pairs corresponding to state *i* and actions listed in U_i set.

6- *TD Error Calculation:* In order to calculate the transfer probability of each state-action to a specific state, TD (0) error is calculated as:

$$\tilde{\varepsilon}_{t+1} = r_{t+1} + \gamma \times Q_t^*(X_{t+1}) - \tilde{Q}_t\big(X, U_t(X_t)\big) \quad (3)$$

Where r_{t+1} is the generated reinforce signal and γ ($0 < \gamma \leq 1$) is the discount factor.

7- *Exploration/Exploitation:* One of the most important issues of reinforcement learning is to balance exploration versus exploitation to find best action while keeping the good performance. We use following formula for action selection:

$$EE(a) = w_t^i(a) + \frac{\theta}{e^{n_t(a)}} \quad (4)$$

in which θ is a positive coefficient for the direct exploration part, $w_t^i(a)$ is action's weight, and $n_t(a)$ is the total number of times that the action has been used till time step *t*. *EE* becomes maximum for the actions that have low weights but are also so scarce. Each action should satisfy some constraints to be applicable such as range checks and distance between thresholds. The control mechanism embedded into our algorithm marks inapplicable actions with setting that they can never be chosen.

8- *Local e-Greedy:* Local action selection in each rule, will be done by a kind of e-Greedy strategy that selects all rules' candidates according to their EE values. Equation 5 formulizes this selection.

$$U_t^i = U^i(k) \quad | \quad EE\left(U^i(k)\right) = \max_{a \in U^i} EE(a) \tag{5}$$

9- *Updating FIS:* This task is taken over by tuning each action's weights according to equation 6:

$$w_{t+1}^i\left(U_t^i\right) = w_t^i\left(U_t^i\right) + \tilde{\varepsilon}_{t+1} \times \alpha_{R_i} \ , \forall R_i \in A(X_t) \tag{6}$$

10- *Double e-Greedy action selection:* After selecting all rules' candidates, in the next higher competition layer the total double e-Greedy or maximum e-Greedy action will be determined by a pure greedy approach based on the rules' truth values and the nominated actions' *EE* from previous steps. This procedure is described in equation 7.

$$U_t(X_t) = U_t^{i^*} \quad | \quad EE\left(U_t^{i^*}\right) \times \alpha_{R_i^*}(X_t) = \max_{R_i \in A(X_t)} \left(EE\left(U_t^i\right) \times \alpha_{R_i}(X_t)\right) \tag{7}$$

11- *Estimation of Current Q-value:* Finally, this phase tends to compute and memorize Q-value of the current state-action pair based on the new Q-value function after tuning action's weight parameters using equation 8.

$$\tilde{Q}_t\left(X, U_t(X_t)\right) = \frac{\sum_{R_i \in A(X_t)} \alpha_{R_i}(X_t) \times w_t^i(U_t^i)}{\sum_{R_i \in A(X_t)} \alpha_{R_i}(X_t)} \tag{8}$$

12- Actuation: In this version of DEEP BLUE, the action set for FQL consists of d_{ij} (i=1, 2, j=1, ..., n) and one "no-operation" action. The values of d_{ij} are defined empirically as: d_{11}=0.000005, d_{12} = 0.0005, d_{13}= 0.005 and d_{21}= 0.0000005, d_{22} = 0.00005, d_{23}=0.0005. Eventually the final DEEP BLUE algorithm is shown in following pseudo code.

For each packet arrival:

- ***Gather information from environment (include now - last update, current P_m and current Queue-size).***
- ***Fuzzify the inputs and create rule base.***
- ***Explore actions to find potential best actions.***
- ***Find best action due to learnt traffic model.***
- ***Update Q-learning tables.***
- ***Commit chosen action (include d_{mn} , m=1, 2 and n=1,2,3 for this implementation).***
- ***$P_m = P_m + d_{mn}$***

VII. RESULTS

In this section, we discuss network configuration and also the simulated model in Opnet. Fig.2 shows Opnet network model forming a simple bottleneck configuration. Each subnet consists of several TCP sources (e.g. 80 sources per subnet) which are based on TCP-Reno.

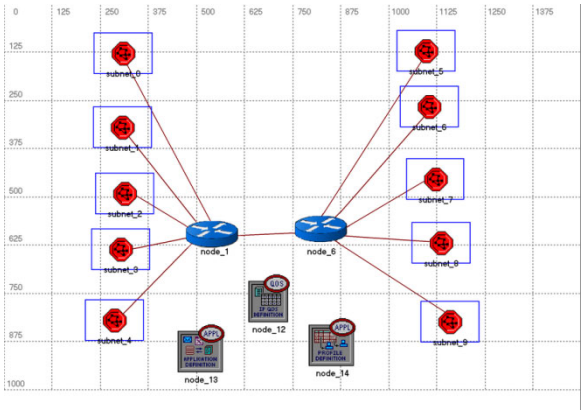


Figure 2. OPNET NETWORK MODEL – BOTTLENECK CONFIGURATION

In the simulation Table I shows BLUE parameters .Fig.3 compares changes of P_m in BLUE and DEEP BLUE. It's clear from that DEEP BLUE reacts quicker to the traffic changes.

TABLE I. Values of BLUE Parameters

Parameter	P_m	*freeze_time*	d_1	d_2
Value	0.35	1ms	0.00005	0.000005

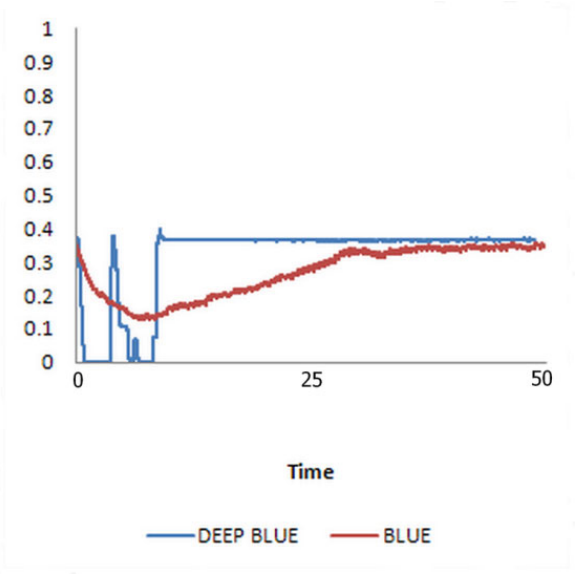
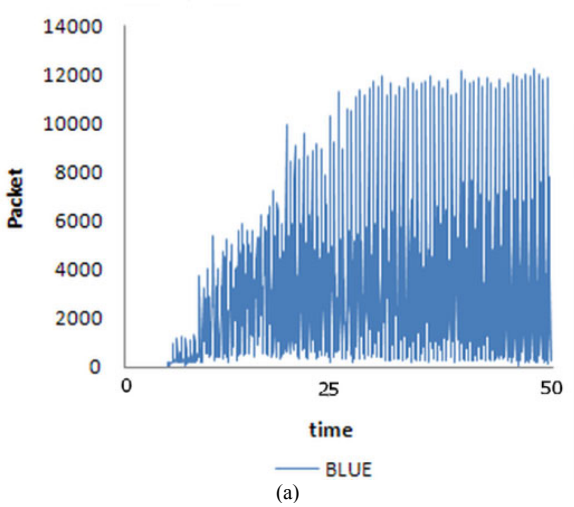
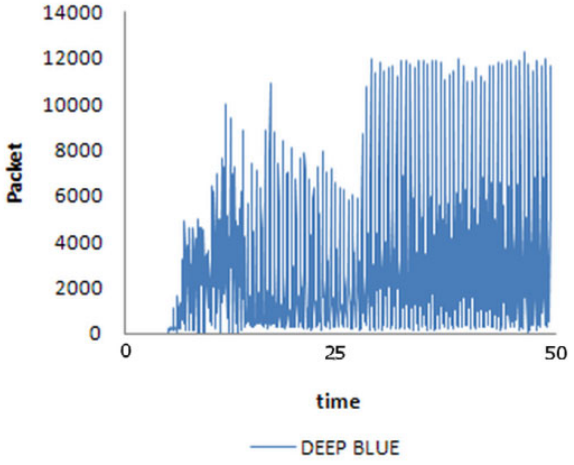


Figure 3. COMPARISON OF P_M BETWEEN BLUE AND DEEP BLUE

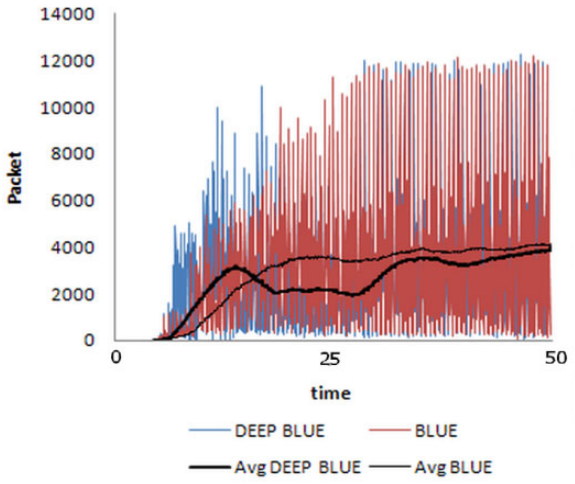
Packet drop and queuing delay are compared for two algorithms in Fig.4 and Fig.5 respectively. Although DEEP BLUE takes random actions early in the simulation, but after a while with the high input data volume, it learns traffic shape and conditions and better optimizes drop policy. DEEP BLUE has better speed of convergence to optimal P_m because of the adaptive step size, i.e. taking large step to reach near the optimal P_m and then finds it with smaller step size.



(a)

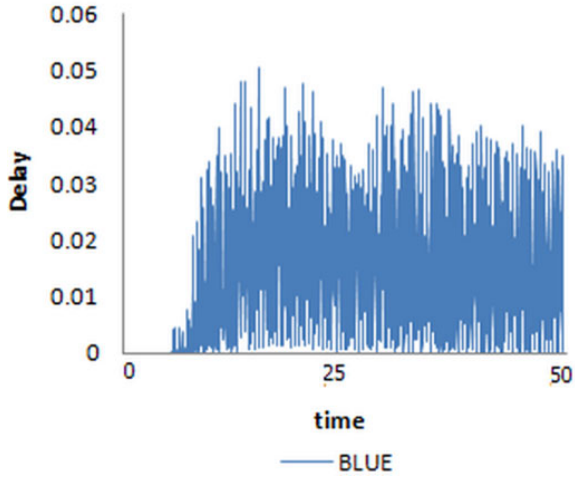


(b)

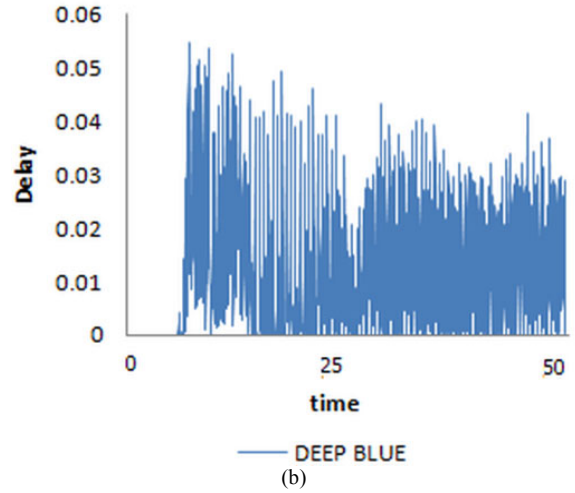


(c)

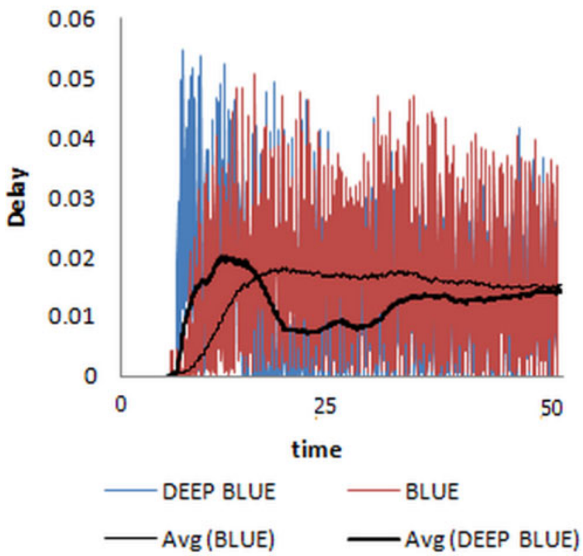
Figure 4. (A) PACKET LOSS IN BLUE (B) PACKET LOSS IN DEEP BLUE AND (C) COMPARISON OF PACKET LOSS BETWEEN BLUE AND DEEP BLUE



(a)



(b)



(c)

Figure 5. (A) QUEUING DELAY IN BLUE (B) QUEUING DELAY IN DEEP BLUE AND (C) COMPARISON OF QUEUING DELAY BETWEEN BLUE AND DEEP BLUE

VIII. CONCLUSION

BLUE algorithm can compensate some defects of RED algorithms; however it is highly dependent to its free parameters. DEEP BLUE could eliminate *freeze_time* parameter and applied multiple variables instead of d_1 and d_2 which selects them regarding the situation. This improves the speed and accuracy of the modified algorithm. Furthermore the independence of preset constants gives this algorithm a good flexibility. DEEP BLUE doesn't consume much memory because of using FIS and its computational complexity is not a deal as we monitor the CPU usage of router. It shows a little surplus over BLUE. Thus approach can be considered as a great applicable module for existing routers meeting their memory and computation limits.

REFERENCES

[1] S. Dijkstra, "Modeling Active Queue Management algorithms using Stochastic Petri Nets," in *Faculty of Electrical Engineering, Mathematics and Computer Science*. Vol. Master: University of Twente, 2004, p. 79.

[2] W. Feng, K. G. Shin, D. D. Kandlur, and D. Saha, "The BLUE active queue management algorithms," *IEEE/ACM Transactions on Networking (TON),* vol. 10, pp. 513-528, 2002.

[3] Y. Hadjadj Aoul, A. Mehaoua, and C. Skianis, " A fuzzy logic-based AQM for real-time traffic over internet," *Computer Networks,* vol. 51, pp. 4617-4633, 2007.

[4] F. Yanfei, R. Fengyuan, and L. Chuang, "Design of an active queue management algorithm based fuzzy logic decision," in *International Conference on Communication Technology Proceedings* 2003, pp. 286-289.

[5] C. N. Nyirenda and D. S. Dawoud, "Multi-objective Particle Swarm Optimization for Fuzzy Logic Based Active Queue Management," in *IEEE International Conference on Fuzzy Systems* 2006, pp. 2231-2238.

[6] M. F. Zhani, H. Elbiaze, and F. Kamoun, "؟_SNFAQM: An Active Queue Management Mechanism Using Neurofuzzy Prediction," in *12th IEEE Symposium on Computers and Communications*, 2007, pp. 381-386.

[7] C. Chrysostomou, A. Pitsillides, L. Rossides, and A. Sekercioglu, "Fuzzy logic controlled RED: congestion control in TCP/IP differentiated services networks," *Control Engineering Practice,* vol. 8, pp. 79-92, 2003.

[8] S. Subasree, "Fuzzy DS RED An Intelligent Active Queue Management Scheme for TCP/ IP Diff-Serv," in *International Conference on Computational Intelligence* Istanbul, Turkey, 2004.

[9] J. Sun, M. Zukerman, and M. Palaniswami, "Stabilizing RED using a Fuzzy Controller," in *IEEE International Conference on Communications*, 2007, pp. 266-271.

[10] J. Sun and M. Zukerman, "Improving RED by a Neuron Controller," in *Managing Traffic Performance in Converged Networks*. vol. 4516: Springer Berlin / Heidelberg, 2007, p. 434.

[11] M.Jahanshahi and M. R. Meybodi, "An Adaptive Congestion Control Method for Guaranteeing Queuing Delay in RED-Based Queue Using Learning Automata," in *International Conference on Mechatronics and Automation*, 2007, pp. 3360-3365.

[12] X. Wang, Y. Wang, H. Zhou, and X. Huai,"PSO-PID: a novel controller for AQM routers,"in *International Conference on Wireless and Optical Communications Networks*, 2006, p. 5.

[13] W. Jun-song, G. Zhi-wei, and S. Yan-tai, "RBF-PID Based Adaptive Active Queue Management Algorithm for TCP Network," in *IEEE International Conference on Control and Automation*, 2007, pp. 171-176.

[14] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*: MIT press, 1998.

[15] H. R. Berenji, "Refinement of Approximate Reasoning-based Controllers by Reinforcement Learning," in *Proceeding of the 8th International Workshop Machine Learning*, 1990, pp. 475-479.

[16] L. Jouffe, "Fuzzy Inference System Learning by Reinforcement Method," *in IEEE Transaction on System, Man, and Cybernetics.*vol.28, pp.338-355, 1998