PI-PD Controller for Adaptive and Robust Active Queue Management for Internet Congestion Control

Seungwan Ryu
Department of Information Systems
Chung-Ang University, Korea, and
Mobile Telecommunications Research Division
ETRI, Korea
rush2384@cau.ac.kr

Byunghan Ryu
Mobile Telecommunications Research Division
ETRI, Korea

Myoungki Jeong
Telecom R&D Center
Samsung Electronics
Suwon, Korea

Seikwon Park
Department of Information Systems
Chung-Ang University
Korea

Since random early detection (RED) was proposed in 1993, many active queue management (AQM) algorithms have been proposed to support better end-to-end Transmission Control Protocol (TCP) congestion control. In this article, the authors introduce and analyze a feedback control model of the TCP/AQM dynamics. Then they suggest the concept of an AQM algorithm that can detect and avoid congestion proactively. Finally, they propose the proportional-integral (PI) proportional-derivative (PD) controller using proportional-integral-derivative (PID) feedback control to overcome the reactive control behavior of existing AQM proposals. The PI-PD controller is able to provide proactive congestion avoidance and control using an adaptive congestion indicator and a control function. A comparative simulation study under a variety of network environments shows that the PI-PD controller outperforms RED and the PI controller in terms of the queue length dynamics, the packet loss rates, and the link utilization.

Keywords: Active queue management, feedback control, congestion control, proportional-integral-derivative control

1. Introduction

The Internet has been designed to provide best-effort traffic, and the end-to-end congestion control mechanism performed at the transport layer in end systems has been developed to support and manage this Internet traffic efficiently. The current Transmission Control Protocol (TCP)–based end-to-end congestion control mechanism operates with first-in, first-out (FIFO)–based tail-drop (TD) queue management at routers. Since TD has some drawbacks, such as low link utilization, high queuing delay, and unfair bandwidth allocation caused from full-queue and lockout phenomena [1], the performance of TCP congestion control is still unsatisfactory, even with improvements on the end-system algorithms such as slow start, fast retransmit, and fast recovery.

To remedy the performance degradation of the current TCP congestion control over TD at routers, active queue management (AQM) algorithms such as random early detection (RED) [2] have been introduced. An AQM algorithm monitors and controls traffic within a router where the congestion occurs and is controlled using more accurate congestion information than at sources. Another advantage of AQM is that in the absence of cooperation from sources,

SIMULATION, Vol. 81, Issue 6, June 2005 437-459

DOI: 10.1177/0037549705056202

or if sources are not responsive to congestion control, AQM is capable of controlling congestion solely at routers. Thus, the AQM algorithm plays like an admission controller at a router to detect and control congestion effectively.

Two main functions are used in AQM: one is the congestion indicator (i.e., how to detect congestion), and the other is the congestion control function (i.e., how to avoid and control congestion). The TD mechanism uses the instantaneous queue length as a congestion indicator and drops packets when the buffer becomes full. Since it is simple and easy to implement, TD is the most currently used queue management at Internet routers. RED enhanced the two functions by introducing queue length averaging and probabilistic early packet dropping. In particular, RED uses the exponentially weighted moving average (EWMA) queue length not only to detect incipient congestion but also to smooth the bursty incoming traffic and its resulting transient congestion. Following RED [2], many AQM-based extensions such as BLUE [3], Adaptive-RED [4], SRED [5], and so on have been proposed.

One important drawback of currently proposed AQM algorithms is that their congestion detection and control functions depend only on the current queue status or the history of the queue status (e.g., average queue length). Hence, the congestion detection and control in these algorithms are *reactive* to current or past congestion, not *proactive* to incipient congestion [6, 7]. For example, RED can detect the long-term traffic patterns using exponentially weighted average queue lengths and notify the onset (persistent) congestion to sources. However, it is unable to detect the incipient congestion caused by short-term traffic load changes. It has been observed that the dynamically changing traffic is mainly caused by short-lived connections in size and lifetime (so-called *mice*) traffic [8-10], which consists of 50% to 70% of TCP flows. In this case, the implicit congestion notification that is fed back to end hosts by a packet drop may be the wrong control signal and can possibly make the congestion situation worse.

To address these problems, it is necessary for an AQM algorithm to have a more efficient congestion indicator and control function. To avoid or control congestion *proactively* before it becomes a problem, both the congestion indicator and control function of an AQM should be adaptive to changes in the traffic environments such as in the amount of traffic, the fluctuation of traffic load, and the nature of traffic.

In this article, we propose a proportional-integral (PI) proportional-derivative (PD) controller algorithm that can detect and control the incipient congestion as well as the current congestion effectively and predictively. The goals of the PI-PD controller are to control congestion predictively, to make the queue length agree with a desired level, and to give smooth and low packet loss rates. It is important to have an efficient congestion prediction and control capability to achieve these goals. The concept of a classical proportional-integral-derivative (PID) feedback control is used in designing the PI-PD controller not only to have the anticipatory congestion detection and control capability but also to achieve the long-term control performance, such as acceptable queue length behavior (or, equivalently, delay), acceptable packet loss rates, or high link utilization.

Recently, the AQM problem has been studied using different techniques of control theory [11-14]. In particular, the concept of PID feedback control has been introduced in designing an AQM algorithm in a few studies [7, 15-17]. In these studies, continuous PID controllers are designed first by using the linearized TCP/AQM dynamic model derived in Hollot et al. [18], and then they are digitized for practical implementation using the emulation method [19]. In particular, to design a continuous PID controller, the time domain design method is used in Ryu and Rump [7], whereas the frequency domain design method is used in Fengyuan and Chuang [15] and Yanfei, Fengyuan, and Chuang [17]. On the other hand, in Ryu and Cho [20], the PI-PD controller was proposed as an AQM algorithm designed based on the concept of PID feedback control using the direct design method [21]. In the direct design method, a discrete PID feedback control equation is used to obtain feedback control equations.

In this article, the PI-PD controller [20] is extended in various ways. First, additional analysis of recently proposed AQM algorithms is performed in the context of feedback control theory. In addition, control performance of the PI-PD controller is evaluated extensively under various traffic situations and compared with those of other AQM algorithms such as the PID controller [7], RED [2], and the PI controller [18]. The key features of the PI-PD controllers are the following:

- The PI-PD controller is able to not only control existing congestion reactively based on the current (P) and the past (I) congestion information but also avoid the incipient congestion proactively based on the future (D) congestion. To achieve these control capabilities, the PI-PD controller is designed based on the concept of the PID feedback control.
- The PI-PD-controller controls traffic effectively by maintaining the queue length around a desired level, with acceptable deviation around it to avoid unnecessary packet drop. So, the PI-PD controller can absorb bursty traffic and gives very low and stable packet loss over time as a result.
- The PI-PD controller can achieve two conflicting design goals—stability and responsiveness—simultaneously by means of PI control and PD control.
- The PI-PD controller maintains only two parameters, the control gain (α) and the sampling time interval (T_s). This feature makes the PI-PD controller scalable and robust. In contrast, RED, for instance, maintains five parameters (max_{th}, min_{th}, max_p, W_q, Q_{avg}; see [2]), and tuning these parameters to the dynamically changing traffic situation is extremely difficult.
- The PI-PD controller gives comparably less computational overhead. For example, the PI-PD controller can be easily implemented with less sampling frequency compared to the link speed implementation of RED.

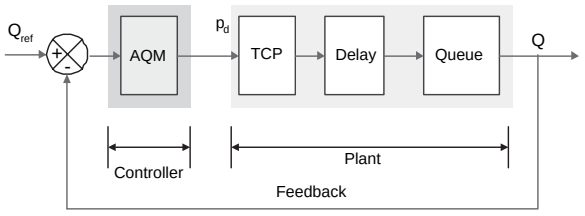

Figure 1. Feedback control modeling of Transmission Control Protocol (TCP) congestion control with the active queue management (AQM) algorithm

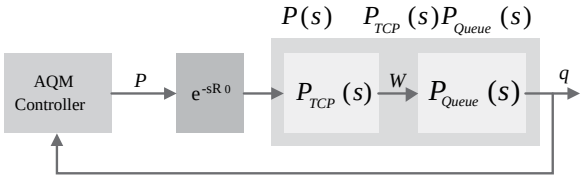

Figure 2. A feedback control model of the Transmission Control Protocol (TCP)/active queue management (AQM) dynamics

This article is organized as follows. In the next section, we review and analyze some representative TCP/AQM dynamics, including RED [2], the PI controller [18], BLUE [3], Adaptive Virtual Queue (AVQ) [22], and the PID controller [7], in the context of feedback control theory. Then, we design the PI-PD controller using PID feedback control and propose a practical implementation method at a router. We compare the control performance of the PI-PD controller with that of other AQM algorithms, such as the PID controller, RED, and the PI controller, under various traffic situations via simulation using `ns-2` [23]. Finally, we conclude our study with suggested directions for future study.

2. Control-Theoretic Design Consideration of an AQM Algorithm

2.1 Feedback Control and TCP Congestion Control with AQM

TCP congestion control dynamics with an AQM can be modeled as a feedback control system (Fig. 1). In this modeling, the feedback control system consists of (1) a desired queue length at a router (i.e., a *reference input*), denoted by Q_{ref}; (2) the queue length at a router as a plant variable (i.e., a *controlled variable*), denoted by Q; (3) a *plant* that represents a combination of subsystems such as TCP sources, routers, and TCP receivers that send, process, and receive TCP packets, respectively; (4) an *AQM controller*, which controls the packet arrival rate to the router queue by generating packet drop probability as a control signal; and (5) a *feedback signal*, which is a sampled system output (i.e., the sampled queue length) used to obtain the control error term, $Q - Q_{ref}$.

In Misra, Gong, and Towsley [24], a system of nonlinear differential equations for TCP/AQM dynamics was developed using fluid-flow analysis while ignoring the TCP timeout and the slow-start mechanism. Then, in Hollot et al. [25], a linearized and simplified TCP/AQM dynamic model was developed and analyzed, especially for TCP/RED dynamics, in terms of (feedback) control theory. The open-loop transfer function of the plant, $P(s) = P_{TCP}(s) \cdot P_{Queue}(s)$, was given by

$$P(s) = \left(\frac{\frac{R_0C^2}{2N^2}}{s + \frac{2N}{R_0^2C}}\right) \cdot \left(\frac{\frac{N}{R_0}}{s + \frac{1}{R_0}}\right), \qquad (1)$$

where N is a load factor (the number of TCP connections), R_0 is roundtrip time, and C is the link capacity. A block diagram of a linearized TCP/AQM dynamic model is shown in Figure 2.

Since the open-loop transfer function of TCP flows (1) has two nonzero poles, it is a type 0 system [26]. Thus, there always exists constant steady-state error for the step function input [26]. Since $\lim_{s\to 0} P(s) = (R_0C)^3/(4N^2)$, the steady-state error of the open-loop transfer function is $e_{ss} = M/(1 + \lim_{s\to 0} P(s))$, where M is the magnitude of a reference input.

EXAMPLE 1. *A Sample Network Configuration.* The TCP flow dynamics ($P(s)$) is shown to be stable when $N \geq N^-$ and $R_0 \leq R^+$ [25]. If we set $N^- = 60$ and $R^+ = 0.246$, similar to the network configuration in Hollot et al. [18, 25],

$$P(s) = \frac{\frac{C^2}{2N}}{(s + \frac{2N}{R_0^2C})(s + \frac{1}{R_0})} = \frac{117187.3}{(s + 0.53)(s + 4.05)}.$$

Then, the undamped system frequency (ω_n), the damping ratio (ξ), and the time domain performance specifications [26] are as follows:

- $\omega_n = 342.3$ rad/sec, $\xi = 4.5793/(2\omega_n) = 0.0067$.
- e_{ss} = the steady-state error $= 1/(1 + \lim_{s\to 0} P(s)) = 1/54985.1 > 0$.
- The maximum overshoot (MOS) (%) $= 100e^{-\pi\xi/\sqrt{1-\xi^2}} = 97.92\%$.
- The rise time (t_r) $= 1.8/\omega_n = 5.26$ msec.
- The settling time (t_s) $\simeq 4/(\xi\omega_n) = 1.75$ sec.

Figure 3 shows the MATLAB simulation result of the unit-step response of the TCP flow dynamics (1). Because of the longer settling time relative to a very short rise time (t_r) with a very small damping ratio, $\xi = 0.0067$, TCP flow shows severely oscillating output signal dynamics.

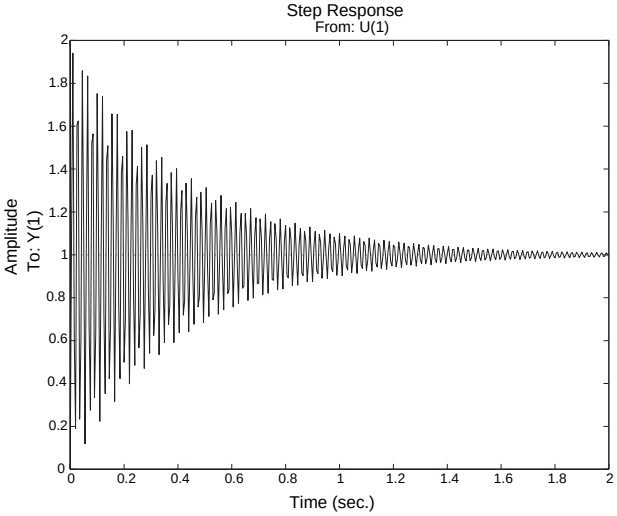

Figure 3. Unit-step response of the linearized Transmission Control Protocol (TCP) flow dynamics to the unit-step function

Therefore, a well-designed AQM controller should be able to compensate for the oscillatory TCP dynamics and give satisfactory control performance such as fast and stable control dynamics.

2.2 The RED Controller

With RED, a link maintains the EWMA queue length, $Q_{avg} = (1 - w_Q) * Q_{avg} + w_Q * Q$, where Q is the current queue length and w_Q is a weight parameter, $0 \leq w_Q \leq 1$. When Q_{avg} is less than the minimum threshold (min_{th}), no packets are dropped (or marked). When it exceeds the maximum threshold (max_{th}), all incoming packets are dropped. When it is in between, a packet is dropped with a probability p_d that is increasing in function with Q_{avg}. More specifically, if $min_{th} \leq Q_{avg} \leq max_{th}$, then the packet drop probability, p_d, is calculated as

$$p_d = max_p * \frac{Q_{avg} - min_{th}}{max_{th} - min_{th}}, \tag{2}$$

where max_p is the maximum value of p_b.

RED attempts to eliminate the steady-state error by introducing the EWMA error terms[1] [27] as an integral (I) control to the open-loop transfer function. However, since RED introduces a range of reference input values (i.e., $[min_{th}, max_{th}]$) rather than a unique reference input for the I control, the TCP/RED model shows oscillatory system dynamics. Moreover, a very small weight factor value ($w_Q = 1/512 \cong 0.002$) to the current queue length in calculating the EWMA queue length is one of the main reasons for queue length oscillation and the bias against bursty sources [28, 29]. In addition, the small value of w_Q brings an effect of large integral time in I control and may accompany large overshoot [28]. As a result, RED shows oscillatory queue length dynamics and gives poor control performance under a wide range of traffic environments.

1. The filtering of the queue length is equivalent to the filtering (integration) of the error terms.

2.3 The PI Controller

The idea behind the PI controller [18] is to make the queue length agree with the desired queue length, Q_{ref}, by introducing an I control to an AQM. The PI controller has been designed analytically based on the linearized TCP flow dynamics model (1) not only to improve responsiveness of the TCP/AQM dynamics but also to stabilize the router queue length around a desired queue length. The latter can be achieved by means of I control, while the former can be achieved by means of proportional (P) control using the instantaneous queue length rather than using the average queue length. Thus, the resulting PI controller is able to eliminate the steady-state error regardless of the load level. The PI controller can be implemented at a router as an alternative AQM algorithm by digitization, where the packet drop probability at time $t = kT$ is

$$p(kT) = a(\delta Q)(kT) - b(\delta Q)(k-1)T + p((k-1)T), \tag{3}$$

where $f_s = 1/T$ is the sampling frequency, which is recommended to be 10 to 20 times the system frequency (f) [18]; a and b are constants; and $\delta Q = Q - Q_{ref}$ is a deviation of the queue length from the desired queue length, Q_{ref}. The recommended design rules [18] are

$$w_g = \frac{2N^-}{(R^+)^2 C}, \quad K_{PI} = w_g \left| \frac{(\frac{jw_g}{p_{queue}} + 1)}{\frac{(R^+C)^3}{(2N^-)^2}} \right|, \tag{4}$$

where $w_g = 2\pi f$ is the unity-gain crossover frequency, and K_{PI} is a PI control gain.

2.4 BLUE

BLUE [3] was proposed to overcome the drawbacks of RED and other AQM algorithms that use the (EWMA) queue length as a congestion indicator. BLUE uses packet loss and link-idle events as a congestion indicator and maintains a single packet drop probability, p, to control congestion. BLUE adjusts p in every fixed time interval, $freeze\_time$, when a packet loss or a link-idle event occurs. In particular, BLUE increases p by a small amount of d_inc in response to the buffer overflow (i.e., packet loss) and decreases p by a small amount of d_dec when the link becomes idle. Thus, BLUE can be considered as a modified, multipositional ON-OFF controller or a *relay controller* with hysteresis $freeze\_time$ and two control outputs [30]: if there is a packet loss (ON), the relay sends a control signal of d_inc units, and if there is an link idle (OFF), the relay sends a control signal of d_dec.

However, since BLUE adjusts p only after the packet loss or link-idle event occurs, BLUE controls congestion reactively based on the current or past congestion information. Thus, some degree of performance degradation, such as multiple packet losses and link underutilization, is not avoidable. Moreover, because of fixed values of the control parameters—d_inc, d_dec, and $freeze\_time$—BLUE is unable to provide adaptive control to the changing network traffic situations.

2.5 Adaptive Virtual Queue

Adaptive virtual queue (AVQ) [22] uses a modified token bucket model as a virtual queue (VQ) to regulate the buffer utilization rather than the queue length. AVQ adjusts the size and link capacity of the VQ proportional to the measured input rate and drops packets when the VQ overflows. In detail, AVQ maintains a virtual queue whose capacity of a link, $\tilde{C}$, is less than the actual capacity of a link, C, and whose buffer size is the same as the buffer size of the real queue. At each packet arrival, the virtual queue is updated according to the differential equation

$$\dot{\tilde{C}} = \alpha(\gamma C - \lambda),$$

where λ is the packet arrival rate, γ is a desired link utilization, and α is a damping factor. If the new packet overflows the virtual queue, the packet is dropped in the virtual queue, and the real packet is dropped in the real queue.

AVQ shows lower average packet loss rates and higher link utilization than other AQM algorithms, such as RED and the PI controller, with a packet-marking mode under a certain traffic situation consisting mainly of elephant FTP flows [22]. However, the control performance of AVQ should be examined under more realistic traffic situations consisting mainly of web-like, short-lived *mice* flows under the packet drop mode. Moreover, the link speed control of AVQ will give very heavy computational overhead to the router or be impossible to implement at a router as the network evolves to high-speed networks.

2.6 PID Controller

Recently, the concept of classical PID control has been introduced in designing an effective AQM algorithm [7, 15, 17]. In Fengyuan and Chuang [15] and Yanfei, Fengyuan, and Chuang [17], the frequency response method was used in designing an AQM algorithm, whereas the time response method[2] is used in Ryu and Rump [7]. In general, the choice of a design method depends on the designer's preference [26]. PID controllers are commonly designed using time domain methods when the plant model is a second-order system [26].

The PID controller [7] was designed to improve the speed of response and to eliminate (or minimize) the steady-state error of the TCP/AQM system at the same time by applying the classical PID feedback control mechanism. The resulting PID controller consists of a PI control portion connected in serial with a PD control portion. The PD control part is designed to improve the damping and speed of response of a control system but cannot eliminate the steady-state error [26]. In contrast, the PI control part is designed to eliminate the steady-state error at the expense of an increase of response time. Then, the PID control equation (7) (in Laplace transform (s-domain)), $D(s) = K_P + K_I/s + K_D s$, becomes an equation consisting of the PD and the PI control parts as follows:

$$\begin{aligned} D(s) &= (K_{P1} + K_{D1}s)\left(K_{P2} + \frac{K_{I2}}{s}\right) \\ &= D_{PD}(s) \cdot D_{PI}(s), \end{aligned} \quad (5)$$

where $K_P = K_{P1}K_{P2} + K_{D1}K_{I2}$, $K_D = K_{D1}K_{P2}$, and $K_I = K_{P1}K_{I2}$.

The PID controller was designed based on the TCP flow dynamics (1) as a plant model by applying (5). Since the controlled system, the TCP flow dynamics (1), is a second-order system, time domain performance specifications such as the rise time, the settling time, the maximum overshoot (MOS), the time constant, and the steady-state error are available analytically [26]. Thus, the PID controller has been designed in Ryu and Rump [7] using the time domain design and analysis method.

The PD control part is designed by finding values of PD control parameters, K_{P1} and K_{D1}, for given acceptable bounds of the time domain performance specifications [26]. Once K_{P1} and K_{D1} are decided in the design of the PD control part, the PID controller is obtained by finding values of the PI control parameters, K_{P2} and K_{I2}. Then, three term PID control parameters, K_P, K_D, and K_I, are obtained from (5).

The resulting PID controller can be implemented at a router by finding an equivalent discrete controller from a continuous model by *emulation* [19]. Particularly, once the sampling frequency ($f_s = 2\pi\omega_n = 1/T_s$) is decided analytically or empirically, the digitized PID control equation is obtained using *Tustin's method* for integration and the *backward* rectangle method for derivation.

$$\begin{aligned} p_d(k) &= p_d(k-1) + \Delta p_d(k) \\ &= p_d(k-1) + a_1 e_k - b_1 e_{k-1} + c_1 e_{k-2}, \end{aligned} \quad (6)$$

where $p_d(k)$ is the packet drop probability at time $k = \lfloor t/T_s \rfloor = 0, 1, \ldots$, $a_1 = \left(K_P + \frac{K_D}{T_s} + \frac{T_s}{2T_I}\right)$, $b_1 = \left(K_P + \frac{2K_D}{T_s} - \frac{T_s}{2T_I}\right)$, $c_1 = \frac{K_D}{T_s}$, and $e_k = Q_k - Q_{ref}$.

3. PI-PD Controller

3.1 Adaptive Congestion Indicator and Control Function

Since each TCP source controls its sending rate through window size[3] adjustment [31], the aggregate input traffic

2. Frequency response and time response methods are often called frequency domain and time domain methods.

3. Total number of TCP sessions (or IP packets) outstanding in the

load (the offered load), λ_t, is proportional to the total window size of all connections, W (i.e., $W \propto \lambda_t(R + Q_t/C)$), where R is the average propagation delay of all connections, Q_t/C is the queuing delay at a router, C is the output link capacity, and Q_t is the current queue length. Because of limited traffic processing capacity at a router (e.g., finite buffer size and output link capacity), not all the offered traffic load λ_t is carried at a router. Thus, the *carried traffic load* (or queued traffic, equivalently), λ_t', will be a fraction of the offered traffic load that is not dropped at a router (i.e., $\lambda_t' = \lambda_t(1 - P_d)$), where P_d is the packet drop probability.

In a time-slotted model,[4] the current queue length, Q_k, is a function of λ_k' (i.e., $Q_k = (\lambda_k' - C)T_s + Q_{k-1}$), where C is the output link capacity and Δt is the unit length of a time slot. However, the incipient congestion will be a function of the queue length of the next time slot, Q_{k+1}, not a function of Q_k. Therefore, to avoid upcoming congestion, an AQM algorithm should be able to detect and regulate the incipient congestion proactively, not the current congestion reactively. On the other hand, an AQM algorithm should be able to control current congestion using the current and past congestion information. Therefore, it is necessary for an AQM to generate the packet drop probability, p_d, based on not only the current and/or the past congestion but also the incipient congestion to provide effective congestion control and avoidance at a router.

Unfortunately, most AQM algorithms such as RED [2] or the PI controller [18] use only the past traffic history such as Q_t (or the average queue length $\overline{Q}$) as a congestion indicator. As a result, these AQM algorithms are unable to detect incipient congestion adaptively to the traffic load variations.

3.2 Proportional-Integral-Derivative Control

To detect and control the incipient as well as the current congestion proactively by regulating the queue length around a desired level (Q_{ref}), an elaborate controller having the ability to predict and adjust control performance is required. This can be achieved by the PID feedback control. The proportional (P), integral (I), and derivative (D) feedback in the PID control is based on the past (I), current (P), and future (D) control error [32]. In particular, PID control generates a control signal proportional to a linear combination of current error (P), the integral of previous errors (I), and the changing rate of current error (D). Thus, the derivative term plays a very important role in controlling the future error (i.e., the incipient congestion). A generic PID control equation is a differential equation:

$$u(t) = K_P e(t) + \frac{1}{T_I}\int e(\tau)d\tau + K_D\frac{d}{dt}e(t)$$

$$= K_P\left\{e(t) + \frac{1}{T_I'}\int e(\tau)d\tau + K_D'\frac{d}{dt}e(t)\right\}, \quad (7)$$

where $u(t)$ is a control signal at time $t \geq 0$, K_P is the proportional gain, $T_I' = K_P T_I$[5] is an integral time (or the reset time), and $K_D' = K_D/K_P$ is a derivative time (or the rate time). Then, the corresponding discrete PID control equation obtained from (7) for a small time interval, T_s, becomes a difference equation [33].

$$u(k) = K_P\left[e(k) + \frac{T_s}{T_I'}\sum_{i=0}^{k-1} e(i) + \frac{K_D'}{T_s}[e(k) - e(k-1)]\right], \quad (8)$$

where $u(k)$ is a control signal at a sampling time, $k = \lfloor t/T_s \rfloor = 0, 1, \ldots$, and $e(k)$ is a sampled error term at time k.

3.3 A Summary of the PI-PD Control Concept

The PI-PD controller is designed to detect the incipient congestion as well as the current congestion by adopting an adaptive congestion indicator and a control function. For predictive congestion control, the PI-PD controller enhanced the control function with the introduction of a predictive traffic measure for the adaptive congestion indication. The goals of the PI-PD controller are to avoid and control congestion proactively by anticipating incipient congestion, to stabilize the queue length at a router around a desired queue length (Q_{ref}), and to provide acceptable bounded queuing delay, higher link utilization, and so on.

For practical implementation of the PI-PD controller, the traffic history of two recent time slots is used for several reasons: (1) the dynamic changes of network traffic with time, (2) the recent observation of Poisson-like network traffic behavior under heavy traffic loads[6] [34, 35], and (3) derivation of the rate of change of the traffic load to estimate the traffic load for the next time slot in the practical implementation of PD control. In general, the linear extrapolating method [36] is used to obtain the estimated traffic measure for the next time slot using the slope of the current measure, that is, $e(t + T_s) \simeq e(t) + T_s\frac{d}{dt}e(t)$. However, in cases of monotonic increasing and decreasing traffic patterns, we can obtain a more accurate estimated measure using the rate of change of the traffic load in two recent time slots, $e(t + T_s) \simeq e(t) + T_s\frac{d^2}{dt^2}e(t)$.

Since one major goal of the PI-PD controller is to resolve two conflicting control targets—the *responsiveness*

network.

4. In this model, time is divided into small time slots (T_s). At the end of each time slot, the queue size, Q_k, and total amount of queued input traffic, λ_k', are calculated, where $k = \lfloor t/T_s \rfloor = 0, 1, 2, \cdots$.

5. $K_I' = 1/T_I'$ is the integral gain.

6. As the network traffic load and types of the traffic sources increase, the long-range dependence of the network traffic decreases to independence. Moreover, because of the statistical multiplexing of the traffic sources, it has been observed that at a higher traffic rate, the packet interarrival processes are expected to behave like Poisson arrivals with independent service times, while at a lower rate, there exists long-range dependence.

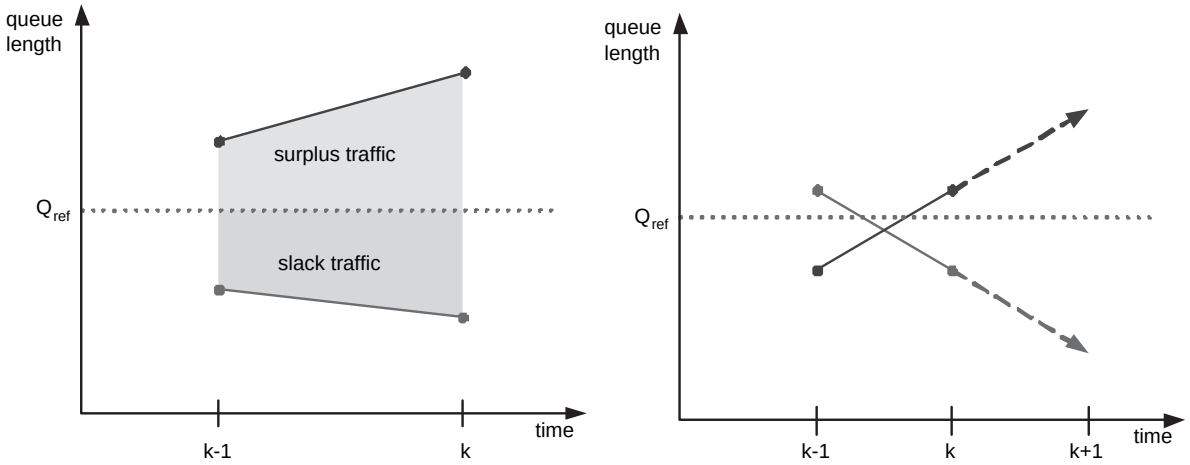

Figure 4. Two cases of traffic load change situations: (1) surplus slack traffic and (2) rapidly changing case

(the short-term control performance) and the *stability* (the long-term control performance)—it is designed based on two parts, a PI control part for stability and a PD control part for responsiveness. In addition, to stabilize the queue length to the desired level, Q_{ref}, the concept of a velocity PID control [33] is also used in the digital implementation of the PI-PD controller. The rationale for using the concept of a velocity PID control is that it updates the control signal recursively by summing the current rate of change of the control signal, $\Delta u(k)$, and the previous control term, $u(k-1)$. Also, a velocity control can provide better control performance from its anticipatory control nature.

If two recently sampled queue lengths, Q_k and Q_{k-1}, are larger (smaller) than Q_{ref} (Fig. 4), there is a surplus (slack) amount of the traffic load. In this case, the control signal—that is, the packet drop probability (p_d)—can be adjusted by taking the surplus (slack) amount of traffic load into account for adjustment of p_d to maintain the queue length around Q_{ref}.

When Q_{k-1} and Q_k are located in different regions against Q_{ref}, as shown in Figure 4 (e.g., $Q_{k-1} < Q_{ref}$ or $Q_k \geq Q_{ref}$), it means that the input traffic load is changing rapidly and tending away from Q_{ref}. In this case, it is necessary to estimate a more accurate queue length for the next time slot than using the linear extrapolating method for regulating the rapidly changing traffic around Q_{ref} effectively. For example, if $Q_{k-1} < Q_{ref}$ and $Q_k \geq Q_{ref}$ (Fig. 4), then the queue length, Q_{k-2}, sampled at time $k-2$ is needed to get the tendency of the traffic load change in the past two time slots, $[k-2, k-1)$ and $[k-1, k)$. In this case, there are three possible cases, as shown in Figure 5.

- When $Q_{k-2} > Q_{k-1}$ (case 1 in Fig. 5), the linear extrapolating method can be used to estimate the queue length for the next time slot, $\hat{Q}_{k+1}$, because traffic tendency has been changed in the last time slot. Thus, a will be the estimated value for $\hat{Q}_k$.

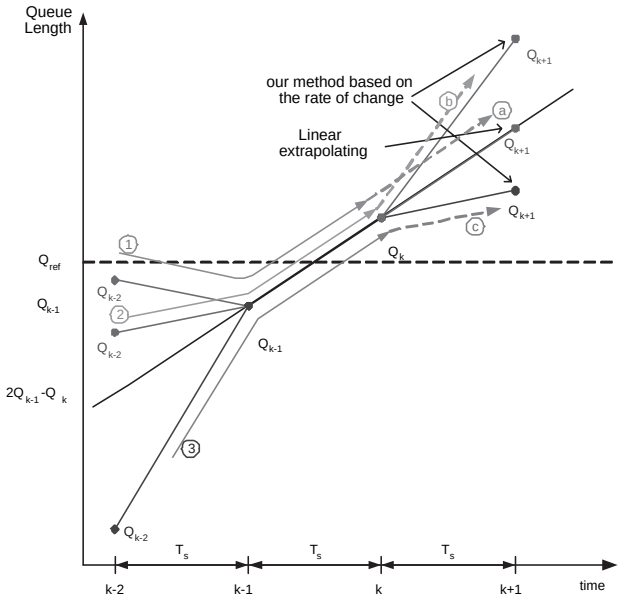

Figure 5. Example cases of the traffic load change in two recent time slots

- When $Q_{k-2} \leq Q_{k-1}$ (cases 2 and 3 in Fig. 5), the queue length in the past two time slots is monotonically increasing. In these cases, application of the rate of change of the traffic load will give a better estimated value for $\hat{Q}_{k+1}$ than of the linear extrapolating method for it. However, there will be two different estimated values for $\hat{Q}_{k+1}$ depending on the rate of change, as shown in Figure 5 (i.e., b for case 2 and c for case 3).

Figure 5 describes three possible cases for the estimation of $\hat{Q}_{k+1}$ and shows the superiority of our method in accuracy on estimating $\hat{Q}_{k+1}$ over the linear extrapolating method.

3.3.1 PI Control

The first part of the PI-PD controller is the PI control. A discrete PI control equation is obtained from (8) when $K'_D = 0$.

$$u(k) = K_P \left[e(k) + \frac{T_s}{T'_I} \sum_{i=k-2}^{k-1} e(i) \right].$$

Then, the velocity control implementation of this PI control is $u(k) = u(k-1) + \Delta u(k)$. Since the integral time, T'_I, is assumed to consist of two time slots ($T_I = 2T_s$) and $e(k-1) + e(k-3) = (Q_{k-1} - Q_{ref}) + (Q_{k-3} - Q_{ref}) = Q_{k-1} + Q_{k-3} - 2Q_{ref}$, the instantaneous PI control signal, $\Delta u(k) = u(k) - u(k-1)$, is

$$\begin{aligned} \Delta u(k) &= K_P \left\{ e(k) - \frac{1}{2}[e(k-1) + e(k-3)] \right\} \\ &= K_P \left\{ e(k) - \left[\frac{Q_{k-1} + Q_{k-3}}{2} - Q_{ref} \right] \right\} \end{aligned} \quad (9)$$

Since the current queue length, Q_k, and at most two recent queue lengths, $Q_i, i = k-1, k-2$, can be used for the integration (or summation) of errors, (9) is modified to use at most two recent queue lengths rather than using Q_{k-3} for a discretized velocity PI control. In particular, if both Q_k and Q_{k-1} are greater than Q_{ref} (or less than Q_{ref}), the average surplus (or slack) amount of traffic is calculated using the trapezoidal-integral rule [26]. Then the packet drop probability p_d is adjusted proportionally to the amount of surplus or slack traffic. In the case of surplus traffic, the packet drop probability becomes aggressive by increasing p_d proportional to the amount of surplus traffic. In the case of slack traffic, the packet drop probability becomes conservative by decreasing p_d proportional to the amount of slack traffic. Thus, the PI control equation is

$$p_d(k) = p_d(k-1) + \alpha \left[\frac{Q_k + Q_{k-1}}{2} - Q_{ref} \right], \quad (10)$$

where α is a control gain. For better scaling of the PI control implementation, the current amount of surplus/slack traffic, $[(Q_t + Q_{t-1})/2] - Q_{ref}$, is normalized by Q_{ref}. This value represents the fraction of the amount of current surplus/slack traffic to the desired queue level. Thus, the normalized PI control equation is

$$p_d(k) = p_d(k-1) + \alpha \left[\frac{[(Q_t + Q_{t-1})/2] - Q_{ref}}{Q_{ref}} \right]. \quad (11)$$

3.3.2 PD Control

The second part of the PI-PD controller is the derivative (D) control. A discrete D control equation is obtained from (8) when $K_P = 0$ and $K_I = 1/T_I = 0$.

$$u(k) = \frac{K_P K'_D}{T_s} \Big[e(k) - e(k-1) \Big].$$

Then, similar to the PI control part, a discretized velocity D control is $u(k) = u(k-1) + \Delta u(k)$, and the instantaneous control signal, $u(k)$, is

$$\begin{aligned} \Delta u(k) &= \frac{K_P K'_D}{T_s} \Big[e(k) - 2e(k-1) + e(k-2) \Big] \\ &= K_P K'_D \left[\frac{e(k) - e(k-1)}{T_s} - \frac{e(k-1) - e(k-2)}{T_s} \right]. \end{aligned}$$

From the relation between a differential equation and a difference equation,

$$\begin{aligned} \frac{d^2 e(t)}{dt^2} &\approx \frac{e(k) - 2e(k-1) + e(k-2)}{(T_s)^2} \text{ and} \\ \Delta u(k) &\approx K_P K'_D \left[T_s \frac{d^2}{dt^2} e(t) \right]. \end{aligned}$$

Thus, $T_s \frac{d^2}{dt^2} e(t)$ represents the predicted amount of an acceleration of changes on $e(t)$ in time T_s ahead, that is, $e(t + T_s) = e(t) + K_P K'_D [T_s \frac{d^2}{dt^2} e(t)]$. Then, the tendency (acceleration) of the input traffic is obtained from the discretized implementation of a velocity D control using the traffic history. The derivative time, K'_D, is assumed to be the unit length of a time slot (i.e., $K'_D = T_s$). Hence,

$$\begin{aligned} \Delta u(k) &= K_P K'_D \left[\frac{e(k) - 2e(k-1) + e(k-2)}{T_s} \right] \\ &= K_P \Big[Q_k - 2Q_{k-1} + Q_{k-2} \Big]. \end{aligned} \quad (12)$$

To obtain the predicted amount of error for the next time slot, $T_s \frac{d^2}{dt^2} e(t)$, and the incipient congestion indicator (i.e., the predicted queue length for the next time slot ($\hat{Q}_{k+1}$)), the traffic status at sampling time i, S_i, is set from the surplus or slack amount of traffic, $\gamma_i = Q_i - Q_{i-1}$:

$$S_i = \begin{cases} 1 & \text{, if } \gamma_i \geq 0 \\ -1 & \text{, otherwise.} \end{cases} \quad (13)$$

Then, the tendency of the input traffic of the previous two time slots, $S = S_k * S_{k-1}$, is used to predict the queue length change. If $S = 1$, then this indicates either a monotonic building or draining of queued traffic, in which case the tendency (or changing rate) is γ_k / γ_{k-1}, and the predicted amount of change in the queued traffic is $\hat{\gamma}_{k+1} = \gamma_k (\gamma_k / \gamma_{k-1})$. If $S \neq 1$, then this indicates that the

tendency of the input traffic has changed, in which case the predicted amount of change in the queued traffic is obtained by linearly extrapolating [36] the current change in the queued traffic (i.e., $\hat{\gamma}_{k+1} = T_s(\gamma_k / T_s) = \gamma_k$). Thus,

$$\hat{\gamma}_{k+1} = \begin{cases} \gamma_k \dfrac{\gamma_k}{\gamma_{k-1}} & \text{, if} \quad S = 1 \\ \gamma_k & \text{, otherwise.} \end{cases} \tag{14}$$

With the above two cases of the velocity D control implementation, PI-PD is able to give a more accurate estimate of the predicted error and generate a more accurate control signal for the next time slot. The predicted queue length for the next time slot is $\hat{Q}_{k+1} = \hat{\gamma}_{k+1} + Q_k$, and the corresponding predicted error is $\hat{e}(k+1) = \hat{Q}_{k+1} - Q_{ref}$. Then the D control equation is

$$p_d(k) = p_d(k-1) + \alpha[\hat{Q}_{k+1} - Q_{ref}],$$

where α is a control gain. For better scaling of D control for implementation, the predicted amount of surplus or slack traffic in the next time slot, $\hat{e}(k+1) = \hat{Q}_{k+1} - Q_{ref}$, is normalized by the surplus buffer capacity, $B - Q_{ref}$. Since the normalization factor, $B - Q_{ref}$, represents the buffer capacity for absorbing the transient surplus bursty traffic without losses, the normalized value represents the fraction of the amount of surplus/slack traffic to the surplus buffer capacity. The normalized D control equation is

$$p_d(k) = p_d(k-1) + \alpha \left[\frac{\hat{Q}_{k+1} - Q_{ref}}{B - Q_{ref}} \right]. \tag{15}$$

4. Simulation Study

In this section, we compare the control performance of the PI-PD controller with the other AQM algorithms such as the PID controller [7], RED [2], and the PI controller [18] in terms of the queue length dynamics and the packet loss rate via simulation over a wide range of traffic environments using the `ns-2` simulator [23]. First, we examine the control performance of AQM algorithms to different traffic load levels. Then, we evaluate the sensitivity of AQM algorithms to two different cases: (1) a case of a sudden increase of traffic load and (2) a case of increased roundtrip time.

4.1 Simulation Setup

We use a simple bottleneck network topology with two routers, `nc0` and `nc1`, and 9 TCP/Reno sources and 9 logically connected destinations.[7] These 9 pairs of TCP/Reno sources and destinations are connected to `nc0` and `nc1`, respectively. The link between `nc0` and `nc1` is assumed to have 30 Mbps of link speed and 10 msec of propagation delay. Since all TCP connections are connected to the routers, `nc0` and `nc1`, with link speed of 50 Mbps, the link between `nc0` and `nc1` is the only bottleneck. Each logically connected pair between the TCP source (`srci`) and the destination (`desti`), $i = 1, \cdots, 9$, is connected with propagation delays from 40 to 200 msec. We consider two types of traffic flows: an *elephant* long-lived FTP flow and *mice* short-lived flows. To obtain realistic TCP flows, n FTP flows and $2n$ with 1 second of average lifetime are connected to each `srci`. Then, 66% of TCP flows will be mice flows. The network setup is shown in Figure 6. The packet is assumed to have an average size of 1000 bytes. All sources and destinations are assumed to use TD queue management with sufficient buffer capacity. The buffer at the bottleneck link uses an AQM algorithm and has a capacity of 800 packets, which is twice the bandwidth-delay product (BDP).[8]

We compare the control performance of the PI-PD controller with that of the PID controller [7], RED [2], and the PI controller [18] under the packet drop mode. In RED, recommended parameter values [37] are used. In the PI controller, we use the parameters a, b and T_s, used in Hollot et al. [18]. In the PI-PD controller, the sampling time interval (T_s) must be selected so that the buffer does not experience overflow or underflow. The amount of packets that arrive in a T_s is $\lambda * T_s$, where λ is the packet arrival rate. Consider a design requirement that the queue length should not be increased/decreased by the amount of Q_{ref} in a T_s. Then, T_s should be less than Q_{ref}/C to prevent the buffer from underflow. On the other hand, the maximum number of queued packets in a T_s, $\lambda(1 - p_d) * T_s$, must be less than the buffer capacity (B) to prevent the buffer from overflow, where p_d is the packet drop probability. Thus, T_s must be selected so that the buffer does not experience overflow. For example, if the maximum packet queuing rate is assumed to be twice the link capacity (i.e., $\lambda(1-p_d) = 2 \cdot C = 7500$ packets/sec), in this network configuration, the maximum surplus traffic will be $3750 \cdot T_s$ packets. Therefore, to satisfy the design requirement, T_s should be less than $Q_{ref}/C = 200/3750 = 53.3$ msec. In this article, we use $T_s \simeq 50$ msec as a T_s for the PI-PD controller.

The amount of unit adjustment (i.e., unit increase or decrease) of p_d (or increase or decrease of queued packets) in a T_s is a function of α. Thus, the value of the

7. In general, Internet flows go through many routers and may experience congestion at some of those routers on the way from a source to its destination. However, if congestion at a router is more severe than congestion at other routers, the network topology can be simplified into a simple dumbbell shape having a single congested router. In this simplification, congestions at other routers can be considered as additional propagation delays for each flow. Therefore, in this study, we used a simple dumbbell network topology with different propagation delays ranging from 40 to 200 msec.

8. Since the average propagation delay is 120 msec, BDP is 450 packets (120 msec · 30 Mbps) and 2 · BDP = 900 packets. However, to compare AQM algorithms under the same network configuration used in Hollot et al. [18], we set the buffer size at 800 packets.

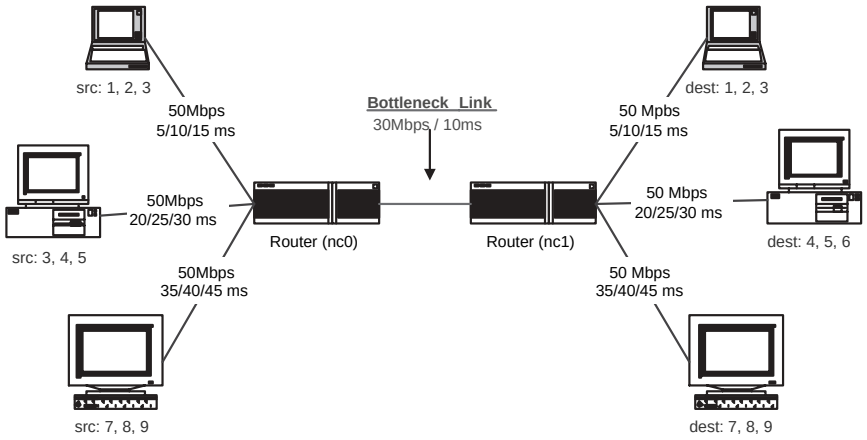

Figure 6. Simulation network topology

control gain is proportional to the buffer capacity (B). However, α should be determined by a trade-off between two conflicting design goals of an AQM, that is, the responsiveness (the short-term design goal) and the stability (the long-term design goal) to produce satisfactory control performance. We select a fractional value of the buffer size, $B \cdot 10^{-6} = 8.0 \cdot 10^{-4}$, as a control gain of the PI-PD controller empirically via extensive simulation studies.

In the PID controller [7], we set the desired maximum overshoot to 5% (i.e., $\xi = 0.6901$ equivalently) and the settling time (t_s) to the time to reach and stay within $\pm 2\%$ of the steady-state value in the design of a PD control part for a given network configuration. We set $R_0/2$ as the time constant for the design of the PD control part because it gives better control dynamics than R_0 in terms of speed of response and the steady-state error. Then, corresponding bounds of the transient performance specifications and PD control parameters are $\omega_n = 2/R_0\xi = 11.78$ radian/sec, $t_s \simeq 0.492$ sec, $K_{P1} = 1.166 \cdot 10^{-3}$, $K_{D1} = 9.97 \cdot 10^{-5}$. Then, $\omega_n^2 = 11.78^2 = (117187.3)(1.166 \cdot 10^{-3})K_{P2} = 136.7K_{P2}$.[9] Finally,

$$K_P = 1.24 * 10^{-3}, \quad K_D = 1.02 * 10^{-4}, \quad \text{and } K_I = 6.23 * 10^{-4}. \tag{16}$$

Although we can explore the key factors governing the TCP/AQM dynamics through simplified theoretical analysis, this simplification may introduce substantial error [38]. In general, three external signals—the reference input (Q_{ref}), load disturbance, and measurement noise—affect a control system [36]. However, the simplified TCP dynamic model (1) does not include load disturbances such as the slow-start and timeout mechanisms. Thus, PID control parameters (16) need to be tuned to generate proper control signals. Since simulation is used not only to check the correctness of analytic approaches but also for allowing exploration of complicated network situations that are either difficult or impossible to analyze [38], tuned PID control parameters and a proper sampling frequency can be found empirically via extensive simulation studies. From the tuning process via simulation study, we selected $\kappa = 0.05$ and $f_s = 10f = 29.5$ Hz as the tuning constant and sampling frequency, respectively, for digital implementation.

The desired queue length (Q_{ref}) for the PID controller, PI-PD controller, and PI controller is set to 200 packets. A summary of the parameter setting of each AQM algorithm is shown in Table 1.

Table 1. A summary of the parameter setting of each active queue management (AQM) algorithm

PID controller	$K_P = 6.2 \cdot 10^{-5}$, $K_D = 5.1 \cdot 10^{-5}$, $K_I = 3.12 \cdot 10^{-5}$, $f_s = 29.5$ Hz
PI-PD controller	$\alpha = 8.0 \cdot 10^{-4}$, $T_s = 50$ msec ($f_s = 20$ Hz),
RED	$w_Q = 0.002, max_p = 0.1$, $max_{th} = 200, min_{th} = 70$
PI controller	$a = 1.822 \cdot 10^{-5}, b = 1.816 \cdot 10^{-5}$, $Q_{ref} = 200$, $f_s = 160$ Hz

Note: In Hollot et al. [18], the frequency of the PI control system, $\omega_g = 0.53$ rad/sec, has been derived analytically, and the sampling frequency (f_s) for implementation has been recommended at $10 \sim 20$ times of the system frequency ($f = \omega_g/(2\pi)$). For example, under the network configuration of Figure 6 with the number of background flows, $N^- = 60$, the recommended sampling frequency is about 3 ~6 Hz in Hollot et al. [18]. However, a much faster sampling

9. See Ryu and Rump [7] for details of the design process for the PID controller.

frequency, 160 Hz, was used in Hollot et al. [18] for implementation of the PI controller via simulation. Through extensive simulation, it turns out that the PI controller with a lower sampling frequency (i.e., 3 $\sim$ 6 Hz) shows poor control performance. Therefore, in this study, we use the sampling frequency (160 Hz) that is higher than recommended (3 $\sim$ 6 Hz) to examine the control performance of the PI controller.

4.2 Performance Metrics

4.2.1 The Queue Length

The control performance of an AQM consists of two elements: the *transient control performance* (i.e., speed of response) and the *steady-state error control* (i.e., stability). We use the instantaneous queue length as a metric for the transient control performance. For the steady-state control performance, we use the quadratic average of control deviation (QACD) [33], defined as follows:

$$S_e = \sqrt{\frac{1}{N+1}\sum_{i=1}^{N}(Q_i - Q_{ref})^2}, \qquad (17)$$

where Q_i is the ith sampled queue length, $i = 1, \ldots, N$, and N is the number of sampling intervals.

4.2.2 The Packet Loss Rate

Since one of goals of an AQM algorithm is to remove bias against bursty sources, maintaining a stable packet loss rate is important. A high and bursty packet loss involves many packet losses at about the same time. If these packets belong to different flows, these flows experience losses about at the same time and then experience global synchronization as a result. On the other hand, if these packets belong to bursty sources, there exists bias against bursty sources. In general, the bias against bursty sources and the global synchronization can be eliminated effectively by achieving a stable and low packet loss rate over time. In addition, to achieve higher throughput (or goodput) or to accommodate more traffic, maintaining a low average packet loss rate is important.

4.3 Control Performance of AQM Algorithms

First, we examine the effect of the traffic load on the control performance of the PI-PD controller, PID controller, PI controller, and RED in terms of the queue length and the packet loss rate under two different traffic load levels (i.e., 189 flows and 378 flows consisting of 33% FTP and 67% mice flows).

4.3.1 The Queue Length Dynamics

Figure 7 shows the queue length dynamics of the PI-PD controller, PID controller, PI controller, and RED, respectively, under 189 flows (left) and 378 flows (right). The PI-PD controller and PID controller show good control performance under two different traffic load levels in terms of the queue length dynamics staying around $Q_{ref} = 200$ packets. As shown in Figure 7, the PI controller fails to maintain the queue length around Q_{ref}. Instead, the queue length stays below Q_{ref} most of time under all traffic load levels. Thus, the PI controller behaves like a TD having a buffer size of Q_{ref} with severe fluctuation. RED maintains the (average) queue length between $min_{th} = 70$ and $max_{th} = 200$ effectively under 189 flows. However, RED behaves like a TD with a buffer size of Q_{ref} under 378 flows, similar to the PI controller. This means that as the traffic load increases, RED behaves like a TD with a buffer size of max_{th}, as observed in Ott, Lakshman, and Wong [5].

Table 2. Summary of ANOVA analysis of traffic load factor on quadratic average of control deviation (QACD) of active queue management (AQM) algorithms

AQM Algorithms	F	p	F_{critical}
PI-PD controller	1.6426	0.2122	3.3541
PID controller	4.7159	0.0175	3.3541
PI controller	426.3	$3.76 \cdot 10^{-21}$	3.3541

The steady-state control performance of the PI-PD controller, PID controller, and PI controller is evaluated in terms of the QACD[10] for three different traffic load levels of 189, 270, and 378 flows. Table 2 shows a summary of the analysis of variance (ANOVA) of the traffic load factor on QACD of the PI-PD controller, PID controller, and PI controller. We run 10 different simulation runs for each combination of an AQM algorithm and a traffic load level. As shown in Table 2, the p value of the PI-PD controller is larger than the significance level (α) (i.e., $p = 0.2122 > \alpha = 0.05$). However, the p value of the PID controller and PI controller is smaller than α (i.e., $p = 0.0175 < 0.05$ and $p = 3.76 \cdot 10^{-21} \cong 0 < 0.05$). Thus, we conclude that the PI-PD controller shows satisfactory steady-state control performance in terms of QACD, independent of the traffic load level. However, the traffic load level significantly affects the steady-state control performance of the PID controller and PI controller in terms of QACD.

Figure 8 shows 95% confidence intervals of the QACD of the PI-PD controller, PID controller, and PI controller for each traffic load level. The PI-PD controller and PID controller show robust and stable control performance regardless of traffic load level. However, the PI controller shows traffic load-dependent performance with larger variance than the PI-PD controller and PID controller, and the steady-state control error (QACD) is reduced as the traffic load increases.

10. Since only the PI-PD controller and PI controller maintain the unique desired queue length, Q_{ref}, we use QACD to compare steady-state control performance of these three AQMs.

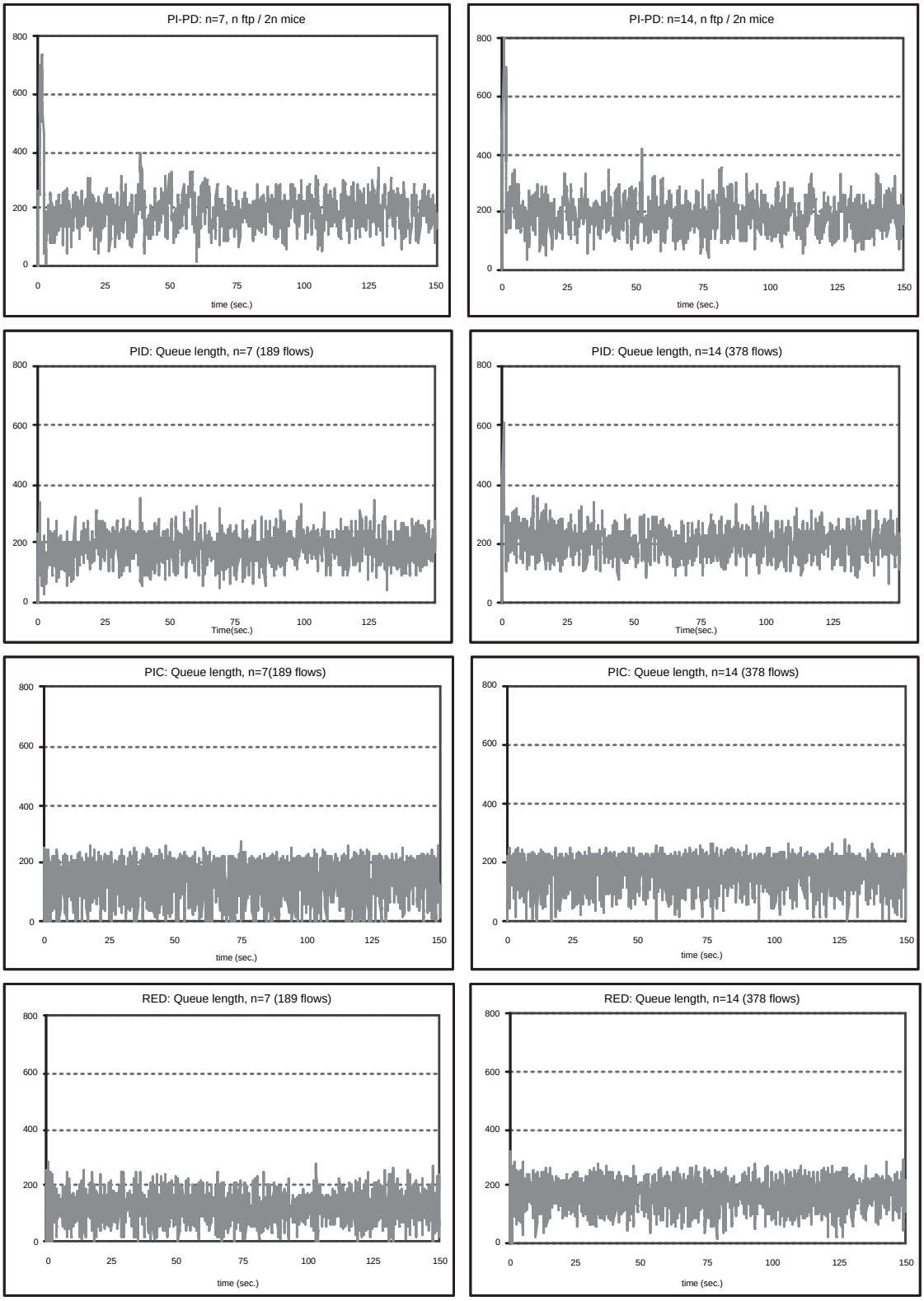

Figure 7. The queue lengths of the PI-PD controller, PID controller, PI controller, and RED under light ($n = 7$, left) and medium ($n = 14$, right) load levels

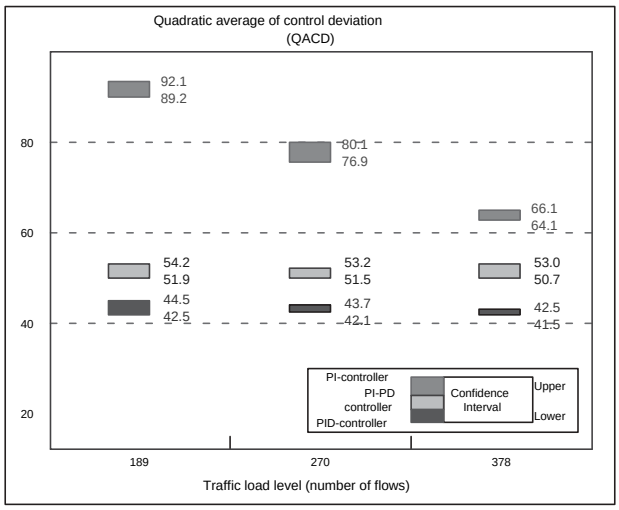

Figure 8. The quadratic average of control deviation (QACD) of the active queue management (AQM) algorithms for different traffic load levels

4.3.2 The Packet Loss Rate

Figure 9 shows the packet loss rates of the PI-PD controller, PID controller, PI controller, and RED, respectively, over time under the same traffic load levels used in Figure 7. The PI-PD controller and PID controller show stable and low packet loss rates over time for each traffic load level, which are slightly increased as the traffic load increases. RED shows stable and low packet loss rates under 178 flows, whereas the PI controller shows a high and severely fluctuating packet loss rate. However, the PI controller and RED show fluctuating and high packet loss rates under 378 flows. As a result, the PI-PD controller can remove the bias against bursty sources effectively, whereas the PI controller and RED may give bias against bursty sources because of multiple packet losses.

Figure 10 shows distributions of the frequencies of the packet loss rates of the PI-PD controller, PID controller, PI controller, and RED under 189 and 378 flows, respectively. As shown in Figure 10, most packet loss rates of the PI-PD controller, PID controller, and RED under 189 flows are distributed below 0.07, whereas a significant amount of the frequencies of the high packet loss rates appears with the PI controller. Thus, under 189 flows, the PI-PD controller, PID controller, and RED can remove the bias against bursty sources and/or global synchronization by maintaining low and stable packet loss rates. However, as the traffic load increases (under 378 flows), the packet loss behavior of RED changes significantly and gives high packet loss rates, whereas the PI-PD controller and PID controller show the same smooth and low packet loss pattern with a slightly larger average value. Thus, RED fails to remove bias against the bursty sources and may cause a global synchronization. Packet loss rates of the PI controller become higher and burstier as the traffic load increases.

Figure 11 shows the average packet loss probability and the link utilization of each AQM algorithm under different traffic load levels. Packet loss rates of the PI-PD controller and PID controller are stable and low (i.e., robust) over time and significantly lower than that of other AQM algorithm under all traffic loads. The link utilization of the PI-PD controller is almost the same as that of the PID controller and higher than that of the PI controller and RED.

4.3.3 Summary of Control Performance Study

In this experiment, we examine the control performance of the PI-PD controller, PID controller, PI controller, and RED under two different traffic load levels (i.e., 189 and 378 flows), consisting of 33% FTP and 67% mice flows. The PI-PD controller and the PID controller outperform the PI controller and RED in terms of the queue length dynamics and the packet loss rates. In particular, the PI-PD controller controls traffic effectively by maintaining the queue length around the desired queue length, Q_{ref}, with acceptable queue length deviation from Q_{ref} to avoid unnecessary packet drops. Also, the PI-PD controller shows robust steady-state control performance independent of traffic load level in terms of QACD.

In contrast, the queue length of the PI controller stays below Q_{ref} instead of being regulated around Q_{ref}. Thus, the PI controller behaves like a TD with a buffer size of Q_{ref}. The steady-state control performance of the PI controller is highly dependent on the traffic load level in terms of QACD. RED shows a good control performance under 189 flows by maintaining the queue length within the desired range $[min_{th}, max_{th}]$. However, as the traffic load increases, RED behaves like a TD with a buffer size of Q_{ref}, similar to the PI controller. Furthermore, the PI controller and RED give more multiple packet losses as the traffic load increases.

4.4 Sensitivity Analysis

In general, the control performance of an AQM algorithm is affected by several network components such as the buffer size (B), the link capacity (C), the traffic load factor (i.e., the number of flows [N]), and the roundtrip time (RTT). Since the buffer size and the link capacity are fixed when an AQM is installed in a router, these parameters are *static* (i.e., time-invariant) factors. In contrast, the traffic load factor (N) and the RTT are *dynamic* (i.e., time-varying) factors because they are changing dynamically over time. Thus, it is important for an AQM algorithm to have adaptive and robust control performance for the dynamic factors. In this section, we examine the sensitivity of the control performance of the PI-PD controller, PID controller, PI controller, and RED to the changes of network environments, particularly on the dynamic factors, the traffic load factor (N), and the RTT.

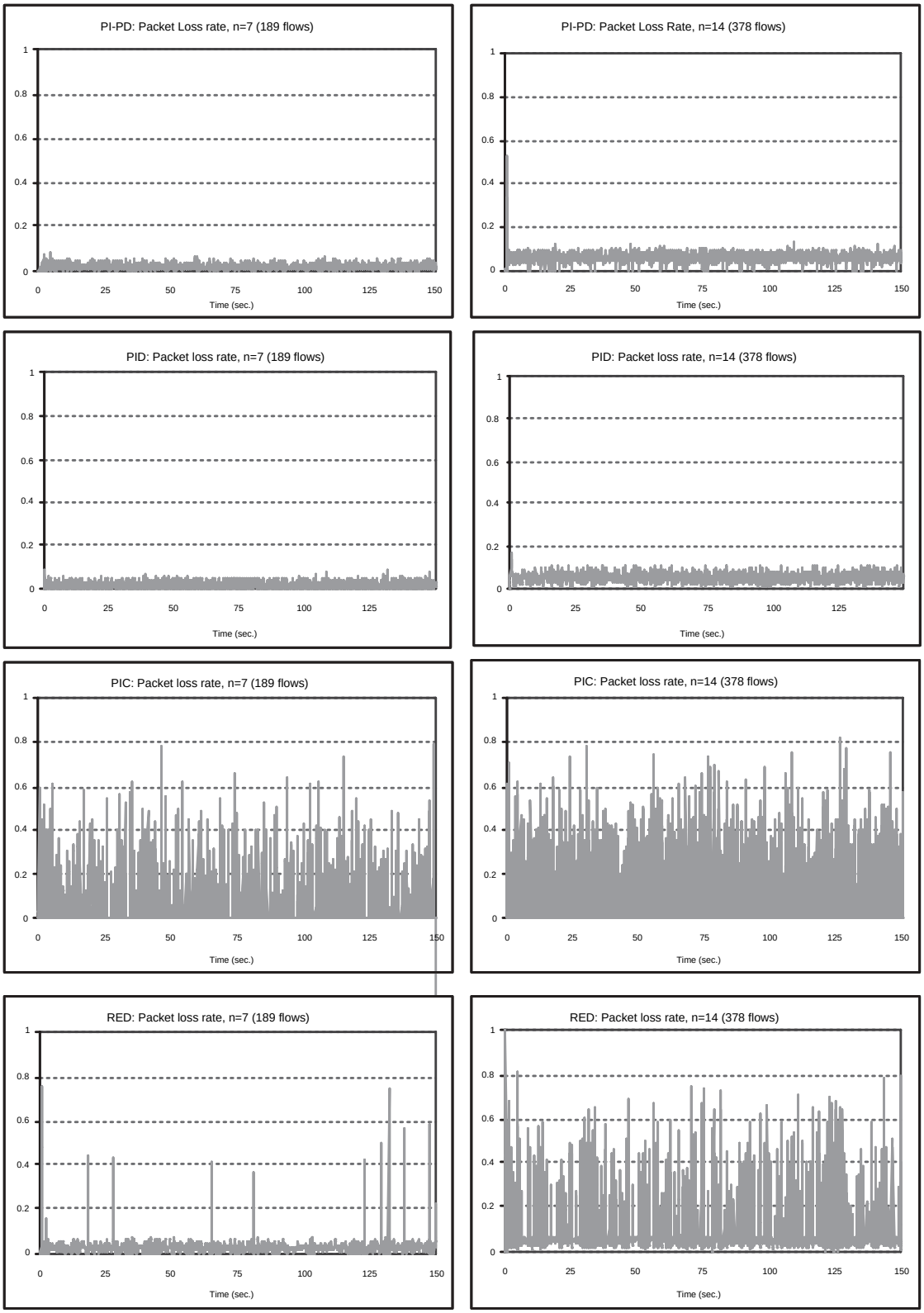

Figure 9. The packet loss rate of the PI-PD controller, PID controller, PI controller, and RED under 189 flows (left) and 378 flows (right)

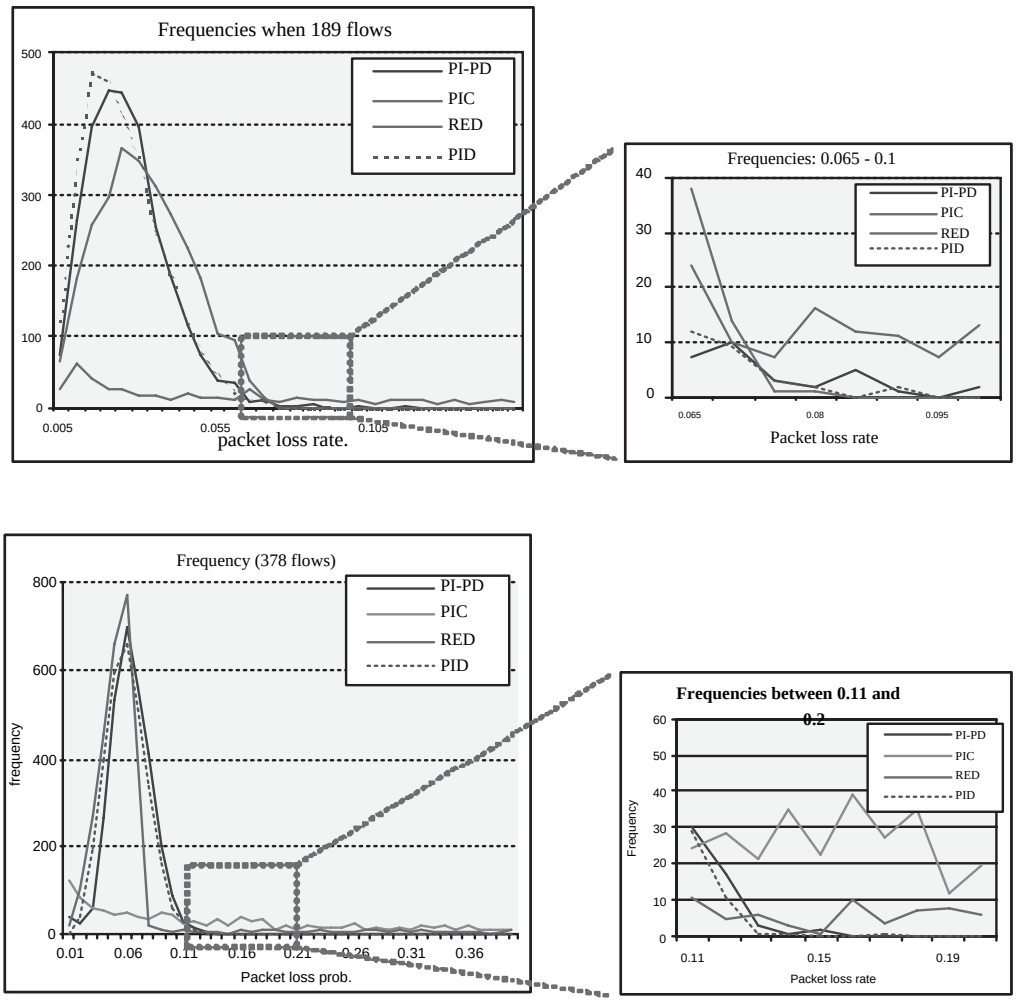

Figure 10. Distributions of the frequencies of the packet loss rate of the PI-PD controller, PID controller, PI controller, and RED under 189 and 378 flows

4.4.1 Sensitivity to the Number of Flows

In this experiment, we examine the control performance and adaptability of AQM algorithms to sudden traffic load changes over time. In Floyd and Kohler [39], it is shown that recently proposed AQM algorithms such as the PI controller and REM [40] perform poorly with traffic situations, where traffic consists of mostly web-like, short-lived mice traffic and the traffic load varies over time. Thus, we examine the control performance of the PI-PD controller, PID controller, PI controller, and RED under a traffic situation consisting of 75% mice flows and 25% elephant flows, with the traffic load varying over time. The simulation begins with a light traffic load that consists of three sources spawning 30 FTP and 90 mice flows. Thus, the initial total number of flows is 120. Then, an additional three sources (i.e., 30 FTP and 90 mice flows) are added at time 50.0 sec. Thus, from time 50.0, the traffic consists of six sources with 240 flows (60 FTP and 180 mice). Finally, an additional three sources (i.e., 30 FTP and 90 mice flows) are added at time 100.0 sec. Thus, from time 100.0, the traffic consists of nine sources with 360 flows (90 FTP and 270 mice). Figure 12 shows the above traffic scenario.

Figure 13 shows the control dynamics of AQM algorithms in terms of the queue length and the packet loss rate with respect to time. We analyze the control dynamics of AQM algorithms in terms of the queue length first. The queue length of the PI-PD controller stays around Q_{ref}, independent of the traffic load levels. The PI-PD controller shows transient overshoots of the queue length when the traffic load is increased suddenly at time 50.0 sec and 100.0 sec. However, these transient responses vanish rapidly, and the queue length stays around Q_{ref} most of time. The queue

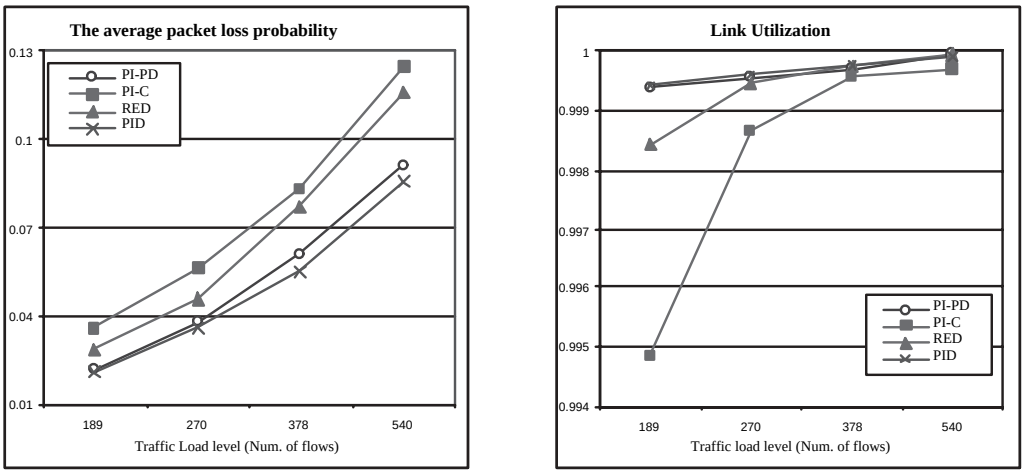

Figure 11. The average packet loss probabilities and the link utilization of active queue management (AQM) algorithms under several different load levels

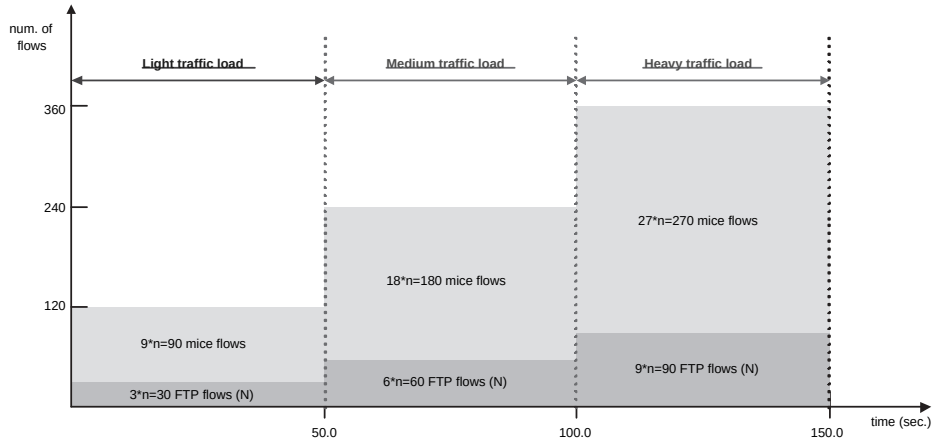

Figure 12. A scenario of varying traffic load over time

length of the PID controller stays around Q_{ref}, except during the time interval [0.0, 50.0] sec, because the number of elephant (FTP) flows (30) in this time interval is smaller than the lower bound ($N^- = 60$) assumed in (1) [7]. The queue length of the PI controller stays below Q_{ref} and fluctuates over time. In particular, under a light traffic load (i.e., in the time interval [0.0, 50.0] sec), the queue length severely fluctuates and shows underutilization of the link by allowing the link to idle occasionally. RED maintains the queue length within the range between $min_{th} = 70$ and $max_{th} = 200$. However, as the traffic load increases, the (average) queue length approaches $Q_{ref} = 200$ and stays around Q_{ref}, especially under heavy traffic loads (i.e., after 100.0 sec).

In terms of the packet loss rate, the PI-PD controller and PID controller show low and stable dynamics under this traffic situation. In particular, the PI-PD controller and PID controller can effectively prevent the buffer from multiple packet losses by allowing acceptable queue length deviation from Q_{ref}. On the other hand, the PI controller allows multiple packet losses over time, and it becomes more frequent as the traffic load increases. RED is able to maintain stable and low packet loss rates under light traffic (i.e., in the time interval [0.0, 50.0] sec). However, as the traffic load increases, RED allows more multiple packet losses.

Figure 14 shows the average packet loss rates of the PI-PD controller, PID controller, PI controller, and RED

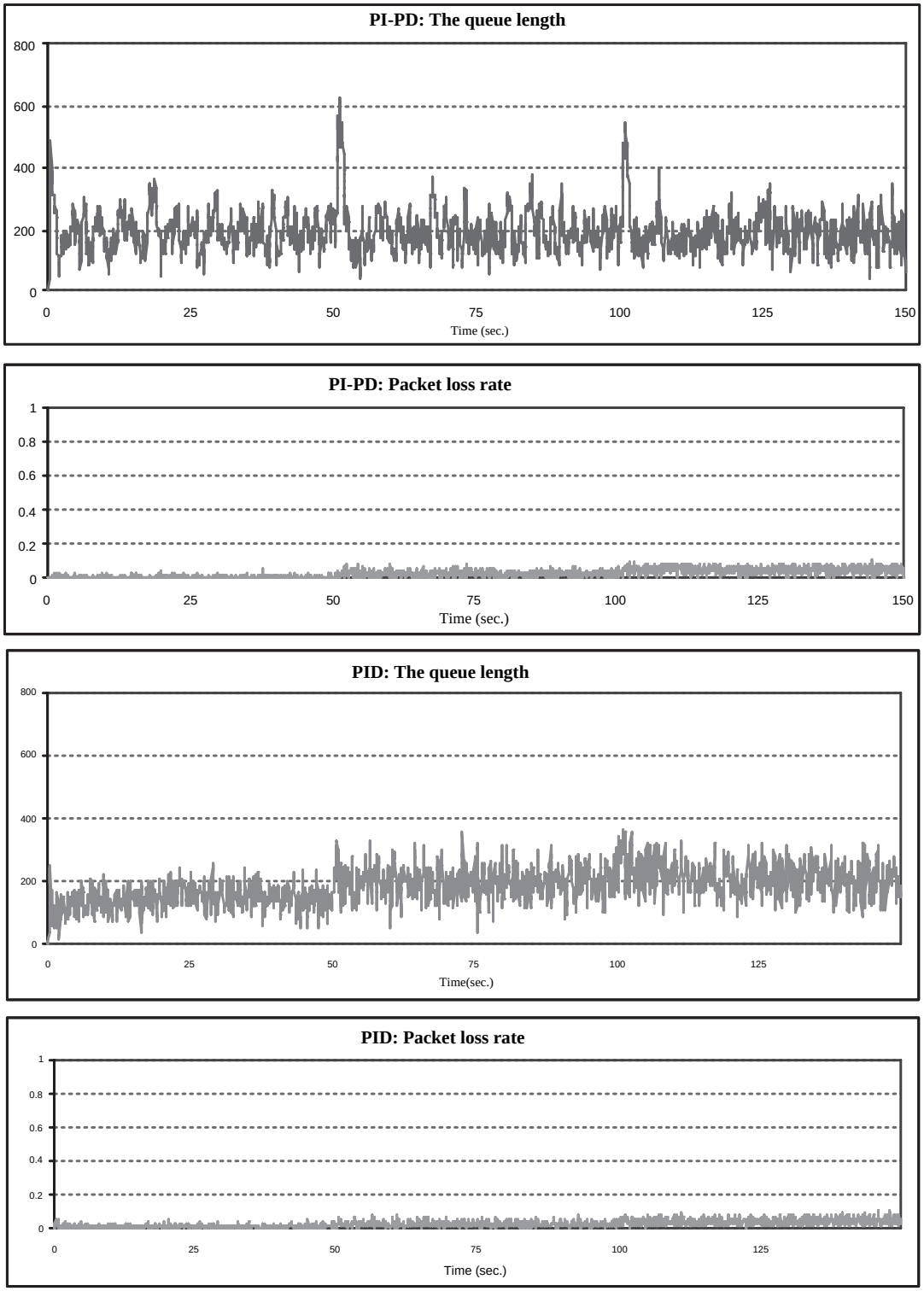

Figure 13. The queue length and packet loss rates of active queue management (AQM) algorithms over time *(continued on next page)*

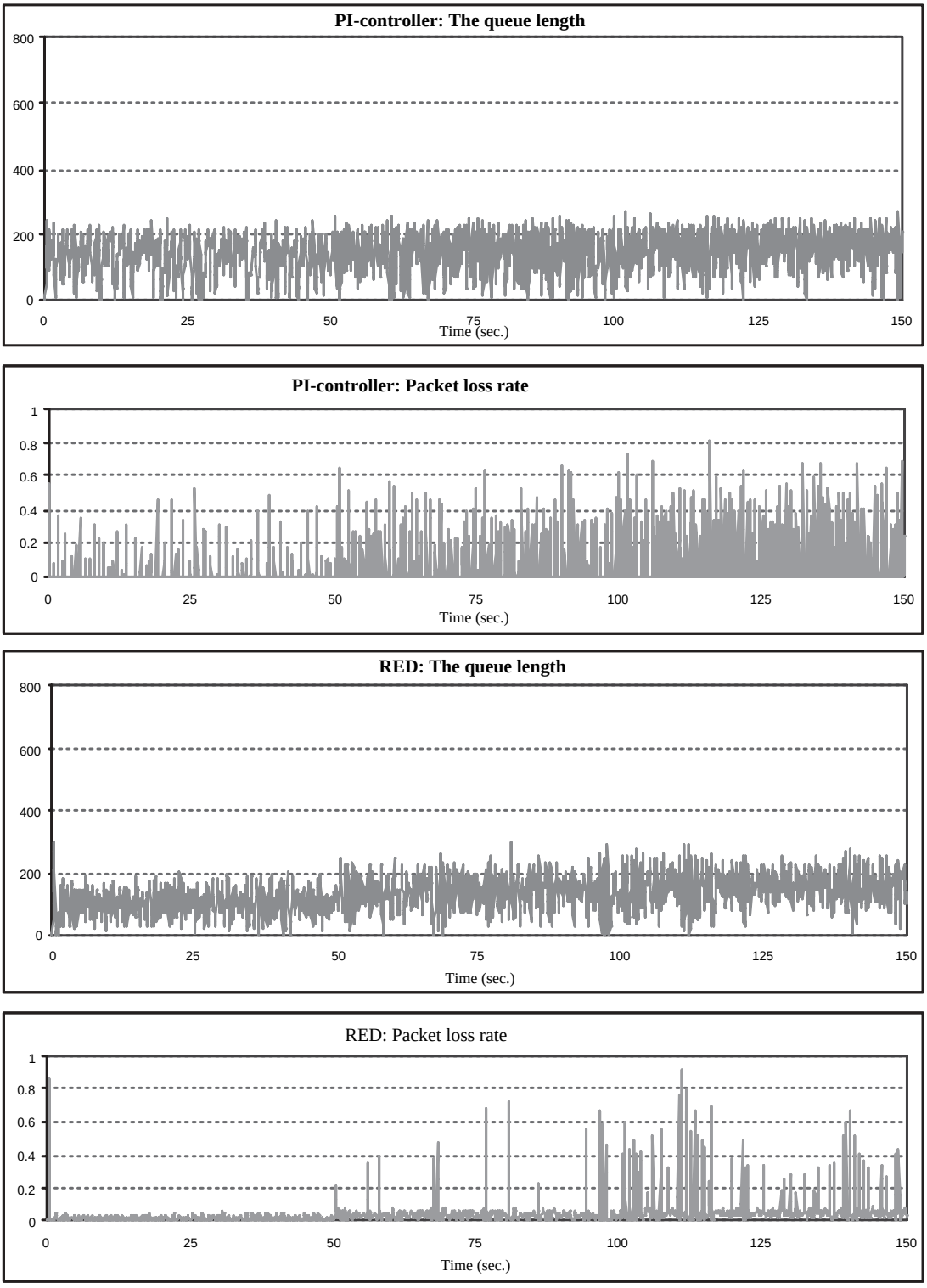

Figure 13. *(continued from previous page)*

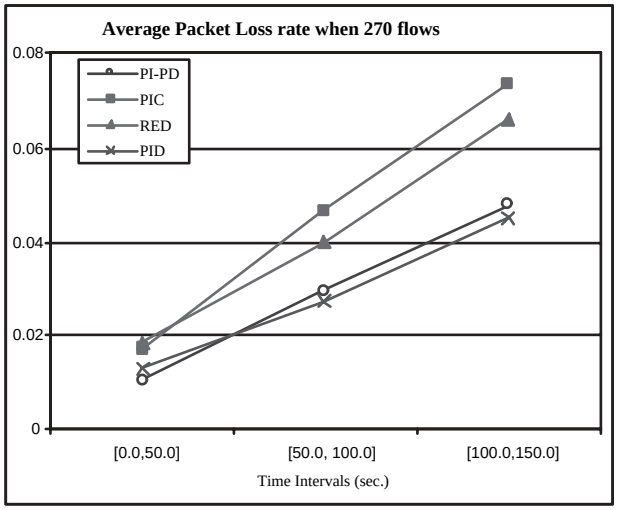

Figure 14. The average packet loss probability in each time interval

algorithms for three different time intervals: [0.0, 50.0], [50.0, 100.0], and [100.0, 150.0]. The average packet loss rate of the PI-PD controller and PID controller is shown to be almost the same under three time intervals and lower than those of the PI controller and RED under all time intervals. Therefore, for the same average packet loss rate requirement, the network can accommodate more traffic with the PI-PD controller (and the PID controller) than with the PI controller and RED.

4.4.2 Sensitivity to the Roundtrip Time

An increase of RTT not only degrades the control performance of an AQM algorithm but also leads the system to fall into instable status. Thus, the effect of RTT should be taken into account in designing an AQM that is robust, especially in wide-area networks (WAN) environments. In this experiment, we examine the effect of an increase of RTT on the control performance of the PI-PD controller, PID controller, PI controller, and RED. We use the same network configuration shown in Figure 6, except that the link delay between routers `nc0` and `nc1` is increased from 10 to 60 msec. Thus, RTTs of all pairs of sources and destinations are increased from [93.3, 253.3] to [193.3, 353.3] msec.

Figure 15 shows the queue length dynamics of the PI-PD controller, PID controller, PI controller, and RED under 270 flows. The control performance of the PI controller and RED is very sensitive to the increase of RTT and may lead the system to fall into instable status. The PI-PD controller and PID controller are less sensitive to the increase of RTT than the PI controller and RED, and they regulate the queue length around Q_{ref} with a slightly larger deviation than for the case of a smaller RTT.

Figure 16 shows the average packet loss rate of the PI-PD controller, PID controller, PI controller, and RED algorithms to the increased RTT under different traffic load levels. Both the PI-PD controller and PID controller outperform the PI controller and RED under all traffic load levels. In particular, by allowing a slightly larger deviation than the PID controller, the PI-PD controller is able to achieve a lower average packet loss rate than the PID controller. The PI-PD controller shows significantly lower average packet loss rates than the PI controller and RED under all traffic load levels. Moreover, the difference on the average packet loss rate between the PI-PD controller and other AQM algorithms such as the PI controller and RED becomes larger as traffic load increases.

4.4.3 Summary of the Sensitivity Analysis

In the sensitivity analysis, we first examine adaptability and sensitivity of the PI-PD controller, PID controller, PI controller, and RED to suddenly increased traffic loads over time. The PI-PD controller and PID controller show a satisfactory transient and steady-state control performance in terms of the queue length dynamics and the packet loss rates. In addition, these two AQM algorithms, designed based on the classical PID control mechanism, are able to maintain a low and stable packet loss rate over time. The control performance of the PI controller is sensitive to the traffic load factor (N), especially to the number of FTP flows. The PI controller shows fluctuating queue length dynamics below Q_{ref} and multiple and high packet losses over time. RED is able to maintain the queue length within the desired range of $[min_{th}, max_{th}]$ and shows stable and low packet loss. However, as the traffic load increases, RED gives more multiple and high packet losses and behaves like a TD with a buffer size of Q_{ref}.

Second, we examine the sensitivity of the control performance of AQM algorithms to the increased RTT. The PI-PD controller and PID controller outperform the PI controller and RED in terms of the queue length dynamics and the packet loss rate. In other words, the PI-PD controller and PID controller are less sensitive to the changes on RTT than the PI controller and RED.

5. Conclusion

It is necessary for an AQM-based congestion control algorithm to control congestion adaptively under a wide range of traffic loads to provide an acceptable quality of service (QoS) such as a bounded and stable delay, a low packet loss rate, and a high link utilization. We have outlined requirements for an AQM to avoid and/or control congestion adaptively to dynamically changing traffic loads. These requirements include an ability to detect and control congestion *proactively* based on the incipient congestion, not *reactively* based on the current congestion.

We designed an adaptive and proactive AQM algorithm, called the *PI-PD controller*, using the concept of classical

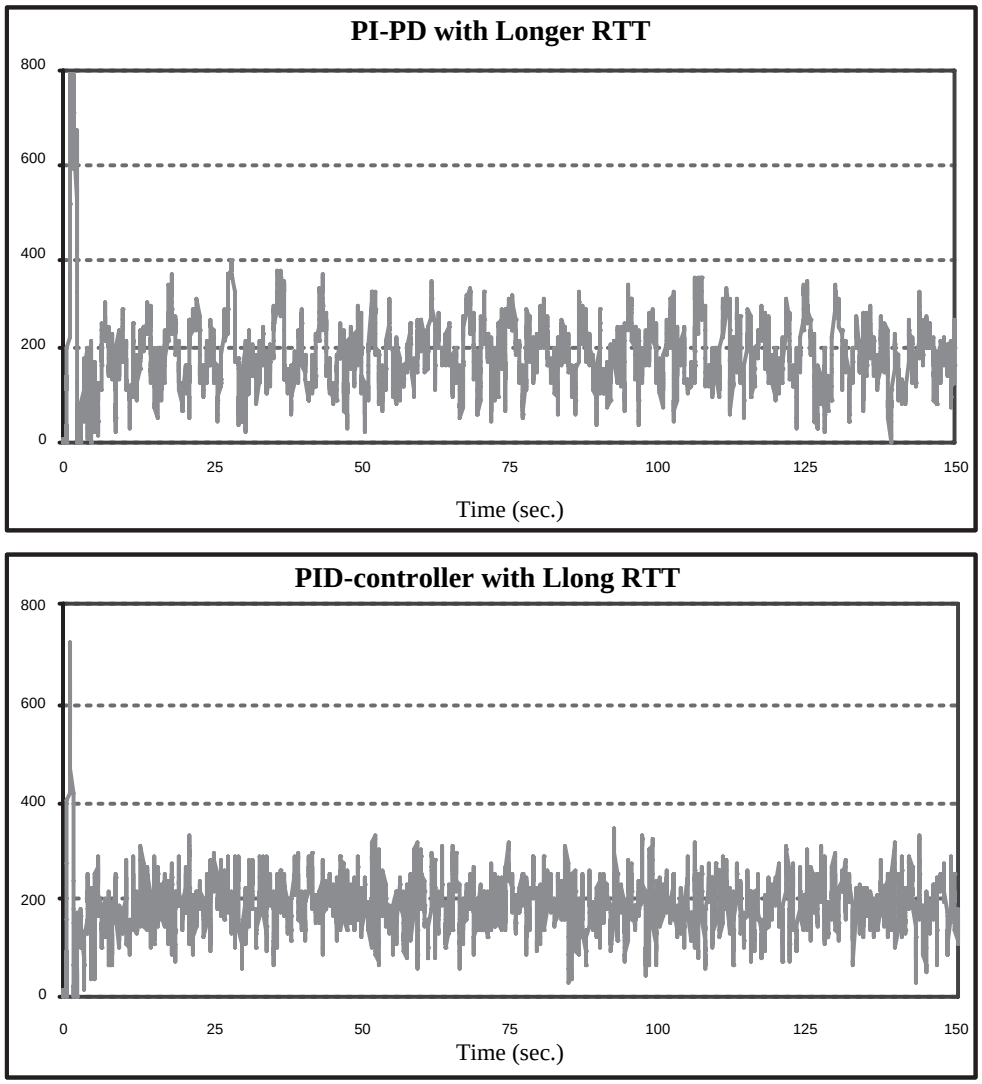

Figure 15. The queue lengths of the PI-PD controller, PID controller, PI controller, and RED with 270 flows and longer roundtrip time (100 msec) *(continued on next page)*

PID feedback control. On one hand, with the introduction of a PD control, the PI-PD controller was able to achieve the short-term system performance such as fast response and proactive control to the changing traffic load via anticipatory traffic prediction and control. On the other hand, with the introduction of a PI control, the PI-PD controller was also able to achieve long-term performance such as the elimination of steady-state error. In the PI controller, the unit sampling time interval for digital implementation is determined by the system frequency, that is, the TCP flow dynamic model (1). Unlike the PI controller, the PI-PD controller does not rely on assumptions on the plant dynamic model. For example, the unit sampling time interval of the PI-PD controller is determined by the buffer size and the input traffic load, independent of the plant dynamic model.

Control performance of the proposed the PI-PD controller has been examined and compared with that of other AQM algorithms, such as the PI controller and RED, under a variety of traffic situations via extensive simulation studies using the `ns-2` simulator. In addition, to take advantage of a well-designed continuous PID controller, the control performance of the PI-PD controller also has been compared with that of the PID controller [7], developed

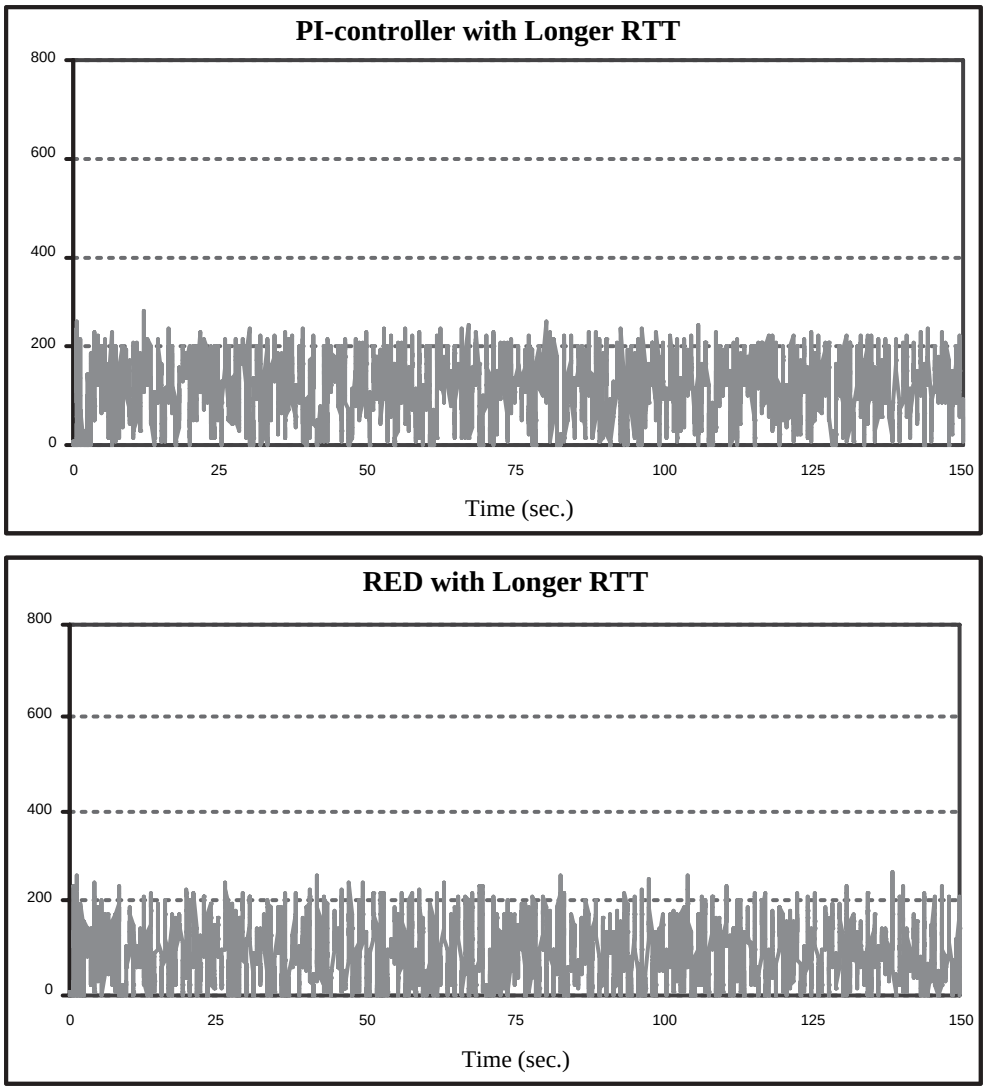

Figure 15. *(continued from previous page)*

based on TCP flow dynamics (1) and digitized by emulation. The PI-PD controller shows robust and adaptive congestion control performance to a variety of traffic situations in extensive simulation studies. In particular, the PI-PD controller outperforms other AQM algorithms such as RED and the PI controller in terms of the queue length dynamics, the packet loss rates, and the link utilization. The steady-state control dynamics (QACD) of the PI-PD controller is robust and independent of the traffic load. Moreover, the PI-PD controller shows a robust control performance to the changes of network environments, such as the traffic mix, the traffic load, and the roundtrip time. In contrast, the PI controller and RED show very sensitive control performance to the changes of the above network environments.

In practical implementation, the PI-PD controller has comparably little computational overhead. In RED, the EWMA queue length and the packet drop probability are calculated at every packet arrival while maintaining several parameters such as w_Q, min_{th}, max_{th}, max_p, and so on. In contrast, the PI-PD controller can be easily implemented with less sampling frequency (20 Hz) compared to the link speed (i.e., 3750 Hz) implementation of RED and 160 Hz of the PI controller while maintaining fewer parameters

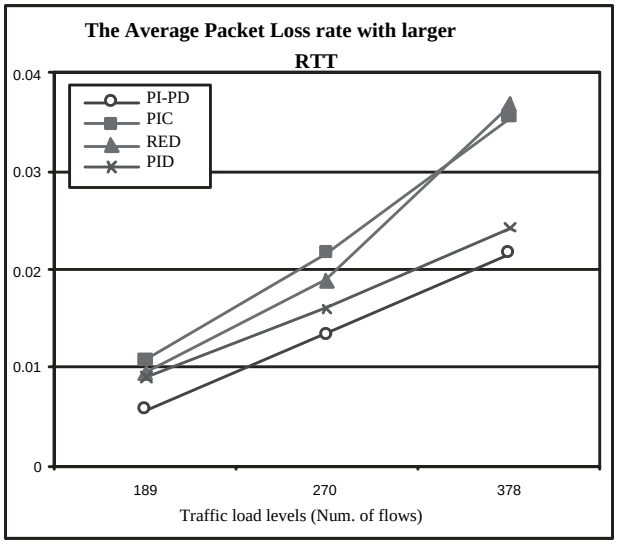

Figure 16. The average packet loss probability with longer roundtrip time

than RED. Therefore, the computational complexity and overhead at a router can be reduced significantly with the PI-PD controller.

There are several issues for further study. First, by examining the impact of the queue length sampling frequency (or, equivalently, the length of the time slot value, T) on the performance of the PI-PD controller, we hope to find a relationship between the optimal T value and the offered traffic load. Then, the PI-PD controller can be implemented with an adaptive sampling time interval to the dynamically changing traffic situations. We are also working on finding the optimal control gain, α, and the stability margin of the PI-PD controller control through control-theoretic modeling and analysis. In this study, we focused only on the end-to-end congestion control of TCP traffic. Control of the traffic consisting of TCP and UDP flows will be another future study issue.

6. References

[1] Braden, B., J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, L. Zhang. 1998. Recommendations on queue management and congestion avoidance in the Internet. Rep. RFC2309.

[2] Floyd, S., and V. Jacobson. 1993. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking* 1 (4): 397-413.

[3] Feng, W., K. G. Shin, D. D. Kandlur, and D. Saha. 2002. The BLUE active queue management algorithms. *IEEE/ACM Transactions on Networking* 10 (4): 513-28.

[4] Feng, W., D. D. Kandlar, D. Saha, and K. G. Shin. 1999. A self-configuring RED gateway. *Proceedings of INFOCOM'1999*, pp. 1320-8.

[5] Ott, T. J., T. V. Lakshman, and L. Wong. 1999. SRED: Stabilized RED. *Proceedings of INFOCOM'1999*, pp. 1346-55.

[6] Ryu, S., C. Rump, and C. Qiao. 2003. Advances in Internet congestion control. *IEEE Communication Survey and Tutorial* 5 (1): 28-39.

[7] Ryu, S., and C. Rump. 2004. Application of a PID feedback control algorithm for adaptive queue management to support TCP congestion control. *Journal of Communications and Networks* 6 (2): 133-46.

[8] Christiansen, M., K. Jaffey, D. Ott, and D. Smith. 2001. Tuning RED for web traffic. *IEEE/ACM Transactions on Networking* 9 (3): 249-64.

[9] Zhang, Y., and L. Qiu. 2000. Understanding the end-to-end performance impact of RED in a heterogeneous environment. Technical Rep. 2000-1802, Cornell University.

[10] Guo, L., and I. Matta. 2001. The war between mice and elephants. *Proceedings of IEEE ICNP'01*, pp. 180-8.

[11] Joo, C., S. Bahk, and S. Lumetta. 2003. Hybrid active queue management. *Proceedings of ISCC'2003*, pp. 1005-11.

[12] Katabi, D., M. Handley, and C. Rohrs. 2002. Congestion control for high bandwidth-delay product networks. *Proceedings of SIGCOMM'2002*, pp. 89-102.

[13] Quet, P., and H. Özbay. 2004. On the design of AQM supporting TCP flows using robust control theory. *IEEE Transactions on Automatic Control* 49 (6): 1031-6.

[14] Yan, P., Y. Gao, and H. Özbay. 2003. Variable structure in active queue management for TCP with ECN. *Proceedings of ISCC'2003*, pp. 1012-7.

[15] Fengyuan, R., and L. Chuang. 2003. Speed up the responsiveness of active queue management system. *IEICE Transactions on Communications* E86-B (2): 630-6.

[16] Heying, Z., L. Baohong, and D. Wenhua. 2003. Design of a robust active queue management algorithm based on feedback compensation. *Proceedings of SIGCOMM'2003*, pp. 277-85.

[17] Yanfei, F., R. Fengyuan, and L. Chuang. 2003. Design a PID controller for active queue management. *Proceedings of ISCC'2003*, pp. 985-90.

[18] Hollot, C. V., V. Misra, D. Towsley, and W. Gong. 2002. Analysis and design of controllers for AQM routers supporting TCP flows. *IEEE/ACM Transactions on Automatic Control* 47 (6): 945-59.

[19] Franklin, G., J. Powell, and M. Workman. 1998. *Digital control of dynamic systems*. 3rd ed. Reading, MA: Addison-Wesley.

[20] Ryu, S., and C. Cho. 2004. PI-PD-controller for robust and active queue management for supporting TCP congestion control. *Proceedings of the Annual Simulation Symposium'2004*, pp. 132-9.

[21] Franklin, G., J. Powell, and A. Emami-Naeini. 1995. *Feedback control of dynamic systems*. 3rd ed. Reading, MA: Addison-Wesley.

[22] Kunniyur, S., and R. Srikant. 2004. An adaptive virtual queue (AVQ) algorithm for active queue management. *IEEE/ACM Transactions on Networking* 12 (2): 286-99.

[23] McCanne, S., and S. Floyd. 1996. Network Simulator—ns (version 2). http://www.isi.edu/nsnam/ns

[24] Misra, V., W. Gong, and D. Towsley. 2000. Fluid-based analysis of a network of AQM routers supporting TCP flows with an application to RED. *Proceedings of ACM SIGCOMM2000*, pp. 151-60.

[25] Hollot, C. V., V. Misra, D. Towsley, and W. Gong. 2001. A control theoretic analysis of RED. *Proceedings of INFOCOM'2001*, pp. 1510-9.

[26] Kuo, B. C. 1995. *Automatic control systems*. 7th ed. New York: John Wiley.

[27] Aweya, J., M. Ouellette, and D. Y. Montuno. 2001. A control theoretic approach to active queue management. *Computer Networks* 36 (2-3): 203-35.

[28] Misra, A., T. Ott, and J. Baras. 2001. Effect of exponential averaging on the variability of a RED queue. *Proceedings of IEEE ICC'2001*, pp. 1817-23.

[29] Ziegler, T. 2002. On averaging for active queue management congestion avoidance. *Proceedings of ISCC'2002*, pp. 867-73.

[30] Wilson, D. I. 2001. *Advanced control*. http://www.ee.kau.se/forskning/ModSim/cnotes.pdf

[31] Allman, M., V. Paxson, and W. Stevens. 1999. TCP congestion control. Rep. RFC2581.
[32] Åstrom, K., and T. Hagglund. 2001. The future of PID control. *Control Engineering Practice* 9:1163-75.
[33] Isermann, R. 1989. *Digital control systems: 1. Fundamentals, deterministic control*. 2nd rev. ed. New York: Springer-Verlag.
[34] Cao, J., W. S. Cleveland, D. Lin, and D. X. Sun. 2001. On the nonstationarity of Internet traffic. *Proceeding of ACM SIGMETRICS 2001*, pp. 102-12.
[35] Cao, J., W. S. Cleveland, D. Lin, and D. X. Sun. 2001. Internet traffic tends to Poisson and independence as the load increases. Technical Report, Bell Labs, Murray Hill, NJ, April 2001.
[36] Åstrom. K., and T. Hagglund. 1995. *PID controllers: Theory, design, and tuning*. 2nd ed. Instrument Society of America.
[37] Floyd, S. 1996. Notes on testing RED implementation. http://www.icir.org/floyd/papers/redtesting
[38] Floyd, S., and V. Paxson. 2001. Difficulties in simulating the Internet. *IEEE Network Magazine* 9 (4): 392-403.
[39] Floyd, S., and E. Kohler. 2002. Internet research needs better models. In *First Workshop on Hot Topics in Networks (HotNets-I)*. http://www.acm.org/sigcomm/HotNets-I
[40] Low, S. H., and D. Lapsley. 1999. Random early marking: An optimization approach to Internet congestion control. In *Proceedings of IEEE ICON'99*, pp. 67-74.

Seungwan Ryu *is is an assistant professor in the Department of Information Systems at Chung-Ang University, Korea, and an Invited Researcher in the Mobile Telecommunications Research Division at ETRI, Korea.*

Byunghan Ryu *is a team leader of the Broadband Mobile MAC Research Team in the Mobile Telecommunications Research Division at ETRI, Korea.*

Myoungki Jeong *is a research team leader at the Telecom R&D Center, Samsung Electronics, Suwon, Korea.*

Seikwon Park *is a professor in the Department of Information Systems at Chung-Ang University, Korea.*