2005 American Control Conference
June 8-10, 2005. Portland, OR, USA

Neural Network Control for TCP Network Congestion

Hyun C. Cho, M. Sami Fadali, Hyunjeong Lee

Electrical Engineering/260, University of Nevada, Reno, NV 89557

Abstract - **Active Queue Management (AQM) has been widely used for congestion avoidance in Transmission Control Protocol (TCP) networks. Although numerous AQM schemes have been proposed to regulate a queue size close to a reference level, most of them are incapable of adequately adapting to TCP network dynamics due to TCP's non-linearity and time-varying stochastic properties. To alleviate these problems, we introduce an AQM technique based on a dynamic neural network using the Back-Propagation (BP) algorithm. The dynamic neural network is designed to perform as a robust adaptive feedback controller for TCP dynamics after an adequate training period. We evaluate the performances of the proposed neural network AQM approach using simulation experiments. The proposed approach yields superior performance with faster transient time, larger throughput, and higher link utilization compared to two existing schemes: Random Early Detection (RED) and Proportional-Integral (PI)-based AQM. The neural AQM outperformed PI control and RED, especially in transient state and TCP dynamics variation.**

I. INTRODUCTION

The essence of congestion control strategies for TCP networks is to rapidly recover from network congestion, or to prevent an incipient congestion. This can be achieved by dynamically adjusting window size at the source side or controlling incoming packets to a router at the link side.

Numerous TCP schemes that optimally adjust window size for congestion avoidance have been explored in the last decade. The first widely used scheme, TCP Tahoe, was later modified to TCP Reno [1], currently the most popular TCP. The congestion window in these protocols is based on the Additive Increase Multiplicative Decrease (AIMD) algorithm: congestion window size is increased by one packet per acknowledgement (ACK) but is halved if a source receives three duplicate ACK signals or does not receive any ACK within a given round-trip time (see [1]). Since the development of TCP Reno, several researchers have suggested additional TCP functions to improve network performance [2]. Whereas these algorithms operate at the source side, AQM is implemented at the link side, especially for incipient congestion avoidance. In other words, AQM provides congestion information acquired from the link side to the sources. The objective of an AQM is primarily to proactively respond to network congestion as its queue begins to increase. Rather than simply waiting for a congested queue to overflow and then tail drop all subsequently arriving packets, it maintains queue size at a predefined level in the router.

RED [3] is a popular example of an AQM scheme. In RED, the router calculates the drop probability using a current queue size. The incoming packets are passed, dropped or marked, based on this probability. By discarding or marking a single packet, the router sends an implicit or explicit warning to the source. As a response to the warning, the source is expected to adjust the congestion window size to reduce its transmission rate. The drop probability is often linearly proportional to queue length. Although RED is an effective TCP congestion control [4], it can induce network instability and major traffic disruption if not properly configured. Hence, optimal parameter selection for RED design under different congestion scenarios has been a problem. Moreover, even if optimally selected, the parameter values must be adjusted in real-time implementations because TCP dynamics change with the number of active TCP flows.

Many studies have addressed optimal parameter selection for RED and its variants. Floyd et al. proposed appropriate parameter ranges in [5] and presented an adaptive RED in which the best parameter settings are based upon a traffic mix flowing through the router [6]. In [7], the authors proposed Random Exponential Marking (REM) as a modification to RED. Their aim was to stabilize the input rate around the link capacity and maintain average queue size around a small reference level. Feng et al. [8] presented a self-configuring RED that adjusted the maximum drop probability according to the past history of the average queue size. They also proposed BLUE [9], a new AQM mechanism. BLUE uses buffer overflow and link idle events, together with average queue size, to control congestion. Another approach, Stabilized RED (SRED) [10], computes the drop probability based on the estimated number of active TCP flows and the instantaneous average queue size.

In recent years, control-theoretic AQM approaches have been proposed, mostly using linear classical control. In [11], design guidelines were proposed for choosing AQM parameters based on Proportional (P) and Proportional-Integral (PI) control. The authors linearized the nonlinear differential equation TCP network model of

[12] at an operating point to derive a transfer function for P and PI controller design. Their AQM scheme was compared to RED and found to be superior.

Kim and Low [13] formulated an AQM design problem for stabilizing a given TCP network described by state-space models and proposed Proportional-Derivative (PD) and Proportional-Integral-Derivative (PID) AQM strategies. In [14], Dynamic-RED (DRED) was proposed to stabilize queue dynamics even if the number of active TCP connections is dynamically varied. DRED aims to maintain queue size close to a reference queue level by a discrete integral control approach. Recently, more complex control methodologies have been proposed for AQM. Quet and Ozbay [15] applied H_∞ controller to AQM using the linearized the TCP model of [12]. They derived the transfer function of the controller and showed through computer simulation that the proposed AQM was superior to PI control and RED.

We note that the choice of control parameters is the key to satisfactory performance of a feedback control system. However, in practice, parameter choices for the nominal model may be suboptimal due to system uncertainty or perturbation. Thus, parameter values must be adjusted to adapt to operational changes. In addition, most control-theoretic AQM proposed to date are based on linear models while TCP networks are nonlinear time-varying stochastic systems.

We present a more sophisticated adaptive control strategy for AQM in TCP networks using a dynamic artificial neural network AQM control. The control can promptly adapt its operation to the nonlinear time-varying and stochastic nature of TCP networks. Neural networks have been widely applied in the last two decades in a variety of engineering fields such like signal processing, process control, communication systems, etc. [16]. They are iteratively trained by a proper learning algorithm to minimize a selected performance measure. As a result, unlike RED and classical linear control based approaches, neural networks are able to determine the optimal AQM system parameters values autonomously after adequate training. Following training, the neural network operates as an adaptive and robust controller that can provide excellent performance even for environmental conditions not included in the training data set.

The dynamic neural network controller presented in this paper is trained to regulate the actual queue size close to a reference value determined by network requirements. After training, the neural network operates as an adaptive controller under changes in TCP dynamics. We choose a multi-layer recurrent (including feedback) dynamic neural model because of its well-known advantages. This model has been popular since the mid 1990's in many applications for dynamical time-varying and nonlinear systems [17]. There are mainly two methods for training recurrent neural networks: a back-propagation-trough-time algorithm [18] and a real-time recurrent learning algorithm [19]. For simplicity, we derive a learning procedure by the general back-propagation (BP) method [16]. To evaluate the proposed neural AQM, a TCP network topology including a simple bottleneck, two routers on the link side, and multiple TCP sessions, is considered. We use the TCP system model of [12] for our neural AQM analysis and illustrate the advantages of our proposed methodology as compared to RED and PI-based AQM.

The outline of this paper is as follows. In Section II, we present our neural network AQM TCP congestion control. A learning algorithm with BP is derived for this model in Section III. Simulation results and discussion are given in Section IV. Our conclusions are given in Section V.

II. NEURAL NETWORK AQM

The block diagram of TCP congestion control with the neural network AQM proposed in this paper is shown in Fig. 1.

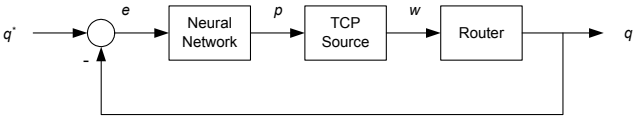


Fig. 1. Neural network AQM of TCP network.

In Fig. 1, the congestion window size, w of the TCP source is determined by the probability, p calculated from the neural network. Queue dynamics at the link side is affected by w. The neural network control system minimizes the error signal, e between the actual queue size, q and the reference queue target value, q^*. The loss probability, p is the control input to the TCP source. The neural network model used in this paper is shown in Fig. 2.

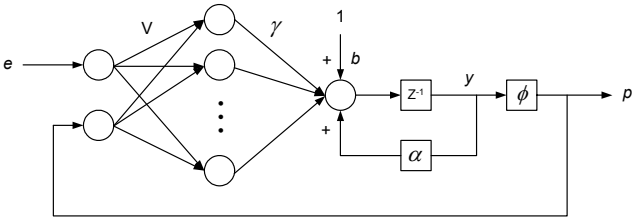


Fig. 2. Recurrent neural network for AQM.

We select a dynamic recurrent neural model including one feedback connection and a three-layer perceptron. The input vector of this neural network includes the error signal, e and the probability, p as a feedback signal from its output. Thus, the input vector, u is given by

$$u = \begin{bmatrix} e & p \end{bmatrix}^T \tag{1}$$

The weight matrix in the first layer is

$$V = \begin{bmatrix} v_{11} & v_{12} \\ \vdots & \vdots \\ v_{m1} & v_{m2} \end{bmatrix} \tag{2}$$

where m denotes the number of nodes in the second layer. In (2), the first column and the second column are related to the error signal and the feedback probability, respectively.

The weight vector in the second layer is

$$\gamma = \begin{bmatrix} \gamma_1 & \cdots & \gamma_m \end{bmatrix}^T \quad (3)$$

Thus, the dynamic behavior of the network is given by

$$y_{k+1} = \alpha y_k + \gamma^T (Vu) + b \quad (4)$$

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

$$p = \phi(y) = \frac{1}{1+\exp(-ay)}, \qquad a > 0 \quad (5)$$

where a is a constant scaling factor.

III. LEARNING PROCEDURE

The neural network designed in Section II must be trained to optimize a TCP network performance measure. During network training, the weights and the bias are iteratively updated until they reach their optimal values. In this section, we present a BP learning algorithm for the proposed network and derive the rules for updating the network weights and bias. The training objective is to minimize the error signal J defined as

$$J = \frac{1}{2}(q^* - q)^2 \quad (6)$$

Adjustments of the weight matrix V, the weight vector γ, and the bias b, are governed by the delta rule as follows

$$\Delta v_{ij} = -\eta \frac{\partial J}{\partial v_{ij}} \quad (7)$$

$$\Delta \gamma_i = -\eta \frac{\partial J}{\partial \gamma_i} \quad (8)$$

$$\Delta b = -\eta \frac{\partial J}{\partial b} \quad (9)$$

where η is the learning rate, i=1,…,m, and j=1, 2. The partial differential equations in the right side of (7), (8), and (9) are expanded respectively by using the chain rule as

$$\frac{\partial J}{\partial v_{ij}} = \frac{\partial J}{\partial q}\frac{\partial q}{\partial p}\frac{\partial p}{\partial y}\frac{\partial y}{\partial v_{ij}} \quad (10)$$

$$\frac{\partial J}{\partial \gamma_i} = \frac{\partial J}{\partial q}\frac{\partial q}{\partial p}\frac{\partial p}{\partial y}\frac{\partial y}{\partial \gamma_i} \quad (11)$$

$$\frac{\partial J}{\partial b} = \frac{\partial J}{\partial q}\frac{\partial q}{\partial p}\frac{\partial p}{\partial y}\frac{\partial y}{\partial b} \quad (12)$$

We calculate the three common terms in (10), (11), and (12) using (5), (6), and the following approximations [20]

$$\frac{\partial J}{\partial q} = -(q^* - q) \quad (13)$$

$$\frac{\partial p}{\partial y} = \frac{a\exp(-ay)}{\left[1+\exp(-ay)\right]^2} \quad (14)$$

$$\frac{\partial q}{\partial p} \approx \frac{q(k)-q(k-1)}{p(k)-p(k-1)} \quad (15)$$

These approximations describe the Jacobian of the TCP system. We use an approximation of the TCP system equation assuming that it is not provided. The approximate derivative in (15) is determined by changing the input p and the output q [20].

The right hand sides of (10), (11), and (12) include derivatives of y obtained using the following equations

$$\frac{\partial y(k+1)}{\partial v_{ij}(k)} = \gamma_i u_j \quad (16)$$

$$\frac{\partial y(k+1)}{\partial \gamma_i(k)} = \sum_{j=1}^{2} v_{ij} u_j \quad (17)$$

$$\frac{\partial y(k+1)}{\partial b(k)} = 1 \quad (18)$$

By substituting (13)-(18) in (7), (8), and (9), we finally obtain the update rules

$$\Delta v_{ij} = \eta \delta \gamma_i u_j \quad (19)$$

$$\Delta \gamma_i = \eta \delta \sum_{j=1}^{2} v_{ij} u_j \quad (20)$$

$$\Delta b = \eta \delta \quad (21)$$

where

$$\delta = (q^* - q)\left(\frac{q(k)-q(k-1)}{p(k)-p(k-1)}\right)\left(\frac{a\exp(-ay)}{\left[1+\exp(-ay)\right]^2}\right) \quad (22)$$

IV. SIMULATIONS

We conducted a simulation study to evaluate the performance of neural network AQM. We considered the TCP network topology of Fig. 3, including a simple bottleneck link between two routers and numerous TCP flows.

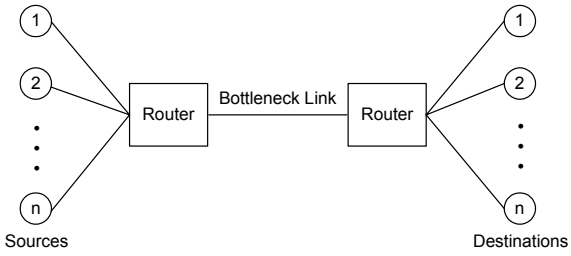


Fig. 3. TCP Network model.

The mathematical model used for AQM design and simulation is a fluid-flow expression [12]. The model describes the dynamics of a queue and a congestion window with the nonlinear differential equations

$$\dot{w}(t) = \frac{1}{R(t)} - \frac{w(t)\,w(t-R(t))}{2R(t-R(t))}\,p(t-R(t)) \quad (23)$$

$$\dot{q}(t) = \begin{cases} N(t)x(t) - C & if\ N(t)x(t) \geq C \\ 0 & otherwise \end{cases} \quad (24)$$

where C is a link capacity, N is the number of TCP connections, $x(t)$ is a transmission rate defined as $w(t)/R(t)$, and the round-trip time $R(t)$ is calculated by $R(t) = q(t)/C + \tau$, where τ is a random propagation delay time. The specifications of the TCP network are from [11], but some parameter values are modified for our simulation

scenarios. We select the packet size as 512 bytes, C as 15 Mb/sec, and a maximum q in a router as 800 packets. τ is uniformly distributed in [0.16, 0.24] sec.

We simulate RED and PI control under same simulation scenarios as our control for comparison purposes. We selected optimal parameter values for RED and PI control from iterative numerical analyses using (23) and (24) under the given TCP specifications with 240 TCP connections. In RED, the minimum and maximum thresholds of an averaging queue size are 200 and 250 packets respectively, the maximum drop probability is 0.1, and the weight in the moving average equations for computing the averaging queue is 0.03. The PI controller used in this simulation is given by

$$p = k_p e(t) + k_i \int e(t)dt \tag{25}$$

where k_p is the proportional gain and k_i is the integral gain. We selected k_p=5×10^{-7} and k_i=5×10^{-5} for our simulations, which are of the same order as the values of [11] and [15].

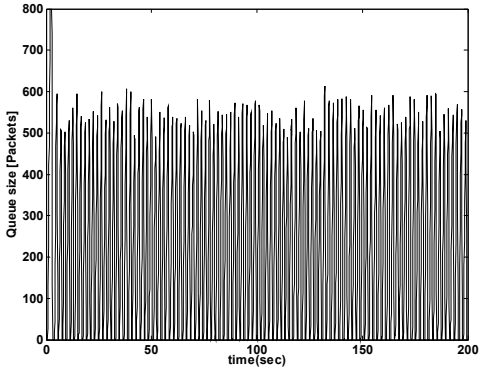
For the AQM neural network, we use three nodes in the hidden layer and a learning rate of 0.1. The initial values of the weights in the first and second layers, and the bias, were uniformly distributed in [–1, 1]. The neural network was trained to determine the optimal weights and bias under the same simulation environment as RED and PI AQM. After iterative network training with randomly chosen initial weights and bias, the optimal weights that minimize our control performance measure were

$$V = \begin{bmatrix} -0.5536 & -0.4417 \\ -0.4545 & -0.6853 \\ 0.2207 & -0.7065 \end{bmatrix} \tag{26}$$

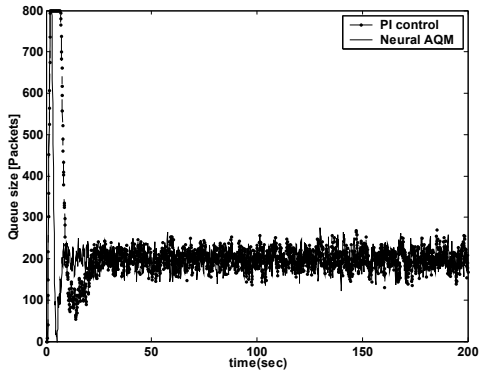
$$\gamma = \begin{bmatrix} 0.3902 & 0.7771 & 0.3087 \end{bmatrix}^T \tag{27}$$

and the bias b was 0.8444. We ran four simulation scenarios to evaluate the three different AQM approaches: RED, PI control, and neural network control. We also tested the robustness and adaptive capacity of the three schemes. To illustrate dynamic queue responses applying these AQM methods, we solve the differential equations in (23) and (24) numerically.

Case I: We used 240 TCP flows (N=240) and a reference queue size of 200 packets for PI control and neural AQM. Time histories of the queue size for the three AQMs are shown in Fig. 4. We observe an initial overshoot in all three AQMs after which the responses drop to their steady-state values. The overshoot saturates at 800 packets due to the maximum queue size in the router. In the steady state, large oscillations continue for RED while the responses for PI control and neural AQM oscillate very closely to the reference level. PI and neural AQM control have considerably different transient responses. For PI control, the settling time is about 25 sec and is three times that of the neural AQM, and transient saturation occurred as in RED. This behavior is a very serious problem that can potentially result in network congestion especially traffic status changes rapidly. By contrast, neural AQM has a much faster response but does not result in saturation. The comparison indicates that neural AQM provides stable control and a better transient response than PI-based AQM.



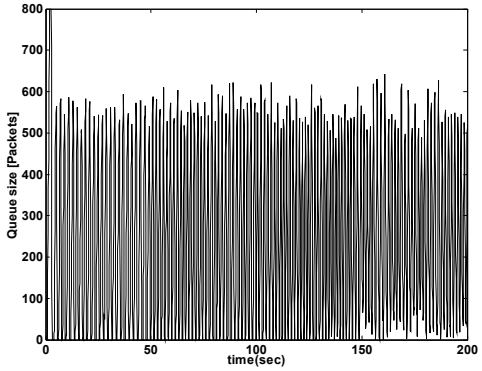
(a) RED



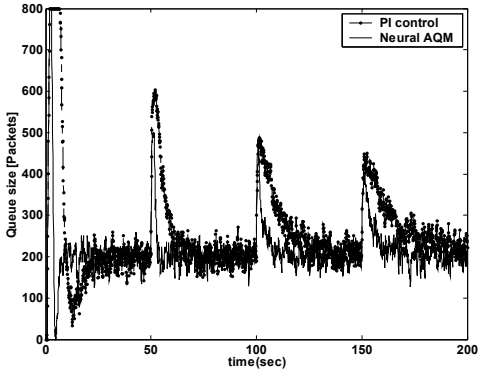
(b) PI and neural AQM

Fig. 4. Queue dynamics for fixed N.

Case II-1: In real-time TCP implementations, the number of TCP flows varies randomly. Thus, we simulated the system with time-varying TCP flows, i.e., the number of TCP connections was varied. We assume that N is progressively increased by 100 every 50 sec, i.e. N =240 in [0, 50] sec, N =340 in [50, 100] sec, N =440 in [100, 150] sec, and N =540 in [150, 200] sec. The reference queue size in PI control and neural AQM is still set to 200 packets. Fig. 5 shows the queue dynamics for this simulation scenario. Fig.5 (a) shows that the response in RED is very similar to *Case* I. In Fig. 5(b), both PI and neural AQM still have the initial overshoots at the starting time as well as at the times when N is changed. The response for PI control initially saturates and has a larger settling time than neural AQM. For both, the overshoot increases while the undershoot decreases as N increases. These results show that neural AQM outperforms PI AQM for dynamic reference queue level.



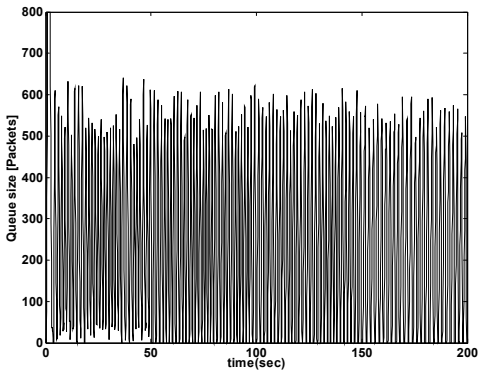
(a) RED



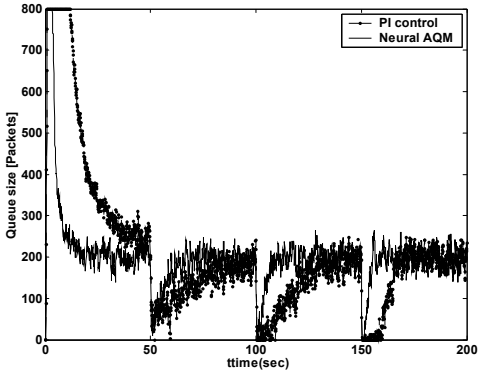
(b) PI and neural AQM

Fig. 5. Queue dynamics for increasing *N*.

Case II-2: This example is the opposite scenario of *Case* II-1. In this case, the number of TCP flows are dynamically decreased, that is, *N*=540 in [0, 50], *N*=440 in [50, 100], *N*=340 in [100, 150], and *N*=240 in [150, 200]. Time histories of the queue size for these three AQMs are plotted in Fig. 6. In this case, undershoots occurred in both PI and neural AQM but we again observe that the neural AQM has a superior transient response to PI-based AQM. The superior transient response of the proposed solution in this case directly explains the higher throughput compared to PI-based AQM because the system can quickly adjust itself to fully utilize the available bandwidth due to less traffic.



(a) RED



(b) PI and neural AQM

Fig. 6. Queue dynamics for decreasing *N*.

Case III: The simulation in this case is for comparisons of the two feedback control schemes: PI-based AQM and neural AQM. RED is not included because it is not a feedback control AQM scheme. We vary the reference queue size, but keep *N* fixed at 240. We set a reference queue level, q^* as 200 packets both in [0, 50] sec and [100,150] sec, and as 400 packets both in [50,100] and [150,200] sec. Fig. 7 shows the simulation results for the queue dynamics. These results indicate that neural AQM has better performance than PI control for varying reference queue size.

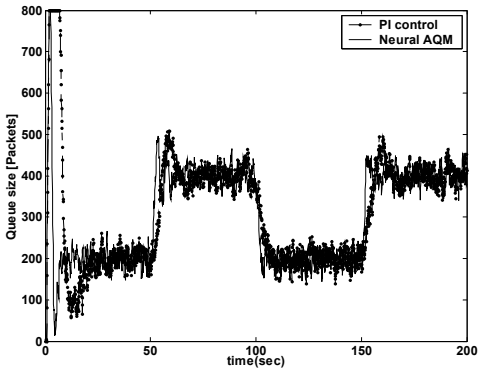


Fig. 7. Queue size for PI and neural AQM for varying q^*.

Table 1 shows the mean values and variances of the throughput and the queue size for each AQM. The queue size of neural AQM has the smallest mean values and the largest throughput in all simulations. Higher throughput implies more efficient utilization of the network link. The mean queue size of neural AQM remains closer to the reference queue value than PI control, and the variance of its throughput is the smallest for all cases. Hence, neural AQM provides more stable queue management.

Table 1. Simulation results for RED, PI control, and neural AQM

Case	Control	Mean		Variance	
		Queue size [Packets]	Throughput [Packets/sec]	Queue size [Packets]	Throughput [Packets/sec]
I	RED	232.11	15.29	4.44×10^4	20.30
	PI	214.38	15.96	1.17×10^4	3.73
	NN	201.26	16.06	3.70×10^3	3.08
II-1	RED	248.35	10.63	4.03×10^4	18.06
	PI	262.99	10.78	1.56×10^4	15.03
	NN	214.80	10.89	5.15×10^3	13.06
II-2	RED	248.03	10.61	4.14×10^4	18.53
	PI	214.28	10.79	3.36×10^4	12.64
	NN	201.21	10.97	8.59×10^3	12.37
III	PI	311.89	15.94	1.90×10^4	3.43
	NN	300.50	16.00	1.34×10^4	2.66

V. CONCLUSION

We presented a novel AQM methodology using a dynamic neural network for TCP congestion control. The neural network acts as a feedback controller to maintain the actual queue size close to a reference target. The neural network is trained by a BP algorithm. We applied the neural AQM to a single bottleneck network supporting multiple TCP flows. Four scenarios were examined in the simulation experiments to compare neural AQM to RED and PI-based AQM. While PI AQM resulted in queue saturation and larger overshoot, neural AQM reduced overshoot and eliminated saturation. Neural AQM was more stable with no packet loss due to congestion. Especially for the case of time-varying TCP dynamics, the neural AQM was superior. We conclude that neural AQM is an effective adaptive controller and provides higher Quality of Service (QoS) in TCP networks. Future work will extend our results to more complex network scenarios, such as heterogeneous RTTs, short TCP connections or noise disturbance networks, and different TCP data streams and will include various simulation scenarios using a network simulation tool such as OPNET to verify our results.

REFERENCES

[1] V. Jacobson and M. Karels, "Congestion avoidance and control," *Proceedings of ACM SIGCOMM*, pp. 314 - 329, 1988.

[2] S. Floyd, "A report on recent developments in TCP congestion control," *IEEE Communications Magazine*, vol. 39, no. 4, pp. 84 – 90, 2001.

[3] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397 - 413, 1993.

[4] T. Bonald, M. May, and J.-C. Bolot, "Analytic evaluation of RED performance," *Proceeding of IEEE INFOCOM*, pp. 1415 - 1424, 2000.

[5] S. Floyd, "Recommendations on using the gentle variant of RED," http://www.aciri.org/floyd/red/gentle.html, 2000.

[6] S. Floyd, R. Gummadi, and S. Shenker, "Adaptive RED: An algorithm for increasing the robustness of RED's active queue management," http://www.icir.org/floyd/papers.html, 2001.

[7] S. Athuraliya, V. H. Li, S. H. Low, and Q. Yin, "REM: active queue management," IEEE Network, vol. 15, no. 3, pp. 48 - 53, 2001.

[8] W. Feng, D. D. Kandlur, D. Saha, and K. G. Shin, "A self-configuring RED gateway," *Proceedings of IEEE INFOCOM*, pp. 1320 - 1328, 1999.

[9] W. Feng, D. D. Kandlur, D. Saha, and K. G. Shin, "Stochastic fair Blue: a queue management algorithm for enforcing fairness," *Proceedings of IEEE INFOCOM*, pp. 1520 - 1529, 2001.

[10] Y. J. Ott, T. V. Lakshman, and L. H. Wong, "SRED: stabilized RED," *Proceedings of IEEE INFOCOM*, pp. 1346 - 1355, 1999.

[11] C. V. Hollot, V. Misra, D. Towsley, and W. Gong, "Analysis and design of controllers for AQM routers supporting TCP flows," *IEEE Trans. on Automatic Control*, vol. 47, no. 6, pp. 945 - 959, 2002.

[12] V. Misra, W. B. Gong, and D. Towsley, "Fluid-based analysis of a network of AQM routers supporting TCP flows with an application to RED," *Proceedings of ACM/SIGCOM*, pp. 151-160, 2000.

[13] K. B. Kim and S. H. Low, "Analysis and design of AQM based on state-space models for stabilizing TCP," *Proceedings of American Control Conference*, pp. 260 - 265, 2003.

[14] J. Aweya, M. Ouellette, and D. Y. Montuno, "A control theoretic approach to active queue management," Computer Networks, vol. 36, pp. 203 - 235, 2001.

[15] P.-F. Quet and H. Ozbay, "On the design of AQM supporting TCP flows using robust control theory," *IEEE Transition on Automatic Control*, vol. 49, no. 6, 2004.

[16] S. Haykin, *Neural networks: A comprehensive foundation*, Prentice Hall, 1999.

[17] L.R. Medsker and L.C. Jain, *Recurrent neural networks: design and applications*, CRC Press, 2000.

[18] P. J. Werbos, "Back-propagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550 - 1560, 1990.

[19] R. J. Williams and J. Peng, "An effective gradient-based algorithm for on-line training of recurrent network trajectories," *Neural Computation*, vol. 2, pp. 490 – 501, 1990.

[20] Guez, J. L. Eilbert, and M. Kam, "Neural network architecture for control," *IEEE Control Systems Magazine*, vol. 8, no. 2, pp. 22 - 25, 1988.