Neural Network AQM Congestion Control Based Genetic Algorithm for TCP/IP Networks

Mohammad Rasoul Tanhatalab*, Modjtaba Rouhani **, Ali Shokouhi Rostami ***

*, **Azad University/Electrical Department, Gonabad, Iran,
*** Azad University/Electrical Department, Behshahr, Iran
*m_r_tanha@yahoo.com, **m.rouhani@ieee.org, ***soheil.shokohi@gmail.com

Abstract

Active Queue Management (AQM) has been widely used for congestion avoidance in TCP networks. Although numerous AQM schemes have been proposed to regulate a queue size close to a reference level as Random early Detection (RED), PI controller, PID controller *(Hollot C., 2002)*, adaptive prediction controller (APC) *(Torkamandi M., 2007)*, and neural network using the Back-Propagation *(Cho Hyun C., 2005)*. Most of them are incapable of adequately adapting to TCP network dynamics due to TCP's non-linearity and time-varying stochastic properties. This subject should be mentioned that neural network with Back-Propagation *(Cho Hyun C., 2005)*, has taken the wrong scheme. In this article is referred to the wrong. Also in this article, unlike previous methods we have used non-linear model of TCP network. We evaluate the performances of the proposed neural network AQM approach using MATLAB and simulation experiments. The proposed approach yields superior performance with faster transient time, larger throughput, and higher link utilization compared to another schemes.

1 Introduction

Considering the increase in use of computer data networks, voice and video, there has been extensive investment in high-quality service and traffic engineering, Therefore the use of algorithms is necessary to avoid congestion to optimum use of network facilities (bandwidth, processing speed of processors, memory of capacity routing and etc.) (*Tanenbaum, 2003)*. Congestion is one of the most important issues in which, the computer networks designers are involved. When the network faces congestion, data packets are destroyed, bandwidth is wasted, unreasonable delays occur. Algorithms which are used in the network router section are known as AQM: Tail-Drop and RED algorithm that has been recognized as the most important queue management algorithm *(Hollot C. , 2002)*. In addition to these two cases, SRED algorithm and BlueRED algorithm have been introduced to improve RED algorithm *(Cho Hyun C., 2005)*. To improve active queue management algorithms in recent years, classical control methods such as PI and PID controller are also used. In design of this controller, APC *(Torkamandi M., 2007)* has been used. In reference *(Cho Hyun C., 2005)*, a neural network trained for the back propagation error is used for AQM controller design .In this article activation function of hidden layer was linear, that based on BP training is wrong. In BP training, the activation function in hidden layer should be as sigmoid. Therefore the scheme in this reference is a wrong way to solve this problem.

In this article we scrutinize controller design method of neural network trained with genetic algorithm and by analyzing and comparing the results with the ones of algorithms RED, PI, PID and APC. We come to this conclusion that the results are significantly improved. In second section the nonlinear model of congestion in TCP networks is given and in the third section to introduce of different algorithms reposed for AQM.

In fourth section we would like to design a neural network controller with genetic training for AQM problem. In fifth section by doing different simulations and comparing various existing methods with presented method, the efficiency of the new method is studied. And in the final paper conclusion, the results are given.

2 TCP Networks congestion review and introduction of the athematical model of the network

Congestion occurs when the number of packets passing a part of the network is more than usual, in this case the network performance decreases. Figure 1 shows the network performance in congestion mode compared to optimum one *(Tanenbaum, 2003)*.

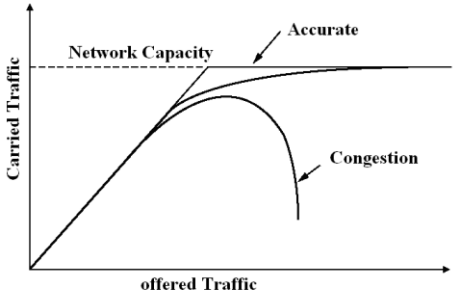


Figure 1: Network performance in congestion mode

Although 90 percent of the traffic data is caused by resources which use the TCP protocol in transfer layer, therefore, most congestion control algorithms are implemented in the protocol TCP.
We consider the dynamic fluid-flow model introduced in *(Hollot C., 2002)* for describing the behavior of a TCP/AQM network. Mathematical model *Eq. (1)* is a nonlinear time-varying differential equations model for TCP networks:
Windows dynamic

$$\dot{w}(t) = \frac{1}{R(t)} - \frac{w(t)}{2}\frac{w(t-R(t))}{R(t-R(t))}p(t-R(t))$$

Queue dynamic

$$\dot{q}(t) = \begin{cases} -C + \dfrac{N(t)}{R(t)}w(t) & q > 0 \\ \max(0, -C + \dfrac{N(t)}{R(t)}w(t)) & q < 0 \end{cases} \quad (1)$$

$$R(t) = \frac{q(t)}{C} + T_p$$

In these equations:

$w(t)$: Average TCP time window size (packets);
$q(t)$: Average queue length (packets);
$R(t)$: Round trip time (sec);
C : Link capacity (packets/sec);
T_p : Propagation delay (sec);
N : Number of TCP sessions;
p : Probability of packet mark;
The specifications of the TCP network are from *(Hollot C., 2002).*

3 Introduction of different algorithms reposed for AQM

As shown in Figure 2 the purpose of active queue management is the design of controller (making queue size dynamically closer to the predetermined value q) or in other words reducing the error, $e(t) = q_{ref}(t) - q(t)$ by a control signal, $u(t) = p(t)$. Control signal changes in the interval [0, 1] and the reference signal $q_{ref}(t)$ must be less than the maximum queue capacity. We must consider that the reference signal will be variable.

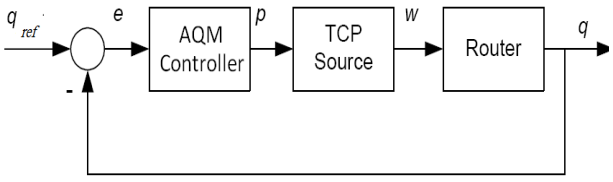


Figure 2: AQM controller for TCP Network

To design and obtain the parameters of RED, PI, PID and APC, a linear model of *Eq. (1)* around the operation points W_0, R_0, Q_0 and p_0 is used. Linear form of equations is shown as follow:

$$\delta\dot{W}(t) = -\frac{N}{R_0^2 C}(\delta W(t) + \delta W(t - R_0)) - \frac{1}{R_0^2 C}(\delta q(t) - \delta q(t - R_0)) - \frac{R_o C^2}{2N^2}\delta p(t - R_0) \quad (2)$$

$$\delta\dot{q}(t) = \frac{N}{R_0}\delta W(t) - \frac{1}{R_0}\delta q(t)$$

3.1 Tuning RED

RED takes an average measure of the queue length and randomly drops packets which are within a threshold between min and max of Average queue length *(Hollot C., 2002).*

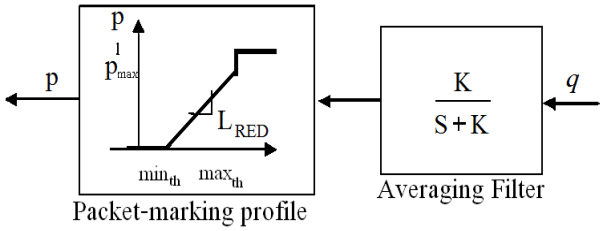


Figure 3: The RED algorithm

Gradient:

$$L_{red} = \frac{p_{\max}}{\max_{th} - \min_{th}} \quad (3)$$

Then selected K as low pass filter pole, $q_{\min}$ and $q_{\max}$ as threshold of average queue length and gain L_{red}, therefore transfer function of RED:

$$C(s) = \frac{K * L_{red}}{(s + K)} \quad (4)$$

We selected optimal parameters value for RED control from iterative numerical analyses using *(Hollot C., 2002)* under the given RED specifications with $p_{\max} = 0.1$, $q_{\max} = 800$, $\min_{th} = 150$, $\max_{th} = 700$:

$$L_{red} = 1.86 * 10^{-4} \quad , \quad K = 5 * 10^{-3}$$

$$C(s) = \frac{(5*10^{-3})(1.86*10^{-4})}{(s + 0.005)} \quad (5)$$

3.2 PI and PID Controller

To improve the controller performance, classical controllers PI and PID are used with linear model of *Eq. (1)* around the operation points $N = 60$, $W_0 = 15$, $p_0 = 0.008$ and $R_0 = 0.246s$ *(Hollot . , 2002).*

PI: $$C(s) = 9.64 * 10^{-6} \frac{\frac{s}{0.53} + 1}{s} \quad (6)$$

PID: $$C(s) = 5.1 * 10^{-5} \frac{s^2 + 1.22s + 0.61}{s} \quad (7)$$

3.3 Adaptive Prediction Controller APC

For designing the APC, we use the method presented in reference *(Torkamandi M., 2007)*. In this reference $(\alpha_0, \alpha_1, ..., \alpha_p)$ and $(\beta_0, \beta_1, ..., \beta_q)$ parameters are necessary for calculating of $\hat{Q}(t+kt)$ and control signal $u(t)$. The adaptive setting of parameters is shown in Figure 4.

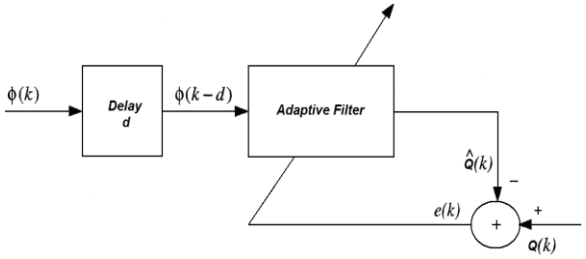


Figure 4: Adaptive setting of parameters

To obtain the prediction model assuming the theory of adaptive signal processing, the error was calculated by the difference of estimation value $\hat{Q}(t)$ and actual value $Q(t)$.
With the following equation:

$$e(t) = \mathrm{Q(t)} - \hat{\mathrm{Q}}\mathrm{(t)} \quad (8)$$

This error is used to set the unknown parameters. For this calculation the well-known algorithm [1] NLMS, is used.

$$Q(t) = [\hat{\alpha}_0 Q(t-d) + \hat{\alpha}_1 Q(t-d-1) + ... + \hat{\alpha}_p Q(t-d-p) + \hat{\beta}_0 \Delta u(t-d-1) + \hat{\beta}_1 \Delta u(t-d-2) + ... + \hat{\beta}_q \Delta u(t-d-q-1)] \quad (9)$$

There is:

$$\hat{\theta}(t) = [\hat{\alpha}_1(t), ..., \hat{\alpha}_p(t), \hat{\beta}_1(t), ..., \hat{\beta}_q(t)]^T \quad (10)$$

And

$$\phi(t-d) = [Q(t-d), ..., Q(t-d-p), \Delta u(t-d-1), .., \Delta u(t-d-q-1)]^T \quad (11)$$

For the initial state, we choose $\hat{\theta}(0) = 0$.
To set the parameters we use the following formula:

$$\hat{\theta}(t+1) = \hat{\theta}(t) + \frac{\mu}{a + \|\phi(t-d)\|^2} \phi(t-d) e(t) \quad (12)$$

By selecting $p = 1$, $q = 1$ and $d = 8$ we will have queue size, q as the output system and the packets drop probability, $p = 1$ as the control signal.

$$Q(t) = [\hat{\alpha}_0 Q(t-8) + \hat{\alpha}_1 Q(t-9) + \hat{\beta}_0 \Delta p(t-9) + \hat{\beta}_1 \Delta p(t-10)] = \hat{\theta}(k)^T \phi(t-8) \quad (13)$$

$$\hat{\theta}(k+1) = \hat{\theta}(k) + \frac{\mu}{a + \|\phi(k-8)\|^2} \phi(k-8) e(k) \quad (14)$$

And at the end, the block diagram of APC controller is shown in Figure 5.

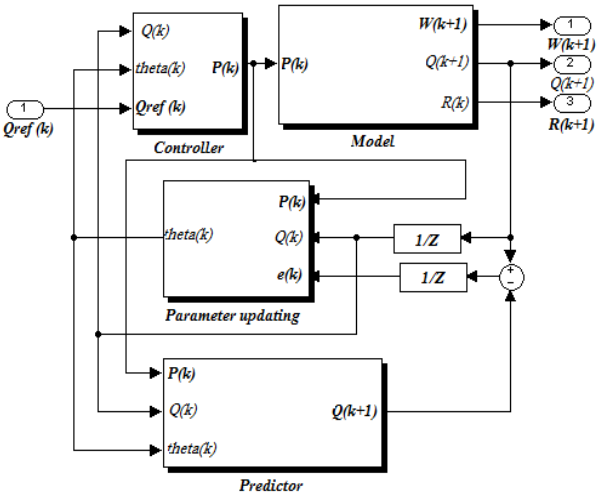


Figure 5: APC block diagram

3.4 Neural network controller with BP training

The neural network model used in *(Cho Hyun C., 2005)* is shown in Figure 6. This neural network includes one feedback connection and a three-layer perceptron.
The dynamic behavior of the network is given by:

$$y_{k+1} = \alpha y_k + \gamma^T (Vu) + b \quad (15)$$

Where α is the feedback gain, k denotes discrete time, and b is a bias connected with unit input. Finally, the network output is obtained from the activation function:

$$p = \varphi(y) = \frac{1}{1 + \exp(-\sigma y)}, \quad \sigma > 0 \quad (16)$$

Where, σ is a constant scaling factor.

1- Normalized Least Mean Square

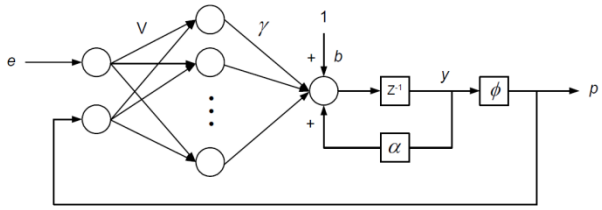


Figure 6: Neural network controller by BP training

In this scheme we have a problem: if the activation function of the output layer *Eq. (16)* is nonlinear and the previous layers *Eq. (15)* are linear, all layers come before the output layer turns to a single-layer, therefore in this case we have not a real neural network.

4 Neural AQM controller design with genetic training

As mentioned in previous algorithms, linear model of TCP is used for designing control. Our target in this article is designing a controller with a nonlinear model of TCP. This is appropriate because a nonlinear control structure such as neural networks can be used to control a nonlinear system. A very effective method for training neural networks control is using of genetic algorithm *(Seiffert Udo., 2001).*

4.1 Description of neural network and relationship between weights and chromosomes for genetic algorithm

In overview of the simple neural network controller, we select a dynamic neural model including one feedback connection and three-layer perceptron. This neural network is shown in Figure 7.

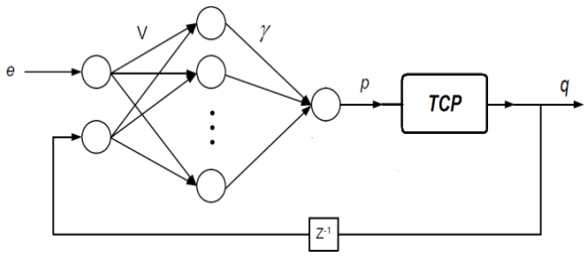


Figure 7: Overview of neural controller

The input vector of this neural network includes the error signal, *e* and the output, *q* as a feedback signal. Thus, the input vector, *u* is given by

$$u = [\,\mathrm{e}\;\;\mathrm{q}\,]^T \tag{17}$$

The weight matrix in the first layer is V_{m*n} where *m* denotes the number of nodes and *n* indicates the number of input, in this case *n=2*.
And the weight vector in the second layer is

$$\gamma = [\gamma_1 .. \gamma_m]^T \tag{18}$$

We determine the weight range value in the interval [-a, a] and the activation function is obtained for each neuron in this neural network as follow:

$$p = \frac{1}{1+\exp(-\zeta y)} \tag{19}$$

Where, ζ is a constant scaling factor.
In training stage we should find the relationship between neural network weights and chromosomes for genetic algorithm. Neural network structure is shown in Figure 7. Weight and bias related to each neuron are placed behind each other and chromosomes occur *(Seiffert Udo., 2001).* This relationship is shown in Figure 8.

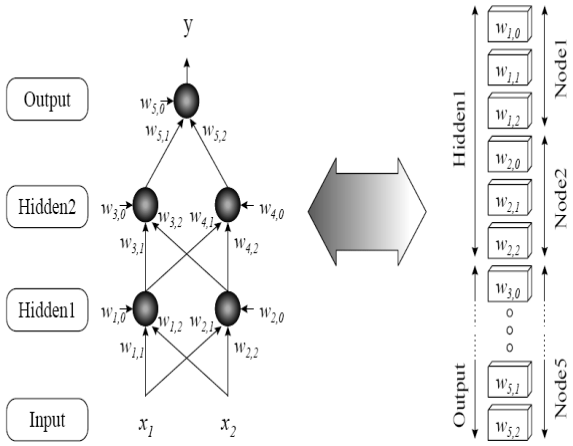


Figure 8: Mapping the weights of the neural network into a chromosome

4.2 Design and training of neural network controller by genetic algorithm

In previous section we had a neural network with three-layer perceptron with input vector, *u* and the output, *q*.
In this structure for network training, genetic algorithm used minimum *E* as target function:

$$\min E = \sum_{k=1}^{T} (q(k) - \hat{q}(k))^2 \tag{20}$$

Here the inverse minimum *E* is used as fitness function for genetic algorithm training:

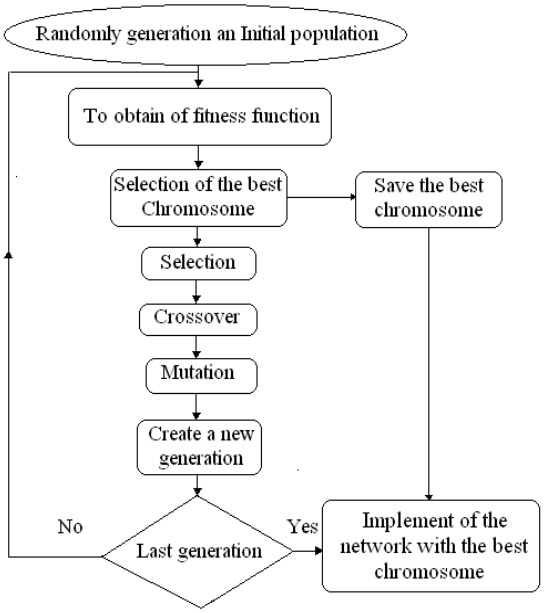
$$\max \mathit{fitness} = \frac{1}{E} \tag{21}$$

The performance of genetic algorithm for designing neural controller is shown in block *diagram (1).*
In this article, different structures are considered for neural networks (different input and different number of neurons) that have been trained by genetic algorithm. Best results obtained by the network are shown in Figure 9.
In this neural network, the weight matrix in the first layer is defined by 3 neurons (V_{4*3}) and also the

number of neurons in the second layer is defined four (γ_{4*1}). The network parameters such as N, the number of TCP connections and $R(t)$, the round-trip time are not constant, therefore dynamic network has delay and time-variable. To have less effects of these parameters on output, we must use the $e(t)$ and $q(t)$ as a feedback signal. In this case the neural network input will include three components: error signal, $e(t)$ error signal with propagation delay, $e(t-T_p)$ and queue size, $q(t)$. The input vector is given for controller as follows:

$$input = \left[(e(t)\ \ e(t - T_p)\ \ q(t - T_p) \right]^T . \qquad (22)$$

The steps can be followed to obtain the weights of neural networks by using block *diagram (1).*



Block diagram 1. Genetic algorithm implementation process to determine weights for the neural network

After examining different networks the best neural network controller design will be achieved such as a network with the following structure:

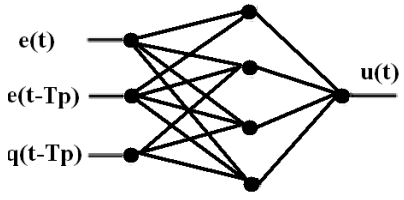


Figure 9: Structure neural network with 3 input and 4 neurons in hidden layer

4.3 Design of neural network controller for different traffic flows

As mentioned, variations in the number of network connections and incoming packet sizes are the unpredictable traffic volume. To overcome this problem *(Hyun C., 2008),* the proposed system considers distinct neural AQMs for different traffic characteristics. Specifically, a TCP network with three different traffic scenarios is adopted: light traffic, medium traffic and heavy traffic. Accordingly, we construct three corresponding modular neural AQMs with different neural network parameters, i.e.

1) Neural AQM I → light traffic
2) Neural AQM II → medium traffic
3) Neural AQM III → heavy traffic

As incoming traffic load is increased due to a growth in the number of network connections, the queue size in the router increases. For this case, the desired queue occupation level must be enlarged to efficiently utilize network resources.

In these cases we have three neural networks that trained under different traffic flows. Following training, we have optimal controller parameters for each network scenario. In real-time implementation, one of the three controllers must be selected based on dynamically changing TCP network conditions, it is shown in Figure 10. We note that each controller is trained for a distinct traffic level at which it performs best. Therefore, we select the network control that best matches traffic level.

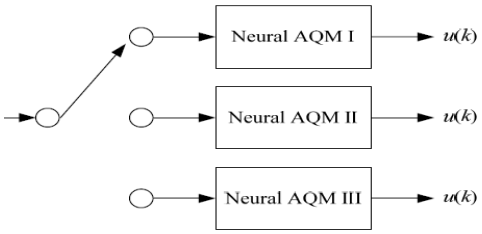


Figure 10: Neural controller selection for different traffic flows

5 Simulation results

For the neural controller simulation of Fluid-Flow model of TCP with (1), we introduced 60 TCP flows ($N = 60$), a reference queue size of 200 packets ($Q_{ref} = 200$), the capacity C is $15 Mbs$ and the propagation delay T_p is $0.16s$ for the simulation of RED, PI, PID and APC controller, we should use the linear model of TCP around the operating points $W_0 = 15$, $p_0 = 0.008$, $R_0 = 0.246s$ and it was used in *Eq*. (2).

We selected neural network weight and bias in the interval [-1, 1] and created the initial population with selected reference signal, $Q_{ref}(t)$ for all generations, the neural network trained by genetic algorithm is performed up to 500 generations. In Figure 11 we can see the error reduction in this algorithm performance.

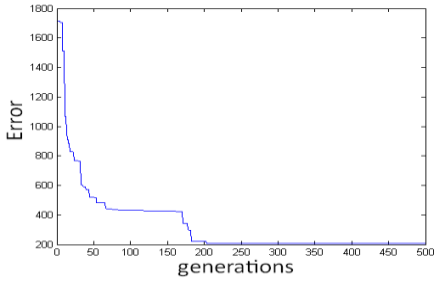


Figure 11: Training process of genetic algorithm

We ran five algorithms by constant signal $Q_{ref}(t) = 200$ to evaluate the different AQM approaches: RED algorithm, PI control, PID control, APC algorithm and neural network control. In Figure 12 simulation results are shown.

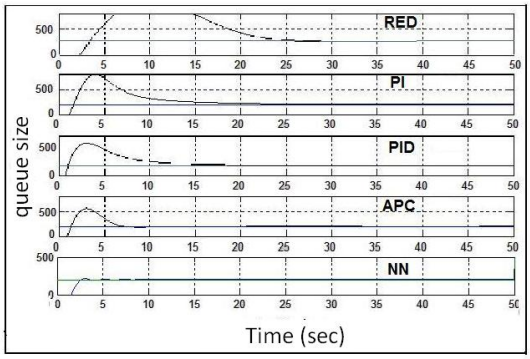


Figure 12: Performance of different algorithms with $Q_{ref}(t) = 200$

Figure 13 shows the simulation results for the queue dynamics, these results indicate that neural AQM performs more effectively than PI, PID and APC controller for varying reference queue size $Q_{ref}(t)$.

The reference queue size is:

$$Q_{ref}(t) = 50 * u(t) + 200 * u(t-30) - 100 * u(t-50) + 200 * u(t-80) - 100 * u(t-100) \quad (23)$$

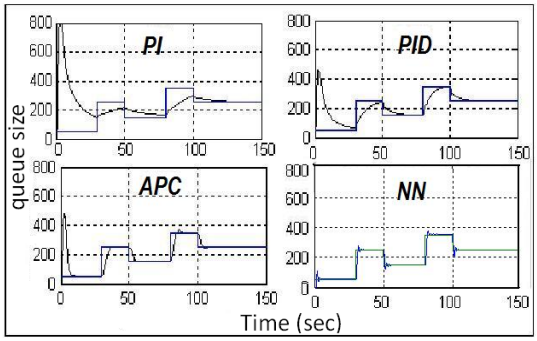


Figure 13: Performance of different algorithms with $Q_{ref}(t) = 50 * u(t) + 200 * u(t-30) - 100 * u(t-50) + 200 * u(t-80) - 100 * u(t-100)$

As mentioned in section 4.3, this simulation is intended to evaluate the modular neural AQM. We separately trained three neural AQMs with different network connections and reference queue lengths, as

NAQM I: *N* ~ *Uniform* (200, 400) & *r*(*t*) = 200 packets
NAQM II: *N* ~ *Uniform* (400, 600) & *r*(*t*) = 400 packets
NAQM III:*N* ~ *Uniform* (600, 800) & *r*(*t*) = 600 packets
Figure 14 shows the time-history of the dynamic queue length for PID and neural AQM.

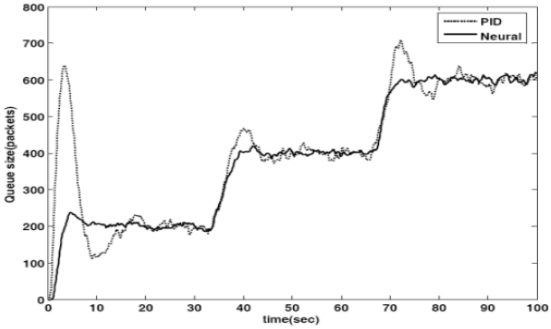


Figure 14: Simulation results of PID and modular neural AQMs

6 Conclusions

In this paper we presented a novel AQM methodology using a dynamic neural network for TCP congestion control. The neural network is trained by genetic algorithm. The results were compared through the simulation of different algorithms and controllers. The neural AQM controller is designed directly from the non-linear model *Eq. (1)*.This type of controller design (neural AQM) is superior to the previous controller design such as PI, RED, PID and APC. For training neural network, Genetic Algorithm is used. Simulation results indicate that this design is the best methods.

References

Tanenbaum Andrew S. Computers Networks, Prentice-Hall Inc. New Jersey, 4'th Edition, 2003.

Hollot C. V., Misra V., Towsley D. and Gong W. B. Analysis and design of controllers for AQM routers supporting TCP flows. "IEEE Trans. - 2002".

Cho Hyun C., Sami Fadali M., Lee Hyunjeong Neural Network Control for TCP Network Congestion. International American Conference "Control Conference", Portland, OR, USA, June, 2005.

Seiffert Udo. Multiple Layer Perceptron Training Using Genetic Algorithms. ESANN'2001 proceedings of European Symposium "Artificial Neural Networks", Bruges (Belgium), April 2001.

Torkamandi M., Hamidi Beheshti M.T. Adaptive Predictive Congestion Con trot Design for TCP/IP Networks. University of Tehran Electronic Journals no.107, December 2007. p. 587-596 (in Persian)

Hyun C. Cho, Sami Fadali M., Hyunjeong Lee. Adaptive Neural Queue Management for TCP Networks. ACM portal, Computers and Electrical Engineering, (November 2008).