Shahram JAMALI[1], Mehdi NOSRATI[2], Seyed Naser SEYED HASHEMI[3]

University of Mohaghegh Ardabili (1), Department of computer engineering, Payame Noor University (2), Islamic Azad University(3)

Prediction-based Active Queue Management in the Internet

***Abstract**. Random early detection (RED) is the most popular active queue management algorithm that is used by the Internet routers. This paper proposes a neuro-fuzzy controller which enhances the network performance by dynamically tuning of RED's max$_p$ parameter. The controller first learns the network behavior against max$_p$ variations and then adjusts max$_p$. Simulation results in ns-2 environment show that, the proposed learning RED, called LRED, keeps queue length and queuing delay in a pre-determined level and outperforms RED in terms of queue length and stability.*

***Streszczenie.** W artykule zaprezentowano sterownik neuro-fuzzy który poprawia dynamiczne strojenie system RED stosowanego do kolejkowania w Internecie. Proponowany uczący się algorytm nazwany LRED pozwala na utrzymanie długości kolejki i opóźnienia w założonych granicach. (**Aktywne zarządzanie kolejką w Internecie bazujące na przewidywaniu**)*

Keywords: Active Queue Management, neuro-fuzzy, LRED.
Słowa kluczowe: Internet, kolejkowanie, RED.

Introduction

RED [3] is the most famous AQM control schemes explicitly introduced for congestion control that solves some major drawbacks of former approaches such as global synchronization. Due to its popularity, RED (or its variants) has been implemented by many router vendors in their products (e.g. Cisco uses WRED [2]). The basic idea of the RED is to detect the congestion by inspecting the average queue length at the routers. Before really occurrence of the congestion take place, RED drops data packets with increasing probability when the average queue length lies between two thresholds (min$_{th}$, max$_{th}$) in order to inform sources of coming congestion. As a consequence, when a TCP source finds out such preventive drops, it reduces the sending rate according to the additive increase multiplicative decrease (AIMD) algorithm inherent to the TCP protocol. As has been addressed in [6, 8, 11], one of RED’s main problems is that the average queue size varies depending on the parameter settings. On the other hand, network operators would naturally like to have a rough a priori estimation of the average delays in their congested routers. To achieve such predictable average delays, RED’s parameters must be adjusted dynamically based on the current traffic conditions. A second, related weakness of RED is that the throughput is also sensitive to the traffic load and to RED parameters. In particular, RED often does not operate well when the average queue becomes larger than max$_p$, resulting in significantly decreased throughput and increased dropping rates. To cope with this problem it is again needed to dynamically tune of RED's parameters base on the network conditions. Toward this idea this paper is going to keep RED's queue size around a target point by dynamic tuning of it's max$_p$. It proposes a neuro-fuzzy based learning procedure that learns the network behavior against max$_p$ variations and then adjusts max$_p$ in such a way that the network is directed to a target point in which queue size is kept around a target queue size. The result is a network with a predictable delay.

Random Early Detection

Random Early Detection algorithm was proposed by Floyd and Jacobson [3] in 1993. Figs. 1 and 2 show the algorithm and drop function of RED. A router implementing RED accepts all packets until the queue reaches minth, after which it drops a packet with a linear probability distribution function. When the queue length reaches max$_{th}$, all packets are dropped with a probability of one.

The RED algorithm, , includes two computational parts: computation of the average queue length and calculation of the drop probability.

```
for each packet arrival
    calculate the average queue size avg
    if $min_{th} \le avg < max_{th}$
        calculate probability $p_a$
        with probability $p_a$:
            mark the arriving packet
    else if $max_{th} \le avg$
        mark the arriving packet
```

Fig. 1. General algorithm for RED gateways

The RED algorithm involves four parameters to regulate its performance. min$_{th}$ and max$_{th}$ are the queue thresholds to perform packet drop, max$_p$ is the packet drop probability at max$_{th}$, and w is the weight parameter to calculate the average queue size from the instantaneous queue length. By making the packet drop probability a function of the level of congestion, RED gateway has a low packet-drop probability during low congestion, while the drop probability increases as the congestion level increases. The packet drop probability of RED is small in the interval min$_{th}$ and max$_{th}$. Moreover, the packets to be dropped are chosen randomly from the arriving packets from different hosts. The performance of RED significantly depends on the values of its four parameters, max$_p$, min$_{th}$, max$_{th}$, and w.

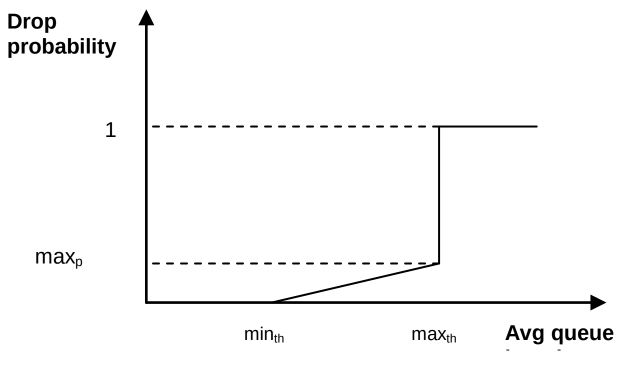

Fig. 2. RED gateway drop function

This paper proposes an algorithm called LRED (Learning RED) that adjusts max$_p$ dynamically to direct the network to the target operational point. The applied adjustment rules is drawn based on a learning process that predicts the network reaction against various amounts of changes on max$_p$ value. This learning method is introduced in the next section.

Learning Method: Locally Linear Neuro-Fuzzy

The base of Locally Linear Neuro-Fuzzy (LLNF) model is dividing the input space to small linear subspaces with fuzzy validity functions. Any produced linear part with its validity function can be considered as a fuzzy neuron. Thus this model is a neuro-fuzzy network with one hidden layer, and a linear neuron in the output layer which simply calculates the weighted sum of the outputs of locally linear neurons as illustrated by equations (1)-(2):

(1) $$\hat{y}_i = w_{i0} + w_{i1}u_1 + w_{i2}u_2 + \ldots + w_{ip}u_p$$

(2) $$\hat{y} = \sum_{i=1}^{M} \hat{y}_i \varphi_i(\underline{u})$$

This structure is depicted in Fig. 3, where $\underline{u} = [u_1\ u_2\ \ldots\ u_p]^T$ is the model input, M is the number of LLM neurons, and w_{ij} denotes the LLM parameters of the ith neuron. The validity functions are chosen as normalized Gaussians as shown in equations (3)-(4); it's necessary for a proper interpretation of validity functions [1, 4, 12].

(3) $$\phi_i(\underline{u}) = \frac{\mu_i(\underline{u})}{\sum_{j=1}^{M} \mu_j(\underline{u})}$$

(4) $$\mu_i(\underline{u}) = \exp\left(-\frac{1}{2}\left(\frac{(u_1 - c_{i1})^2}{\sigma_{i1}^2} + \ldots + \frac{(u_p - c_{ip})^2}{\sigma_{ip}^2}\right)\right) = \exp\left(-\frac{1}{2}\frac{(u_1 - c_{i1})^2}{\sigma_{i1}^2}\right) \times \ldots \times \exp\left(-\frac{1}{2}\frac{(u_p - c_{ip})^2}{\sigma_{ip}^2}\right)$$

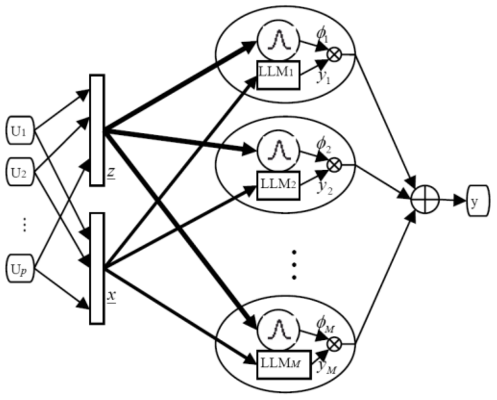

Fig. 3. Structure of locally linear neuro-fuzzy model

The $M \times p$ parameters of the nonlinear hidden layer are the parameters of Gaussian validity functions: center (c_{ij}) and standard deviation (σ_{ij}). Optimization or learning methods are used to adjust the two sets of parameters, the rule consequent parameters of the locally linear models (ω_{ij} s) and the rule premise parameters of validity functions (c_{ij} s and s). Global optimization of linear consequent parameters is simply achieved by least squares technique. Container of global parameter vector is $M \times (p+1)$ elements as in equation (5).

(5) $$\underline{\omega} = \begin{bmatrix} \omega_{10} & \omega_{11} & \cdots & \omega_{1p} & \omega_{20} & \omega_{21} & & \\ & & & \cdots & \omega_{M0} & \cdots & \omega_{Mp} \end{bmatrix}^T$$

The associated regression matrix $\underline{X}$ for N measured data samples are represented by equations (6)-(7).

(6) $$\underline{X} = [\underline{X}_1\ \ \underline{X}_2\ \ \ldots\ \ \underline{X}_M]$$

(7) $$\underline{X}_i = \begin{bmatrix} \phi_i(\underline{u}(1)) & \cdots & u_p(1)\phi_i(\underline{u}(1)) \\ \phi_i(\underline{u}(2)) & \cdots & u_p(2)\phi_i(\underline{u}(2)) \\ \vdots & & \vdots \\ \phi_i(\underline{u}(N)) & \cdots & u_p(N)\phi_i(\underline{u}(N)) \end{bmatrix}$$

Therefore we will have:

(8) $$\underline{\hat{y}} = \underline{X}.\underline{\hat{\omega}} \quad ; \quad \underline{\hat{\omega}} = \left(\underline{X}^T \underline{X}\right)^{-1} \underline{X}^T \underline{y}$$

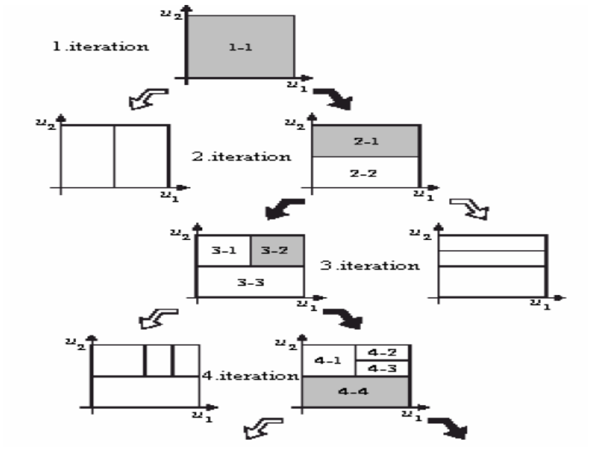

Fig. 4. An example Operation of the LOLIMOT algorithm

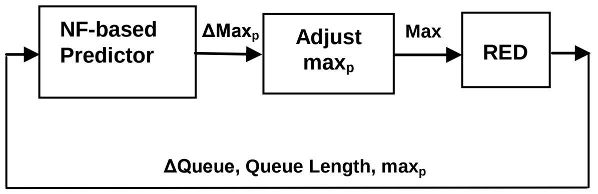

Fig. 5. Active Queue Management based on neuro-fuzzy prediction

An incremental tree based learning algorithm is appropriate for tuning the rule premise parameters [7], i.e. determining the validation hypercube for each locally linear model. In each iteration the worst performing locally linear neuron is determined to be divided. All the possible divisions in the p dimensional input space are checked and the best is chosen. The splitting ratio can be simply adjusted as $1/2$, which means that the locally linear neuron is divided into two equal halves on the selected input dimension. Based on such a division the centers (c_{ij}) and standard deviations (σ_{ij}) of the new neurons are computed and the fuzzy validity functions for the new structure are updated according to the equations (10) and (11). The center of validity functions are the centers of the new hyper-cubes, and the standard deviations are usually set as 0.7 as discussed in [5]. The algorithm can be found in [9] but five iterations of the procedure for an application with two dimensional input space is depicted in Fig. 4. This learning algorithm as automatic provides the best linear or nonlinear model with maximum generalization, and performs well in prediction applications [5, 9]. The error index used in the experiments of this study is Normalized Mean Square Error (NMSE), which is defined as in (9).

$$NMSE = \left(\frac{\sum_{i=1}^{n} (y - \hat{y})^2}{\sum_{i=1}^{n} (y - \bar{y})^2} \right) \tag{9}$$

where y, $\hat{y}$, and $\bar{y}$ are observed data, predicted data and average of the observed data respectively.

Proposed Algorithm: The Learning RED Algorithm

As said above, in this paper the goal is to design an AQM scheme that keeps the queue length robustly in a predetermined target region. For this purpose we design a neuro-fuzzy based intelligent controller that first learns how the network behaves against maxp variations and then uses this learned knowledge to direct the network toward the desired operational point. As shown in Fig. 5 to design a more accurately learning procedure, not only maxp is considered as an input parameter, but also we consider the current queue length and ΔQueue as other inputs of the neuro-fuzzy network. In the other words, the designed controller learns that in RED algorithm for a given queue length and maxp which adjustment of maxp (i.e. Δmaxp) can change the queue size with the amount of ΔQueue. Then it simply applies an appropriate Δmaxp to direct the queue size to the target range.

Packet Level Simulation

In order to evaluate the proposed algorithm, we implement it as an extension to RED module of ns-2 simulator [10]. We present a group of simulation results to demonstrate the validity of our design. We demonstrate through extensive simulations that LRED can learn and stabilize the queue behavior in the router. Our simulation uses the reference dumbbell topology, shown in Fig. 6. This network has a single bottleneck link with bandwidth of 3 Mbps, shared by 30 identical and long-lived TCP/Reno flows. All non-bottleneck links save bandwidth of 100 Mbps and execute drop-tail queue management algorithm. The propagation delay from each source to its corresponding edge router is 10 ms and the delay between the two routers is 30 ms; hence, round trip time (RTT) is 100 ms for all connections. The buffer size is set to 100 packets in each router. The bottleneck capacity is 3 Mbps and all other links have bandwidth of 100 Mbps. The basic parameters of RED are selected as follows: min_{th}=15 packets, max_{th}=75 packets, max_p=0.01, w_q =0.002, interval time = 0.5 second

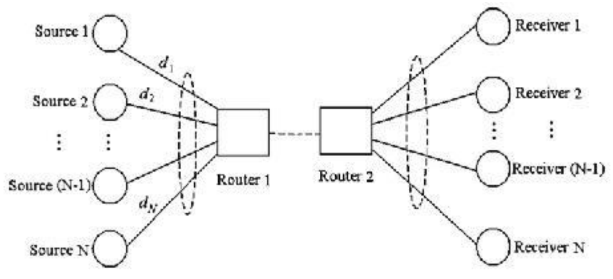

Fig. 6. The simulation network topology

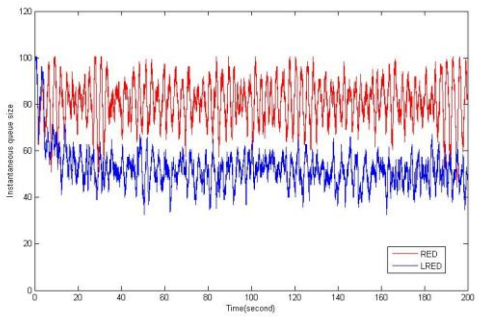

Fig. 7. Instantaneous queue length of LRED revolves around the target size i.e. 50 packets

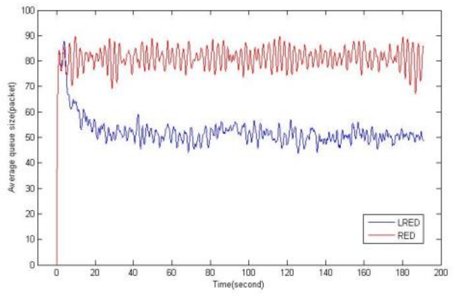

Fig. 8. Average queue length for RED and LRED

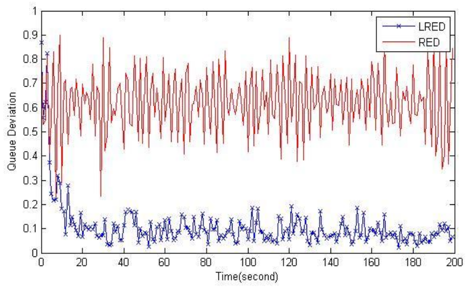

Fig. 9. QD comparison

As an example this experiment aims to fix the queue size around 50 packets i.e. the target queue length=50 packets. In general this target queue size is set based on the maximum delay that is acceptable for the applications.

Figs. 7-8 shows the simulation results. In Fig. 7 you can see how instantaneous queue sizes of RED and LRED evolve during the simulation time. According to this figure while RED's queue size remains almost always over max_{th} (75 packets in this simulation), LRED's queue size revolves around the target queue length (50 packets in this simulation). Success of the proposed AQM algorithm in stabilizing the queue size around the target queue length can be better seen in the Fig. 8. According to this figure LRED's average queue size converges to the target value. As another important issue it can be found in this figure that average queue size of LRED follows a more stable manner and has a minimum level of fluctuations around the target queue size. This stable behavior leads to a lower level of jitter in compare with the RED algorithm.

To quantify the stability of various AQM schemes, we introduce Queue Deviation (QD) that is computed by using equation (10).

(10) $$QD = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{q_{len}(i)}{q_{ref}} - 1\right|$$

In this equation $q_{len}(i)$ is ith sample of queue length, n is the number of queue length samples and q_{ref} is the target queue length that the designer aims to stabilize queue length around it. Obviously, lower values of QD refer to low level of deviation and equivalently high degree of stability. For a stable AQM in which queue length always remains around qref, QD approaches to zero. Fig. 9 shows QD values for two algorithms. According to this figures, LRED has the lowest value for QD and hence offers the most stable queue management scheme.

These simulation results show that LRED has a pre-determined queue size and is stable around its target size. This means that in a network whose queue is managed by LRED the queuing delay is predictable (as function of pre-determined queue size) and has negligible jitter. Hence, LRED is a good option for the networks that support real time applications such as steaming media.

These simulation results show that the learning algorithm can be an efficient approach to design effective AQM algorithms. Due to extreme complexity of AQM algorithms it is a very difficult and even impractical to develop mathematical or other types of analytical models to study how AQM's parameters affect its behavior. As we saw in this paper learning is a powerful alternative that extracts the nonlinear relation of inputs and outputs of a AQM algorithm. Although we applied this idea to study about impact of maxp parameter on RED's performance, it can be easily applied to study about impact of other parameters as well. Also this approach can be employed in other fields of computer networks.

Conclusion

This paper proposed a prediction-based active queue management algorithm that keeps queue size around a target value that causes to roughly constant queuing delay in the network. The proposed controller uses LoLiMoT algorithm as learning method of the network behavior, learns network's behavior against max_p variations and then tunes RED's max_p regarding the target queue size. Simulation results showed that this AQM scheme controls the queue size successfully and stably around the target queue size and hence controls robustly delay and jitter in the network.

Acknowledgements

This research was supported by a research grant from the INSF (Iran National Science Foundation). We express our thanks to Dr. Morteza Analoui who checked our manuscript.

REFERENCES

[1] Arani E., Lucas C., Araabi B. N., “WLoLiMoT: A Wavelete and LoLiMoT Based Algorithm for Time Series Prediction”, Integrated Systems, Design and Technology, 2010.
[2] Cisco Systems, Congestion Avoidance Overview, Available from: http://www.cisco.com/univercd/cc/td/doc/product/software/ios120/12cgcr/qos_c/qcpart3.
[3] Floyd, V. Jacobson, “Random early detection gateways for congestion avoidance”, IEEE/ACM Transactions Networking, 1993.
[4] Gholipour A., Araabi B. N., Lucas C., “Predicting Chaotic Time Series Using Neural and Neurofuzzy Models: A Comparative Study”, neural processing letters, volume 24, number 3, 2009.
[5] Gholipour A., Lucas C., Araabi B. N., Mirmomeni M., and Shafiee M., “Extracting the main patterns of natural time series for longterm neuro fuzzy prediction,” Neural Computing and Applications, 2006.
[6] May M., Bolot J., Diot C., Lyles B., “Reasons Not to Deploy RED”, 7th. International Workshop on Quality of Service, 1999.
[7] Mirmomeni M., Lucas C., Moshiri B., Araabi B. N., “Introducing adaptive neurofuzzy modeling with online learning method for prediction of time-varying solar and geomagnetic activity indices”, Expert systems with applications, Volume 37, Issue 12, 2010.
[8] Misra V., Gong W. Bo, Towsley D. F., “Fluid-based Analysis of a Network of AQM Routers Supporting TCP Flows with an Application to RED”, SIGCOMM, 2000.
[9] Nelles O., Nonlinear system identification, Springer Verlag press, 2001.
[10] Network Simulator-ns2, http://-mash.cs.berkelay.edu/ns.
[11] Ott T., Lakshman T., Wong L., SRED: Stabilized RED, Infocom, 1999.
[12] Pedram A., Jamali M. R., Pedram T., Fakhraie S. M., Lucas C., “Local Linear Model Tree (LOLIMOT) Reconfigurable Parallel Hardware”, World Academy of Science, Engineering and Technology, 2006.

Authors: Shahram Jamali, Departmebt of Computer Engineering, University of Mohaghegh Ardabili, Ardabil, Iran, E-mail:Jamali@iust.ac.ir;
Mehdi Nosrati, Depatment of computer engineering, Payame Noor University, I.R. of Iran, E-mail:nosrati@pnu.ac.ir;
Seyed Naser Seyed Hashemi, Young Researcher Club, Ardabil Branch,. Islamic Azad University, Ardabil, Iran, E-mail: n.s.hashemi@qiau.ac.ir.

The correspondence address is:
e-mail: jamali@iust.ac.ir