Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2011, Article ID 872347, 17 pages
doi:10.1155/2011/872347

Research Article

Self-Tuning Random Early Detection Algorithm to Improve Performance of Network Transmission

Jianyong Chen, Cunying Hu, and Zhen Ji

Shenzhen City Key Laboratory of Embedded System Design, College of Computer Science and Software Engineering, Shenzhen University, Shenzhen 518060, China

Correspondence should be addressed to Jianyong Chen, cjyok2000@hotmail.com

Received 22 August 2010; Accepted 26 September 2010

Academic Editor: Ming Li

We use a discrete-time dynamical feedback system model of TCP/RED to study the performance of Random Early Detection (RED) for different values of control parameters. Our analysis shows that the queue length is able to keep stable at a given target if the maximum probability $p_{\max}$ and exponential averaging weight w satisfy some conditions. From the mathematical analysis, a new self-tuning RED is proposed to improve the performance of TCP-RED network. The appropriate $p_{\max}$ is dynamically obtained according to history information of both $p_{\max}$ and the average queue size in a period of time. And w is properly chosen according to a linear stability condition of the average queue length. From simulations with *ns*-2, it is found that the self-tuning RED is more robust to stabilize queue length in terms of less deviation from the target and smaller fluctuation amplitude, compared to adaptive RED, Random Early Marking (REM), and Proportional-Integral (PI) controller.

1. Introduction

In computer network, congestion control has been a serious problem since the traffic always appears complex nonlinear phenomena [1–3]. Especially, since multimedia communication becomes more and more popular [4, 5], lack of effective management of congestion significantly affects the performance of the network system, such as degradation of link utilization, more round-trip time, and even makes a network inaccessible. At the same time, various approaches have been proposed to solve the issue. These schemes can be divided into two categories. One category strengthens congestion management at the sources while making few changes to internet [6, 7]. The other one is to adapt Active Queue Management (AQM) at each router, and eventually control congestion level, such as Random Early Detection (RED) [8], Random Early Marking (REM) [9, 10], and Adaptive Virtual Queue (AVQ) [11]. Among them, RED is the most prominent and well-studied AQM scheme. Also, RED is used in wireless network to increase TCP throughput [12]. Although RED can prevent global synchronization, reduce packet loss, and achieve high throughput, its performance of both QoS and security is suspected by many researcher [13–17]. In particular, it is difficult to parameterize RED to achieve good performance in various networks.

The main goal of AQM is to keep queue length stable while making a good tradeoff between high throughput and low delay. As a queue management policy, RED gateway manages queue by monitoring the average queue size q_{ave}. When q_{ave} exceeds the preset lower threshold $q_{\min}$, the arriving packets are randomly dropped or marked with a certain probability. In that case, some connections can sense the early congestion, and the window sizes are changed to avoid serious congestion. Once the average queue size is larger than the upper threshold $q_{\max}$, the RED gateway drops/marks every arriving packet to keep q_{ave} below $q_{\max}$. Although the RED algorithm is an effective mechanism to achieve high throughput and low delay, some researchers note that RED is very sensitive to traffic load and control parameters [13–16]. In particular, when q_{ave} becomes larger than $q_{\max}$, lots of packets are dropped and the throughput of the congested link decreases seriously. Many modified RED algorithms are proposed to improve its robustness by optimizing either drop probability function or control parameters.

Among these algorithms, one approach is to modify the packet drop probability function to improve the performances of RED and its robustness. In [18], Hollot et al. propose a PI controller (Proportional-Integral controller) to response quickly to the change of TCP/RED dynamics, and some research also study new adaptive schemes to improve the performance of PI controller in recent years [19, 20]. Besides that, SPI-RED (Self-tuning PI-RED) and NPD-RED (autonomous Proportional and Differential RED algorithm) are developed based on the thought of PI controller in [21, 22]. In [23, 24], the authors analyze a decentralized network congestion control algorithm and its local stability and propose E-RED to get high link utilization in which the packet drop rate increases exponentially with q_{ave}. In [25], LRED (Loss Ratio-based RED) is proposed, which takes the packet loss rate and queue length into account to compute the dropping probability. In [26], some appropriate tools from feedback control theory are used to design a novel AQM with the delay information of all the links. Also, some studies attempt to use neural network for early congestion prediction [27, 28]. Although the performances of these modified algorithms are much better than the original RED with higher throughput and lower queuing delay, they always induce additional parameters that are needed to be optimized to accommodate different network scenarios.

Another approach does not change the basic idea of RED, but tunes the control parameters to improve its robustness. Feng et al. have argued that there is no single set of RED parameters which can work well under various congestion. In [14], original ARED is proposed, which merely tunes the parameter $p_{\max}$ based on network traffic to keep q_{ave} in the expected range $[q_{\min}, q_{\max}]$. To get better performance, Floyd et al. present a revised version, also called Adaptive RED [15], which adopts AIMD mechanism (Additive-Increase-Multiplicative-Decrease) to changes $p_{\max}$ slowly, ensuring q_{ave} in a smaller target range between $q_{\min}$ and $q_{\max}$. At the same time, they also provide some insights to set other control parameters. In [29], SARED (Stabilized ARED) is proposed to increases the effect of instant queue on average queue size by using a larger queue weight when the queue size is out of the target area. By this way, SARED reacts much faster to queue changes which further improves the robustness and throughput. In [30], an improved ARED is developed to enhance the robustness of RED by adapting more proper range of $p_{\max}$, and the weight w is also adjusted according to a linear stability condition. Besides that, Sun et al. proposed PD-RED (Proportional Derivative controller) to improve the performance of RED [31, 32]. However, PD-RED does not consider the past history information. It is sensitive to the short-lived and non-TCP traffics. In [33, 34], based on TCP channel model and traffic characteristics, a framework is developed to get the bounds of $p_{\max}$ and the optimal weight, respectively. In [35], the authors propose an AP-RED (Autoparameterization RED) based on a fluid-flow model which stabilizes the instantaneous queue size under various networks by adjusting $q_{\min}$, $q_{\max}$, $p_{\max}$, and w. In [36–38], the authors also attempted to adjust $p_{\max}$ to achieve better performance in some network conditions. In recent years, other control parameters except $p_{\max}$ and w are also studied to improve the performance of RED [39]. Simulations demonstrated that these algorithms can maintain the queue length at the target in specific network scenarios by tuning control parameters.

Although RED is sensitive to control parameters and network parameters, few of the above reports concern its robustness affected from the control parameters. The setting of parameters is still a problem in applications. In this paper, we analyze the performance of RED with different values of control parameters for a given target and network scenario. Based on the analysis, an improved algorithm, named self-tuning RED, is proposed whose control parameters $p_{\max}$ is dynamically adjusted and the weight w is properly chosen according to a linear stability condition [16]. A set of experiments demonstrate that the self-tuning RED can keep queue length stable at the target value with smaller standard deviation and less deviation from the target.

The rest of the paper is organized as follows. In Section 2, we describe a network topology and then introduce the gentle RED algorithm and a discrete-time TCP-RED model. In Section 3, based on the discrete-time network model, we analyze the impacts of the control parameters on queue length and discuss the proper values of control parameters in detail when the queue length stabilizes at a given target. In Section 4, a self-tuning RED algorithm is described, and the setting of parameters is also discussed. Section 5 presents simulations results and compares the self-tuning RED with various AQM schemes.

2. Discrete-Time Feedback Model for TCP-RED

In this paper, the network topology used is a dumbbell with a single bottleneck as shown in Figure 1. There are N connections sharing a single link with the capacity C. It is assumed that these connections are identical, long-lived TCP Reno connections and have the same round-trip propagation delay d. The main purpose is to stabilize the nonlinear traffic and thus improve the throughput [40, 41].

2.1. Random Early Detection

RED is a recommended scheme of AQM to avoid network congestion by the Internet Engineering Task Force (IETF) [8]. In order to better control network congestion, the RED gateway always measures its average queue size q_{ave}, and drops/marks the arriving packets with the probability p to notify TCP ends of the incipient congestion when q_{ave} exceeds the expected lower threshold $q_{\min}$. When q_{ave} exceeds the expected upper threshold $q_{\max}$, RED gateway drops every arriving packet, so that it can control the average queue size within the expected range $[q_{\min}, q_{\max}]$, even in the absence of cooperating sources.

Gentle RED is a revised version of RED proposed by Floyd [42]. The main difference of the two versions lies in the value of the drop probability p when q_{ave} varies from $q_{\max}$ to double $q_{\max}$. In the case, the original RED sets p to 1, while Gentle RED increases p linearly from $p_{\max}$ to 1. By doing so, Gentle RED prevents the system from large oscillation due to

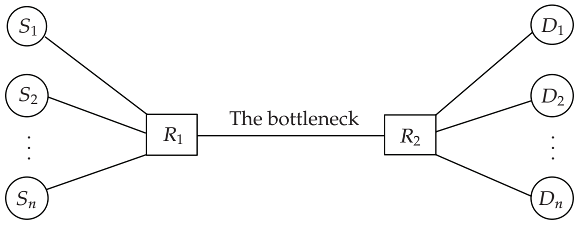


Figure 1: Structure of the network.

dropping a lot of packets suddenly. In the study, Gentle RED is used, which can be described by the following piecewise function

$$p = \begin{cases} 0 & q_{\text{ave}} \le q_{\min}, \\ 1 & q_{\text{ave}} \ge 2q_{\max}, \\ \dfrac{q_{\text{ave}} - q_{\min}}{q_{\max} - q_{\min}} p_{\max} & q_{\min} < q_{\text{ave}} < q_{\max}, \\ 2p_{\max} - 1 + \dfrac{1 - p_{\max}}{q_{\max}} q_{\text{ave}} & q_{\max} < q_{\text{ave}} < 2q_{\max}. \end{cases} \tag{2.1}$$

Here, $p_{\max}$ is the drop probability when q_{ave} is equal to $q_{\max}$ and is also called the maximum drop probability. $q_{\max}$ and $q_{\min}$ are the expected upper and lower thresholds of q_{ave}, respectively. At the time of packet arrival, q_{ave} is updated with [8]

$$q_{\text{ave}} = (1 - w)q'_{\text{ave}} + wq. \tag{2.2}$$

Here, q'_{ave} is average queue size at the previous time. q is instantaneous queue length. w is the exponential averaging weight which is limited in the range $[0, 1]$.

2.2. Discrete-Time Feedback Model for TCP-RED

To better understand and analyze the nonlinear interaction between RED and TCP, some research propose different network model to analyze the property of real network [16, 43, 44]. In the paper, a discrete-time dynamical feedback TCP-RED model developed by Ranjan et al. is used [16]. It is assumed that ECN (Explicit Congestion Notification) mechanism is implemented at the RED gateway to provide the ends with congestion signals. According to the model, q can be determined by some network parameters, such as the buffer size B, mean packet size M, the number of connections N and a constant K within the range $[1, (8/3)^{0.5}]$. q_{ave} can be expressed mathematically as [16]

$$q_{\text{ave}} = f\left(q'_{\text{ave}}\right) \begin{cases} (1 - w)q'_{\text{ave}}, & q_{\text{ave}} \ge q1, \\ (1 - w)q'_{\text{ave}} + w \cdot B, & q_{\text{ave}} \le q2, \\ (1 - w)q'_{\text{ave}} + w \cdot \left(\dfrac{NK}{p} - \dfrac{Cd}{M}\right), & \text{otherwise.} \end{cases} \tag{2.3}$$

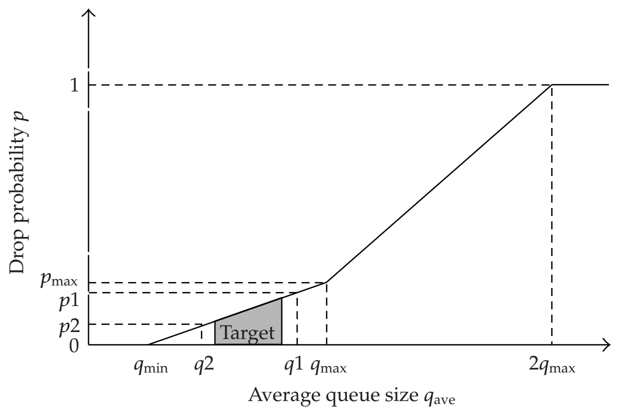


Figure 2: Gentle RED algorithm.

Here, $q1 = p1(q_{\max} - q_{\min})/p_{\max} + q_{\min}$, where $p1 = (NMK/(dC))^2$ is the smallest drop rate that leads to an empty queue at next time. $q2 = p2(q_{\max} - q_{\min})/p_{\max} + q_{\min}$, where $p2 = (NMK/(dC + BM))^2$ is the biggest drop probability with full queue B at next time. C is the capacity of the shared link, and d is round-trip propagation delay. Based on the discrete-time model, the drop probability of Gentle RED algorithm is shown in Figure 2.

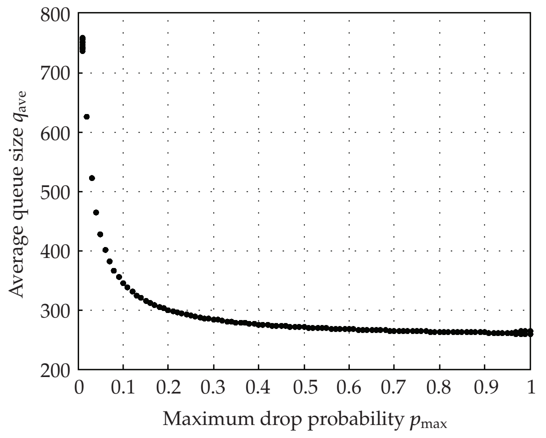
In the discrete-time model, it is noted that q_{ave} is updated over time scale of approximately one RTT (Round-Trip Time) [16, 45], which is much longer than typical interarrival times of packets in practice. Therefore, w in the model is much larger than the value in *ns*-2 simulations. Given that w_{red} is the exponential averaging weight at a RED gateway, and that n is the number of packets transmitted over the link in one RTT, $w \approx 1 - (1 - w_{\text{red}})^n \approx n \cdot w_{\text{red}}$ in the model.

3. Control Parameters of RED

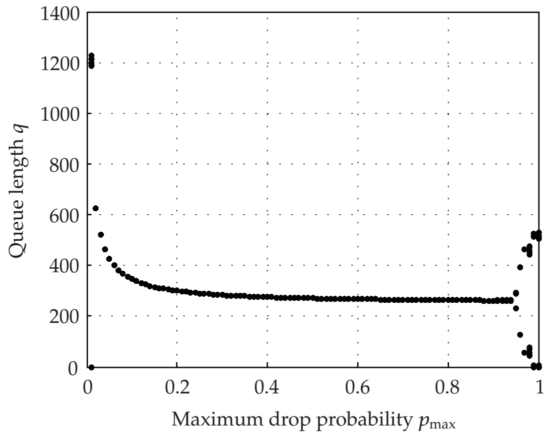
In the section, we discuss the effect of $p_{\max}$ on the queue length and analyze the value of $p_{\max}$ when q converges to the given target. Then, the effect of w on performance is also studied. Finally, according to the linear stability condition, the maximum value of w that keeps q stable is obtained.

3.1. Maximum Drop Probability $p_{\max}$

An important objective of AQM is to stabilize the queue length at a given target so that the network performance can be improved. First, the throughput of network can be improved. Without stable queue, the queue size may be too short, leading to waste of bandwidth, or become too long, which means serious congestion and more packets to be dropped. Second, stable queue size means stable end-to-end delay and better QoS. However, the control parameter $p_{\max}$ directly impacts the aggressiveness of RED [14]. If a small value of $p_{\max}$ is used in heavily-congested networks, then early detection is too conservative to provide sufficient congestion signals. As a result, q_{ave} oscillates around $q_{\max}$. By contrast, a large $p_{\max}$ in light congestion networks may lead to lots of packets lost, and q_{ave} may fluctuate around $q_{\min}$. Figure 3 presents the average queue size and queue length with RED when $p_{\max}$ is varied



(a)



(b)

Figure 3: (a) Average queue size and (b) queue length with respect to the maximum drop probability.

from 0.01 to 1.0. The result is in the agreement with the general results in [14]. In simulations, the parameters are the same as in [16]: $C = 75\,\text{Mb/s}$, $K = (3/2)^{0.5}$, $M = 4,000\,\text{b}$, $d = 0.1\,\text{s}$, $N = 250$, $q_{\max} = 750$ packets, $q_{\min} = 250$ packets, $B = 3750$ packets, and $w = 0.02$.

From Figure 3, we can see that different values of $p_{\max}$ result in both q_{ave} and q either oscillating or stabilizing at the same values. When $p_{\max}$ is set to 0.01, q_{ave} is very large and oscillates around $q_{\max} = 750$, and q oscillates more seriously. As $p_{\max}$ increases, both q_{ave} and q stabilize at the same value within $[q_{\min}, q_{\max}]$. We also find that when $p_{\max}$ is larger than 0.95, q_{ave} and q fluctuate with the same oscillation center $q_{\min} = 250$. To sum up the results, if $p_{\max}$ is set appropriately with other proper control parameters, it is possible to stabilize q_{ave} and q at a desired value within $[q_{\min}, q_{\max}]$. In other words, if q_{ave} is stable, it means that q is also stable. Therefore, only q_{ave} is necessary to be examined. In the following section, configuration of $p_{\max}$ is studied to get robust value of q_{ave} at a given target queue length.

Suppose that q_{ave} stabilizes at the point q^*_{ave}, the point satisfies the equation $q^*_{\text{ave}} = f(q^*_{\text{ave}})$ from (2.3). By substituting (2.1) into (2.3), we have [45]

$$\frac{CM(q_{\max} - q_{\min})}{r^{1.5}}\sqrt{p_{\max}} + \left(CMq_{\min} + dC^2\right)\left(\frac{p_{\max}}{r}\right)^{0.5} - NMKC = 0. \tag{3.1}$$

Here, $r = (q_{\max} - q_{\min})/(q^*_{\text{ave}} - q_{\min})$. $p_{\max}$ can be obtained from above equation,

$$p_{\max} = \frac{r(NMK)^2}{\left(Mq^*_{\text{ave}} + dC\right)^2}. \tag{3.2}$$

From (3.2), one can see that $p_{\max}$ not only depends on some fixed network parameters, such as the number of connections N, mean packet size M, and the round-trip propagation delay d, but also depends on the value of the fixed point q^*_{ave}. In other words, when network parameters are unchanged, there exists a unique value of $p_{\max}$ to keep q_{ave} at the given expected value. Suppose that the target queue length q_{target} is the fixed point, $p_{\max}$ that yields to $q_{\text{ave}} = q_{\text{target}}$ can be configured by (3.2). In that case, $r = (q_{\max} - q_{\min})/(q_{\text{target}} - q_{\min})$.

3.2. The Properties of $p_{\max}$

In (3.2), there is a constant K, which relies on network scenarios used in practical application [43], such as TCP version, ACK ways. Therefore, it is necessary to get the value of K before $p_{\max}$ is configured. In the discrete-time model, given RTT with R and drop probability with p, the throughput of a TCP Reno connection can be simply expressed as [16]

$$T(p, R) = \frac{MK}{\sqrt{p}R}. \tag{3.3}$$

By solving the above equation, the constant K can be obtained. However, the configuration of each user may be different, and K is different. To capture the whole network character, we use the mean throughput (the total throughput of the congested link/the number of connections ratio) to represent the throughput of each connection.

In order to smooth out burst traffic, time is firstly divided into consecutive sampling period with each of size δt, and then $q^{\sim}_{\text{ave}}$ is taken as the mean value of q_{ave} in δt. It can be regarded as the fixed point at which the queue length stabilizes in δt. If $q^{\sim}_{\text{ave}}$ is larger than 0, the link utilization is 100%. In the case, K can be calculated by

$$\frac{MK}{\sqrt{p}(d + q^*_{\text{ave}}M/C)} = \frac{C}{N}. \tag{3.4}$$

From the drop probability function (2.1), it is noted that the RED gateway drops every arriving packet if $q_{\text{ave}} > q_{\max}$, while it does not drop any one if $q_{\text{ave}} < q_{\min}$. Therefore, q_{ave} cannot be always above $q_{\max}$ or below $q_{\min}$. When a small $p_{\max}$ is adapted, q_{ave} oscillates around $q_{\max}$, but a large $p_{\max}$ leads q_{ave} oscillating around $q_{\min}$. Therefore, if other control parameters are proper except $p_{\max}$, $q^{\sim}_{\text{ave}} \in [q_{\min}, q_{\max}]$ at most time. Given that $p^*_{\max}$ is the maximum drop probability used in the last period δt, we have $p = p^*_{\max}(q_{\tilde{\text{ave}}} - q_{\min})/(q_{\max} - q_{\min})$ by (2.1), therefore K can be expressed as follows:

$$K = \frac{\sqrt{((q_{\tilde{\text{ave}}} - q_{\min})/(q_{\max} - q_{\min}))p^*_{\max}}(dC + Mq_{\tilde{\text{ave}}})}{NM}. \tag{3.5}$$

By substituting $q^*_{\text{ave}} = q_{\text{target}}$, $r = (q_{\max} - q_{\min})/(q_{\text{target}} - q_{\min})$ and (3.5) into (3.2), we have

$$p_{\max} = p^*_{\max}\frac{q_{\tilde{\text{ave}}} - q_{\min}}{q_{\text{target}} - q_{\min}}\left(\frac{d + Mq_{\tilde{\text{ave}}}/C}{Mq_{\text{target}}/C + d}\right). \tag{3.6}$$

From (3.6), it is found that $p_{\max}$ is independent of the load of the network (the number of connections N). This is a desirable property as a good AQM scheme since the number of connections accessing a link is out of network manager's control. In particular, when one packet needs to go through different sets of bottlenecks, the traffic load may vary dramatically. However, we can adjust $p_{\max}$ according to (3.6) as long as we monitor the variation of q_{ave} in recent δt and degrade the sensitivity to traffic load to some extent.

In order to simultaneously achieve low average queuing delay and high throughput, one rule of thumb is to require that the average queuing delay d_{target} is only a fraction of round-trip time R, [15]. Thus, the end-to-end delay is mainly caused by d, and d_{target} is also a fraction of d, that is, $d_{\text{target}} \ll d$. In order to ensure $q_{\text{ave}} \in [q_{\min}, q_{\max}]$, q_{target} is generally set to $(q_{\min} + q_{\max})/2$ in applications, that is, $r = 2$ [15]. Since $r = 2$, $d_{\text{target}} = q_{\text{target}} * M/C$ and $q_{\tilde{\text{ave}}} \in [q_{\min}, q_{\max}]$, we have

$$\frac{Mq_{\tilde{\text{ave}}}}{C} \ll \frac{2Mq_{\text{target}}}{C} \ll d. \tag{3.7}$$

According to (3.6), the maximum drop probability can be approximately rewritten as follows:

$$p_{\max} \approx p^*_{\max}\frac{q_{\tilde{\text{ave}}} - q_{\min}}{q_{\text{target}} - q_{\min}}. \tag{3.8}$$

As you can see that $p_{\max}$, to a large extent, is determined by the lower threshold $q_{\min}$, the expected queue length q_{target}, the prior network state $q_{\tilde{\text{ave}}}$ and $p^*_{\max}$. Therefore, $p_{\max}$ tends to be less sensitive to the round-trip propagation delay d and more stable for a large value of d. In other words, $p_{\max}$ determined by (3.6) is suitable to be used for a wider variety of scenarios.

3.3. *Exponential Averaging Weight* w

Based on the above analysis in Section 3.2, it is concluded that if other parameters are configured properly, the value of $p_{\max}$ given by (3.6) can yield to a stable queue length of q_{target}. Besides $p_{\max}$, there are also various system parameters (d, M, C) and control parameters $(q_{\min}, q_{\max}, q_{\text{target}}, w)$. Since system parameters are fixed once the network

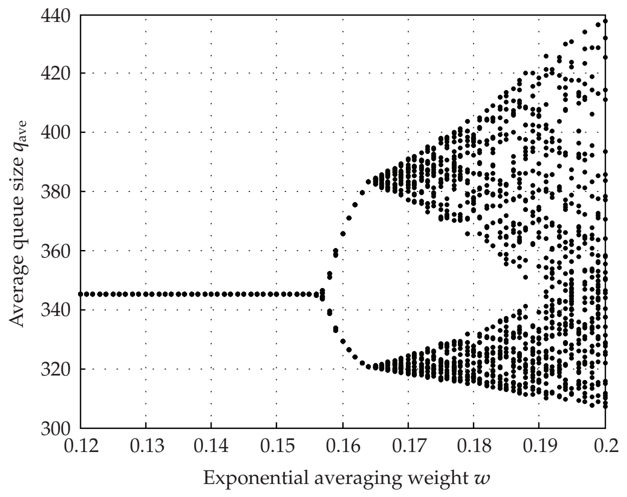


Figure 4: Bifurcation diagram of average queue size with respect to w.

topology is determined, network managers usually adjust control parameters to achieve stable queue length. The thresholds $(q_{\min}, q_{\max})$ and q_{target} refer to the tradeoffs between throughput and delay, which are constants in general and should be set in advance. To sum up, the weight w becomes the only parameters to be adjusted besides $p_{\max}$. As discussed in Section 3.1, stabilizing q at a target queue size can be obtained by maintaining q_{ave} at the point. By this means, we study the effect of w on q_{ave} in the following section.

The linear stability of a fixed point q^*_{ave} can be achieved, as long as the absolute value of the associated eigenvalue λ is not larger than 1 [16]

$$|\lambda| = \left| \left. \frac{\partial f(q'_{\text{ave}})}{\partial q'_{\text{ave}}} \right|_{q^*_{\text{ave}}} \right| = \left| 1 - w \frac{wNK}{2\sqrt{v}\left(q^*_{\text{ave}} - q_{\min}\right)^{3/2}} \right|. \tag{3.9}$$

Here, $v = p_{\max}/(q_{\max} - q_{\min})$ means the increase factor of p. Figure 4 plots the average queue size for the fixed point $q^*_{\text{ave}} = 345.1$ when w varies from 0.12 to 0.2. In simulations, we set $p_{\max} = 0.1$, and other parameters are the same as those described in Section 3.1.

For $w < 0.1578$, q_{ave} converges to the fixed point $q^*_{\text{ave}} = 345.1$. At the same time, $|\lambda|$ also stays below 1 that can be got from (3.9). As w increases, PDB (Period Doubling Bifurcation) emerges from the fixed point, and then chaos appears with $|\lambda| > 1$. It implies that if $|\lambda| > 1$, q_{ave} will be unstable. In other words, if q_{ave} is on the point, the system oscillates and even becomes chaotic. In the following section, we discuss the case $|\lambda| < 1$

Because all the parameters in (3.9) are positive, λ must be smaller than 1. The linear stability condition can be rewritten as $\lambda > -1$.

$$\lambda = 1 - w - \frac{wNK}{2\sqrt{v}\left(q^*_{\text{ave}} - q_{\min}\right)^{3/2}} > -1. \tag{3.10}$$

Now, we consider the case that q_{ave} stabilizes the target queue length q_{target}. By substituting $q^*_{\text{ave}} = q_{\text{target}}$, $r = (q_{\max} - q_{\min})/(q_{\text{target}} - q_{\min})$ and (3.2) into (3.10), we have

$$\begin{aligned}\lambda &= 1 - w - w\frac{NK}{2\sqrt{\left(r(NMK)^2/(Mq_{\text{target}} + dC)^2\right) \cdot (1/(q_{\max} - q_{\min})) \cdot ((q_{\max} - q_{\min})/r)^3}} \\ &= 1 - w - \frac{1}{2}wr\frac{Mq_{\text{target}} + dC}{M(q_{\max} - q_{\min})} > -1.\end{aligned} \tag{3.11}$$

Solving the above inequality with $r = (q_{\max} - q_{\min})/(q_{\text{target}} - q_{\min})$, we have

$$w < \frac{2}{1 + (1/2)r(Mq_{\text{target}} + dC)/M(q_{\max} - q_{\min})} = \frac{4}{3 + (q_{\min} + dC/M)/(q_{\text{target}} - q_{\min})}. \tag{3.12}$$

The inequality (3.12) shows that whether q_{ave} is stable at q_{target} or not is determined by control parameters $(q_{\min}, q_{\text{target}})$ as well as network system parameters (d, C, M). For example, it is evident that the maximum value of w is 0.1578 by (3.12) for $q_{\text{target}} = 345.1$ in the above simulation, which is consistent to the result of Figure 4. In applications, the exponential averaging weight at gateway can be approximately expressed as $w_{\text{red}} = w/n$, as discussed in Section 2.2.

It is noteworthy that the inequality (3.12) is the linear stability condition of q_{ave} rather than q. If w is set properly (not too small) by (3.12), q_{ave} can reflect the current network congestion level and the variation of q. In the case, stable average queue length at the target means stable queue length at the point, which is verified in Section 3.1. If w is too small, the effect of instant queue length on q_{ave} can be ignored, and q_{ave} cannot accurately represent the network congestion level. Therefore, the queue length may oscillate even though q_{ave} is stable. The inequality (3.12) provides the upper threshold of w, we still need further study the case with lower bound of the weight in future.

4. The Self-Tuning RED Algorithm

From the analysis in the previous sections, it is evident that that both $p_{\max}$ and w are key factors to make q_{ave} approach to q_{target}. Once the network sceneries (d, C, M), the control parameters $(q_{\min}, q_{\max})$ and the expected target queue length q_{target} are determined, the maximum value of w can be obtained by (3.12), while $p_{\max}$ can also be got indirectly by monitoring the performance of the network in (3.6). Based on the above conclusions, a new self-tuning RED can be developed to improve the robustness of RED, as shown in Algorithm 1.

There are two parameters δt and w to be configured in the self-tuning RED. The inequality (3.12) gives the upper threshold of w, but there is not a theoretical method to choose a better value. Simulations in Section 5.1 also show that the queue length keeps close to the target for all the given different values of w once (3.12) is satisfied. Considering that RED is used in congestion control, it is necessary to respond fast to network congestion,

```
set $w$ by the inequality(3.12)
every $q_{\text{ave}}$ update
    Sum = sum + $q_{\text{ave}}$;
    $n = n + 1$;
    If now > $L$ time + $\delta t$
    $\tilde{q_{\text{ave}}}$ = sum/$n$;
    update $p_{\max}$ by formula (3.6)
    sum = 0;
    $n = 0$;
    $L$ time = now;
Parameters:
now: current time
$L$ time: the last time $p_{\max}$ was calculated
$\delta t$: sampling period
sum: the sum of $q_{\text{ave}}$ in the period of $\delta t$
$n$: the total number of $q_{\text{ave}}$ sampled during $\delta t$
```

Algorithm 1: Self-tuning RED algorithm.

especially, in heavy congestion. In addition, with a large value of w, the average queue size is more sensitive to the change of queue length. Therefore, the value of w should be large enough to response to the variation of the queue length.

δt is related to the accurate estimation of K and $\tilde{q_{\text{ave}}}$. If δt is too short, the estimation of q^*_{ave} may be affected by temporary burst traffic or instability of queue length. However, if δt is set too long, it will take a long time to respond to network congestion. In our improved algorithm, it is reasonable to observe q_{ave} and $p_{\max}$ for several RTTs. There are two reasons: (1) the connections need more time than one RTT to react effectively to the current congestion level, so the system needs at least such long time to become stable; (2) if $p^*_{\max}$ is adjusted slowly and infrequently, the queue may be more stable that has been verified in Adaptive RED [15]. In the following simulations, we set $\delta t \approx 10RTT$s.

There are some parameters ($q_{\min}, q_{\text{target}}, d, C$, and M) in (3.6) and (3.12). In application, the control parameters ($q_{\min}$, q_{target}) are configured according to the expected queuing delay and the capacity of the link [15]. The link capacity C and mean packet size M is known for RED gateway. The round-trip propagation delay d may be difficult to know. However, d can be replaced by RTT in (3.12), since the linear stability condition is also satisfied. For the maximum drop probability, d has less effect on it as discussed in Section 3.2. Therefore, we do not need the accurate value of d, and we can estimate the value of d based on statistics. In the following section, we randomly generate propagation delay of the links except the shared one, and adapt approximate value $d = 150$ ms to simulate. The results show that the self-tuning RED can performs well with the estimated value.

5. Simulations

In this section, we evaluate the performance of the proposed RED algorithm by a number of simulations using *ns*-2 simulator. Other RED variants such as REM, Adaptive RED, and PI-controller are used to compare with the self-tuning RED. The network topology used in simulations is a dumbbell topology with a single bottleneck link and many identical, long-lived TCP Reno flows, as shown in Figure 1. The parameters used are the same as those in

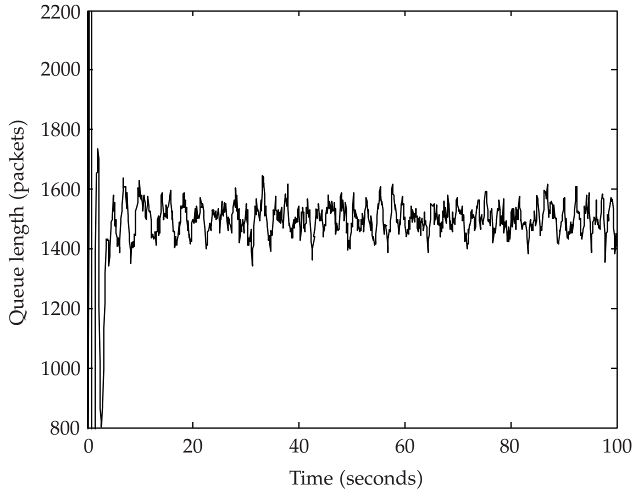


Figure 5: Instantaneous queue length ($N = 200$).

[16]: the capacity and delay of the common link are set to 150 Mb/s and 30 ms, respectively. Other links have a capacity of 30 Mb/s, and the propagation delays are randomly selected from [10 ms,35 ms]. Given these parameters, the average round-trip propagation delay is about 150 ms. All sources are ECN-capable, and mean packet size is 500 bytes. The buffer size B between R1 and R2 is set to 7500 packets (about bandwidth-delay product). The window size of each TCP connection is 1125 (the bandwidth-delay product of the link). Therefore, the maximum window size will not restrict the sending rate of packet in all scenarios, and it can trigger congestion even with small connections.

The basic parameters of adaptive RED and the self-tuning RED are set at $q_{\min} = 750$ packets, $q_{\max} = 2250$ packets, and $p_{\max} = 0.01$. For adaptive RED, the parameters are set the same as [15]: $\alpha = 0.01$, $\beta = 0.9$, $w = 0.000014305$ and interval $= 2$ s. For PI controller, PI coefficients a and b are 1.822×10^{-5} and 1.816×10^{-5}, respectively [18]. For REM, $\varphi = 1.001$, $\alpha = 0.1$, $\gamma = 0.001$, and the sampling interval is 2 ms. For the self-tuning RED, $\delta t = 2$ s and $w = 0.00048814$. In the following simulations, the target queue size is 1500 packets. The simulations run for 100 s, and the results are recorded every 0.1 s.

5.1. Performance under Extreme Conditions

In this experiment, we test the performance of the self-tuning RED for a constant number of TCP connections under two extreme cases: (1) light congestion with $N = 200$, (2) heavy congestion with $N = 1000$.

In order to clearly present the variation of the instantaneous queue length, the range of y-axis is reduced to [750, 2250]. At the same time, the range of x-axis is also reduced. Figures 5 and 6 demonstrate the dynamic change of the instantaneous queue length with $N = 200$ and $N = 1000$, respectively. From Figures 5 and 6, one can see that in both cases, the queue length stabilizes near the target of 1500 packets after about 4 s and 10 s, respectively. The mean value of instantaneous queue length is 1499.4 for $N = 200$ versus 1500.0 for $N = 1000$, and the standard deviations is 44.8 for $N = 200$ versus 48.3 for $N = 1000$. It implies that the queue length can be basically stable at the target with only small steady-state errors. However, it

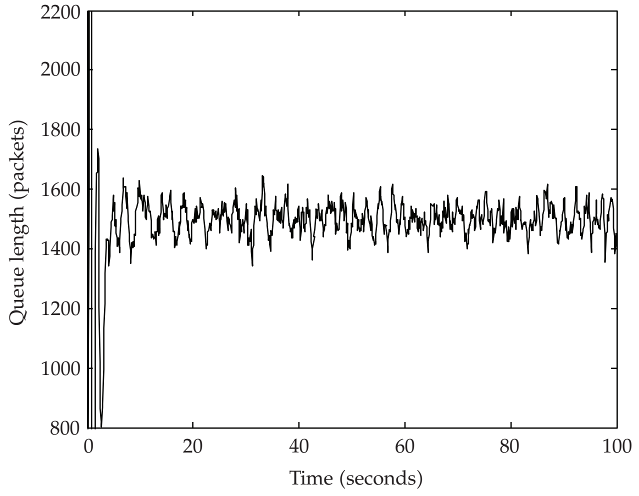


Figure 6: Instantaneous queue length ($N = 1000$).

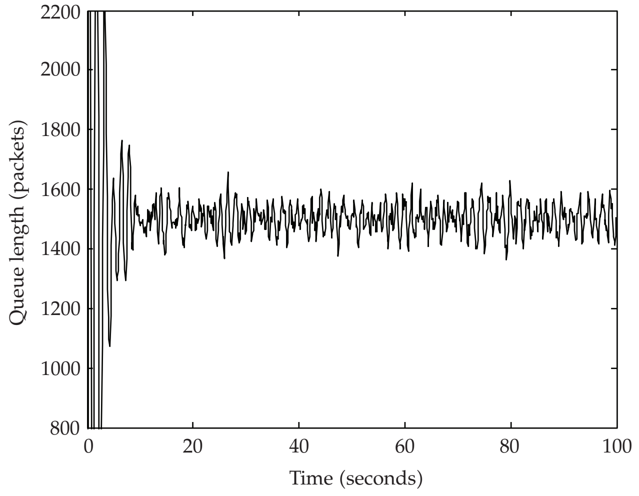


Figure 7: Instantaneous queue length with changeable number of connections.

seems to take a bit long time to become stable. The reason is that the large window size of connections leads to slow response. From the two experiments, it can be seen that the time to keep queue length stable does not increase greatly as the load increases (the number of connections). In addition, the fluctuation amplitude of queue length is very small under the above conditions. To some extent, the results show that the self-tuning RED overcomes the sensitivity to the load and shows good robustness.

5.2. Response with a Variable Number of Connections

In the experiment, the performance of the self-tuning RED is examined when the number of TCP connections varies. In initialization, the number of connections is set to 500. 100

Table 1: Comparison of simulation results.

AQM schemes	Experiment 1		Experiment 2		Experiment 3			
	Mean	STD	Mean	STD	Mean	STD	Mean	STD
REM	1511.5	136.3945	1503.0	222.6198	1512.9	133.7802	1500.6	138.6165
Adaptive RED	1421.4	41.1214	1393.9	**56.2061**	1480.1	54.1401	1383.2	47.2175
PI	1498.2	61.1963	1501.8	61.4224	**1501.8**	57.8878	1490.5	58.8450
IRED	**1499.5**	**39.3947**	**1499.0**	59.9636	1503.1	**43.8353**	**1495.8**	**46.0050**

additional TCP connections join the link at 50.0 s. Figure 7 shows that the instantaneous queue length becomes relatively stable near the queue target of 1500 packets. After 100 additional flows starting at 50.0 s, the queue first oscillates with a period of 2 s and then quickly becomes stable with about 1500 packets. The result is got with large window size. If small window size is adapted, it may take shorter time to approach robustness. From Figure 7, we can find that the self-tuning RED achieves a short response time and good robustness with variation of the number of connections.

5.3. Comparisons with Existing AQM Schemes

To investigate the performance of various AQM schemes in real network, we implement the following experiments: (1) adding short-lived TCP flows, (2) adding unresponsive UDP flows, and (3) changing the number of the long-lived TCP flows.

Table 1 gives the average value (mean in Table 1) and the standard deviation (STD in Table 1) of the instantaneous queue length for the four AQM schemes in every experiment. To compare the steady-state performance, we calculate the average and the standard deviation of queue length for second half (50 s).

Experiment 1. Adding short-lived TCP flows. There exist some short-lived flows as burst traffic in networks, which can affect the performance of an AQM scheme. In the experiments, the impact of the short-lived flows is investigated for various AQM schemes. In the first, the initial number of connections is set to 500, and then 500 short-lived TCP flows randomly arrive in interval $[0, 100]$ s. The duration of the short-lived flows is uniformly distributed in $[1\,\text{s}, 2\,\text{s}]$. Other parameters are the same as those in the above simulations.

From Table 1, it is evident that the performance of REM scheme is the worst since the algorithm cannot maintain the queue length at 1500, and its standard deviation is the largest. With adaptive RED, the fluctuation amplitude of the average queue length is smaller than those with REM, which shows that Adaptive RED behaves much better than REM in the experiments. Although adaptive RED has less standard deviation than PI controller, the deviation from the target is larger than the latter. Thus, adaptive ARED performs worse than PI controller. It is found that the self-tuning RED has less standard deviation with 39.3947 versus PI controller with 61.1963. Evidently, the self-tuning RED is more stable. To sum up the results, the self-tuning RED clearly performs better than the other schemes in the experiment.

Experiment 2. Adding unresponsive UDP flows. In this experiment, we study the performance of the AQM schemes with the UDP flows. According to the study in [46], long-lived TCP connections account for 95% of the traffics in Internet. It is necessary to examine the effect from the UDP flow. In the experiment, 50 UDP flows are added besides 500 long-lived TCP connections. These UDP flows start at random time uniformly distributed in $[0, 100]$ s, so the UDP flows can be regarded as 25 long-lived TCP connections. Each UDP is CBR (Constant Bit Rate) at the rate of 300 Kb, and the mean packets size is 500 bytes. When the queue is not empty, the sending rate of each UDP flows is nearly equal to that of TCP. By this means, the total traffic of the UDP flows account for about 5% of the traffic, which is enough to study its impact on the performance of the network.

From Table 1, we can find that with Adaptive RED, the queue cannot converge to the target value of 1500 packets (it does at about 1393), but the standard deviation with 56.2 is smaller than other algorithms. By comparing the magnitudes of oscillation for the four AQM schemes, it is found that REM leads to a largest oscillation, which implies its weak robustness. When the self-tuning RED is compared with PI controller, it can be seen that the two schemes are successful to get stable queue size with only small fluctuation at the target value. We can also see that the self-tuning RED has a little better performance than PI controller with less deviation from the target and smaller fluctuation amplitude. To sum up, the self-tuning RED is better than other algorithms in terms of robustness when the UDP flows exist.

Experiment 3. Change the number of the TCP connections. In the following experiments, the self-tuning RED is compared with REM, adaptive RED, and PI controller in cases that the number of TCP connections varies. In the first simulation, the initial number of connections is set to 500, and then 100 TCP flows start at a random time in the range $[0, 100]$ s. In the second simulation, the initial number of connections is set to 600, and 100 of them stop at the time uniformly distributed over a period of 100 s.

In both cases, the four AQM schemes perform as similarly as in the two above simulations. Due to large fluctuation, REM performs poorly in terms of robustness. With adaptive RED, the fluctuation amplitude of the queue size is very small, but it has a smaller mean queue length. PI controller outperforms Adaptive RED and REM because of less fluctuation amplitude with about 58 and smaller deviation with 10 packets from the target value. Since there are no other contenders, the self-tuning RED is compared with PI controller. In the first experiment, the mean queue length is 1503.1 with the self-tuning RED versus 1510.8 with PI controller. The standard deviation of queue length is 43.8353 with the self-tuning RED versus 57.8878 with the PI controller. In the second experiment, the mean queue length is 1495.8 with the self-tuning RED versus 1490.5 with the PI controller. The standard deviation of queue length is 46.0050 with the self-tuning RED versus 58.8450 with the PI controller. It is evident that the self-tuning RED is slightly closer to the 1500 target with smaller fluctuation amplitude, and performs much better than PI controller, adaptive RED, and REM.

6. Conclusions

In this paper, we study the effect of control parameters on queue length for a given target. Based on theoretical analysis, a self-tuning RED algorithm is proposed, which can keep queue length stable at the target value by adjusting the maximum drop probability and setting a proper exponential averaging weight. A number of simulations show that the self-tuning RED performs better than various AQM algorithms in term of deviation from the target and the fluctuation of queue length. In addition, the self-tuning RED does not change the basic principle of the RED algorithm, and can be simply applied in practice. For future work, it is necessary to further study the issue under wider network scenarios with various bandwidths and multiple bottleneck links.

Acknowledgment

The work was supported by National Natural Science Foundation of China (Grants nos. 60703112, 60873264, and 61070214).

References

[1] E. G. Bakhoum and C. Toma, "Dynamical aspects of macroscopic and quantum transitions due to coherence function and time series events," *Mathematical Problems in Engineering*, vol. 2010, Article ID 428903, 13 pages, 2010.

[2] E. G. Bakhoum and C. Toma, "Relativistic short range phenomena and space-time aspects of pulse measurements," *Mathematical Problems in Engineering*, vol. 2008, Article ID 410156, 20 pages, 2008.

[3] M. Li, "Fractal time series—a tutorial review," *Mathematical Problems in Engineering*, vol. 2010, Article ID 157264, 26 pages, 2010.

[4] S. Y. Chen, Y. F. Li, and J. Zhang, "Vision processing for realtime 3-D data acquisition based on coded structured light," *IEEE Transactions on Image Processing*, vol. 17, no. 2, pp. 167–176, 2008.

[5] S. Y. Chen, Y. F. Li, Q. Guan, and G. Xiao, "Real-time three-dimensional surface measurement by color encoded light projection," *Applied Physics Letters*, vol. 89, no. 11, Article ID 111108, 2006.

[6] L. S. Brakmo and L. L. Peterson, "TCP Vegas: end to end congestion avoidance on a global internet," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 8, pp. 1465–1480, 1995.

[7] R. J. La and V. Anantharam, "Utility-based rate control in the Internet for elastic traffic," *IEEE/ACM Transactions on Networking*, vol. 10, no. 2, pp. 272–286, 2002.

[8] 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.

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

[10] H. Wang, X. T. Ma, and Z. H. Tian, "A fuzzy self-tuning random exponential marking algorithm based on enhanced price," *Computer Simulation*, vol. 35, no. 8, pp. 128–146, 2009.

[11] S. Kunniyur and R. Srikant, "Analysis and design of an adaptive virtual queue (AVQ) algorithm for active queue management," in *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computers Communications (SIGCOMM '01)*, pp. 123–134, San Diego, Calif, USA, August 2001.

[12] S. B. Lee, K. Hur, J. Park, and D.-S. Eom, "A packet forwarding controller for mobile IP-based networks with packet buffering," *IEEE Transactions on Consumer Electronics*, vol. 55, no. 3, pp. 1344–1350, 2009.

[13] M. Li and W. Zhao, "Representation of a stochastic traffic bound," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 9, pp. 1368–1372, 2010.

[14] W. Feng, D. D. Kandlur, D. Saha, and K. G. Shin, "A self-configuring RED gateway," in *Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societie (INFOCOM '99)*, pp. 1320–1328, March 1999.

[15] S. Floyd, R. Gummadi, and S. Schenker, "Adaptive RED: an algorithm for increasing the robustness of RED's active queue management," 2001, http://www.icir.org/floyd/papers/adaptiveRed.pdf.

[16] P. Ranjan, E. H. Abed, and R. J. La, "Nonlinear instabilities in TCP-RED," *IEEE/ACM Transactions on Networking*, vol. 12, no. 6, pp. 1079–1092, 2004.

[17] C. Zhang, J. Yin, Z. Cai, and W. Chen, "RRED: robust RED algorithm to counter low-rate denial-of-service attacks," *IEEE Communications Letters*, vol. 14, no. 5, pp. 489–491, 2010.

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

[19] X.-C. Lu, M.-J. Zhang, and P.-D. Zhu, "Adaptive PI active queue management algorithm," *Journal of Software*, vol. 16, no. 5, pp. 903–910, 2005.

[20] J. Sun, M. Zukerman, and M. Palaniswami, "A stable adaptive PI controller for AQM," in *Proceedings of the International Symposium on Communications and Information Technologies (ISCIT '07)*, pp. 707–712, Sydney, Australia, October 2007.

[21] N. Xiong, Y. Pan, X. Jia, J. H. Park, and Y. Li, "Design and analysis of a self-tuning feedback controller for the Internet," *Computer Networks*, vol. 53, no. 11, pp. 1784–1797, 2009.

[22] N. Xiong, A. V. Vasilakos, L. T. Yang et al., "A novel self-tuning feedback controller for active queue management supporting TCP flows," *Information Sciences*, vol. 180, no. 11, pp. 2249–2263, 2010.

[23] S. Liu, T. Başar, and R. Srikant, “Exponential-RED: a stabilizing AQM scheme for low- and high-speed TCP protocols,” *IEEE/ACM Transactions on Networking*, vol. 13, no. 5, pp. 1068–1081, 2005.

[24] S. Guo, X. Liao, C. Li, and D. Yang, “Stability analysis of a novel exponential-RED model with heterogeneous delays,” *Computer Communications*, vol. 30, no. 5, pp. 1058–1074, 2007.

[25] C. Wang, J. Liu, B. Li, K. Sohraby, and Y. T. Hou, “LRED: a robust and responsive AQM algorithm using packet loss ratio measurement,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 1, pp. 29–43, 2007.

[26] Y. Ariba, F. Gouaisbaut, and Y. Labit, “Feedback control for router management and TCP/IP network stability,” *IEEE Transactions on Network and Service Management*, vol. 6, no. 4, pp. 255–266, 2009.

[27] B. Hariri and N. Sadati, “NN-RED: an AQM mechanism based on neural networks,” *Electronics Letters*, vol. 43, no. 19, pp. 1053–1055, 2007.

[28] H. C. Cho, M. S. Fadali, and H. Lee, “Neural network control for TCP network congestion,” in *Proceedings of American Control Conference (ACC '05)*, pp. 3480–3485, June 2005.

[29] H. Javam and M. Analoui, “SARED: stabilized ARED,” in *Proceedings of the International Conference on Communication Technology (ICCT '06)*, pp. 1–4, November 2006.

[30] J. Chen, C. Hu, and Z. Ji, “An improved ARED algorithm for congestion control of network transmission,” *Mathematical Problems in Engineering*, vol. 2010, Article ID 329035, 14 pages, 2010.

[31] J. Sun, K.-T. Ko, G. Chen, S. Chan, and M. Zukerman, “PD-RED: to improve the performance of RED,” *IEEE Communications Letters*, vol. 7, no. 8, pp. 406–408, 2003.

[32] T. Wei and S. Y. Zhang, “Fuzzy self-tuning PD-RED algorithm,” *Computer Engineering and Applications*, vol. 43, no. 5, pp. 124–126, 2007.

[33] B. Zheng and M. Atiquzzaman, “A framework to determine bounds of maximum loss rate parameter of RED queue for next generation routers,” *Journal of Network and Computer Applications*, vol. 31, no. 4, pp. 429–445, 2008.

[34] B. Zheng and M. Atiquzzaman, “A framework to determine the optimal weight parameter of RED in next-generation internet routers,” *International Journal of Communication Systems*, vol. 21, no. 9, pp. 987–1008, 2008.

[35] W. Chen and S.-H. Yang, “The mechanism of adapting RED parameters to TCP traffic,” *Computer Communications*, vol. 32, no. 13-14, pp. 1525–1530, 2009.

[36] L. Tan, W. Zhang, G. Peng, and G. Chen, “Stability of TCP/RED systems in AQM routers,” *IEEE Transactions on Automatic Control*, vol. 51, no. 8, pp. 1393–1398, 2006.

[37] F. Liu, Z.-H. Guan, and H. O. Wang, “Controlling bifurcations and chaos in TCP-UDP-RED,” *Nonlinear Analysis: Real World Applications*, vol. 11, no. 3, pp. 1491–1501, 2010.

[38] S. Woo and K. Kim, “Tight upper bound for stability of TCP/RED systems in AQM routers,” *IEEE Communications Letters*, vol. 14, no. 7, pp. 682–684, 2010.

[39] S. Misra, B. J. Oommen, S. Yanamandra, and M. S. Obaidat, “Random early detection for congestion avoidance in wired networks: a discretized pursuit learning-automata-like solution,” *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 40, no. 1, pp. 66–76, 2010.

[40] C. Cattani, “Harmonic wavelet approximation of random, fractal and high frequency signals,” *Telecommunication Systems*, vol. 43, no. 3-4, pp. 207–217, 2010.

[41] C. Cattani and A. Kudreyko, “Harmonic wavelet method towards solution of the Fredholm type integral equations of the second kind,” *Applied Mathematics and Computation*, vol. 215, no. 12, pp. 4164–4171, 2010.

[42] S. Flyod, “Recommendation on using the “gentle” variant of RED,” 2000, http://www.icir.org/floyd/red/gentle.html.

[43] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, “The macroscopic behavior of the TCP congestion avoidance algorithm,” *Computer Communication Review*, vol. 27, no. 3, pp. 67–82, 1997.

[44] C. Y.-F. Ho, B. W.-K. Ling, and H. H. C. Iu, “Symbolic dynamical model of average queue size of random early detection algorithm,” *International Journal of Bifurcation and Chaos*, vol. 20, no. 5, pp. 1415–1437, 2010.

[45] R. J. La, P. Ranjan, and E. H. Abed, “Analysis of adaptive random early detection (Adaptive RED),” in *Proceedings of the 18th International Teletraffic Congress (ITC'03)*, Berlin, Germany, 2003.

[46] C. Casetti and M. Meo, “New approach to model the stationary behavior of TCP connections,” in *Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '00)*, pp. 367–375, March 2000.