A Hybrid Active Queue Management for Stability and Fast Adaptation

Changhee Joo, Saewoong Bahk, and Steven S. Lumetta

Abstract: **The domination of the Internet by TCP-based services has spawned many efforts to provide high network utilization with low loss and delay in a simple and scalable manner. Active queue management (AQM) algorithms attempt to achieve these goals by regulating queues at bottleneck links to provide useful feedback to TCP sources. While many AQM algorithms have been proposed, most suffer from instability, require careful configuration of non-intuitive control parameters, or are not practical because of slow response to dynamic traffic changes.**

In this paper, we propose a new AQM algorithm, hybrid random early detection (HRED), that combines the more effective elements of recent algorithms with a random early detection (RED) core. HRED maps instantaneous queue length to a drop probability, automatically adjusting the slope and intercept of the mapping function to account for changes in traffic load and to keep queue length within the desired operating range. We demonstrate that straightforward selection of HRED parameters results in stable operation under steady load and rapid adaptation to changes in load. Simulation and implementation tests confirm this stability, and indicate that overall performances of HRED are substantially better than those of earlier AQM algorithms. Finally, HRED control parameters provide several intuitive approaches to trading between required memory, queue stability, and response time.

Index Terms: **Active queue management (AQM), random early detection (RED), response time, stability.**

I. INTRODUCTION

Active queue management (AQM) algorithms help end hosts make decisions about transmission rates by providing congestion information based on characteristics of a router's queue. The advantages of such feedback seem obvious, and the IETF recommends the use of AQM to reduce loss rate, to support low-delay interactive services, and to avoid TCP lock-out behavior [1]. The difficulty of configuring AQM algorithms for stable operation in dynamic networking environments, however, has prevented them from replacing the traditional tail-drop mechanism in the Internet [2], [3].

A successful AQM algorithm must provide simple, intuitive means of configuring parameters to achieve stable operation under steady load and rapid adaptation to changes in load. Large oscillations in queue length, for example, typically result in periods during which the queue is empty, reducing the utilization of the link. Similarly, an algorithm that fails to adapt to changes in traffic load or adapts too slowly may perform poorly in the Internet environment, which is highly dynamic [4]. Finally, an algorithm capable of meeting the stability and response criteria may remain unused if control parameters are non-intuitive or inadequate for achieving desired goals. A system may target a particular queue length based on available memory, or an administrator may wish to trade between stability and response time; in either case, the algorithm must provide simple means to attain these ends.

The most well-known AQM algorithm is the random early detection (RED) gateway [5]. Stochastic drops allow RED to avoid global synchronization and bias against bursty traffic when used in conjunction with TCP-based flows regulated by additive increase/multiple decrease (AIMD) congestion algorithm [6]. RED can also be used in combination with explicit congestion notification (ECN) [7], in which case RED marks packets instead of dropping them. For simplicity, we use the term *drop* to mean either *drop* or *mark* in the remainder of our paper.

Despite RED's advantages over tail-dropping, the difficulty of configuring RED for deployment in a variety of networking environments has prevented its widespread use. In particular, RED's vulnerability to oscillations under steady load has made it much less attractive [3], [8]–[10].

Numerous AQM algorithms have been proposed to eliminate the drawbacks associated with RED while retaining its advantages. These efforts include stabilized RED [11], adaptive RED [12], Blue [13], the PI controller (PIC) [14], random exponential marking (REM) [15], and the adaptive virtual queue (AVQ) [16]. In general, these algorithms calculate drop probability from measurements of queue length and packet arrival rate, with the goal of ensuring stable operation in dynamic environments. As more recent algorithms (e.g., PIC, REM, and AVQ) are known to be stable, the capability to control the queueing delay, the response time to load changes, and the intuitive value of an algorithm's parameters become the primary metrics for evaluation.

In this paper, we introduce and evaluate a new AQM algorithm that combines a RED design with aspects of other AQM algorithms, such as Blue and the proportional controller [14]. For this reason, we call it hybrid RED, or HRED. HRED extends RED with a second probability parameter to provide stability under steady load [17]. It utilizes a self-configuring adjustment algorithm to adapt to changes in load. The result is stable operation with rapid response to change. HRED can be easily configured to achieve demanded performance of queueing delay, jitter, and response time through its parameters that are more intuitively meaningful than other AQM algorithms.

Manuscript received January 3, 2004; approved for publication by Tony Lee, Division III Editor, November 16, 2005.

C. Joo and S. Bahk are with the School of Electrical Engineering and INMC, Seoul National University, Seoul, Korea, email: {cjoo, sbahk}@netlab.snu.ac.kr.

S. Lumetta is with the Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, IL, USA, email: lumetta@uiuc.edu.

The remainder of the paper is organized as follows. We begin with an overview of previous work in Section II, briefly covering a few principles and defining common terminology and variables. In Section III, we outline the HRED algorithm and explain with a simple analysis how its design satisfies the criteria stated earlier for a successful AQM algorithm. We simulate HRED and other AQM algorithms in Section IV to evaluate their behaviors in steady load and in response to load changes. The results of this section demonstrate that our analytic assessment of HRED is valid in the simulation environment. We present results of implementation tests in Section V. After discussing other relating issues in brief, and we conclude in Section VII.

II. PREVIOUS WORK

RED, as proposed in [5], uses two operations to calculate drop probability: it first computes an average queue length ℓ using an exponential weighted moving average (EWMA) with weight w_q, then calculates drop probability p from ℓ using a fixed linear mapping function. The mapping function has three parameters: $\min_{th}$, $\max_{th}$, and $p_{\max}$. Average queue lengths in the operating range $[\min_{th}, \max_{th}]$ are linearly mapped to drop probabilities in the range $[0, p_{\max}]$. Below $\min_{th}$, packets are not dropped, and above $\max_{th}$, they are always dropped. The simplest modification of RED to improve stability is to use the `gentle_` option [18], which eliminates the discontinuity at $\max_{th}$ and makes the drop probability function piecewise linear by mapping the interval between the maximum threshold and the buffer size, $[\max_{th}, Q_{\max}]$ (or $[\max_{th}, 2\max_{th}]$), linearly to the range $[p_{\max}, 1]$.

Rather than expanding the linear range, adaptive RED (ARED) adjusts $p_{\max}$ [12]. It increases $p_{\max}$ when ℓ gets over $\max_{th}$ and decreases when ℓ gets below $\min_{th}$. ARED attempts to decouple queue length from drop probability to adapt to a range of network scenarios, but can not completely remove instability due to steep slope of the mapping function [17].

The developers of ARED also proposed a new AQM algorithm called Blue [13]. Blue neither averages queue length nor uses thresholds. Instead, it drops packets according a probability independent of queue length, and adjusts the probability in response to buffer overflow (tail-drop) and underflow (empty) events. Blue can be implemented very simply, and requires very little processing power.

PIC applies classical control theory to obtain stability [14]. Based on analysis with linearized models of TCP and RED [8], PIC is proven to be locally stable. It maintains local stability by guaranteeing gain and phase margins of the system function. REM has the same control mechanism as PIC but, instead of directly controlling the drop probability, it controls *price*, which is mapped to the drop probability by exponential function [15].

AVQ maintains a virtual queue with a virtual capacity [16]. On every packet arrival, AVQ updates the virtual capacity from arrived packet size and interarrival time, and drops the received packet if the virtual queue overflows. Unlike RED, it regulates utilization rather than queue length.

III. DESIGN AND CONFIGURATION

The purpose of AQM algorithms is to maintain high link utilization with low loss rate and queueing delay in a wide range of network environments [1], [15], [19], [20]. Stable operation and quick response to changes in load are also desirable [16], [21]. This section outlines the HRED algorithm and explains its design in terms of these attributes. We begin with a general discussion of how AQM algorithms provide control of queueing delay, then provide a simple analysis of local stability, i.e., stability under steady load, showing that HRED improves upon previous algorithms and provides intuitive controls for trading between buffer size and stability. We next describe the adjustment algorithm adapting to changes in load, and finish the section with detailed configurations focusing on global stability.

A. Queuing Delay Control

Two measures of queueing delay are important for AQM algorithms: The expected delay and the expected variance. To provide independent control of both measures, an algorithm requires at least two parameters. PIC, for example, has only a single parameter, q_{ref}, for the desired queue length. This parameter allows straightforward control of the average delay, but variance in delay depends on parameters of the control algorithm that offer no intuitive relationship to delay.

As delay is proportional to queue length, RED and its variants bound delay via the queue thresholds when the system is in steady state. In particular, RED tries to keep delay within the range $[\min_{th}/C, \max_{th}/C]$, where C is the link capacity. HRED uses the same dual-threshold model to control the queueing delay state. As with RED, delay in steady state can vary within the operating range.

B. Local Stability

AQM algorithms avoid incipient network congestion by providing feedback to flows in the form of marked or dropped packets. The system as a whole—including the AQM algorithm, cooperative end-hosts, and end-to-end delay—forms a closed feedback system. Through appropriate modeling of each element, algorithmic stability can be analyzed and proven. Recent research, for example, applies a Laplace transformation to linearized models of TCP and designs an AQM algorithm to provide local stability [8], [10], [14], [16]. In [9], the nonlinear instability problem of TCP-RED is analyzed to provide stability condition for a fixed point. In this section, we employ a discrete time model to analyze the control system like [9] and prove local stability by using the linearization technique. Note that the linearization of TCP's congestion avoidance element limits the proof to LOCAL stability.[1]

Suppose that a bottleneck link shared by N TCP connections with the same round-trip propagation delay of RTT, the expected window size of each connection is proportional to $1/\sqrt{\overline{p}}$, where $\overline{p}$ is the expected drop probability in steady state [9]–[11], [23]. These assumptions simplify the N-flow system to a single flow feedback system. In steady state (or equilibrium), the

[1] It was proved that if a small signal linear model is valid near an equilibrium and is stable, there is a small region containing the equilibrium where the nonlinear system is stable [22]. We state that the nonlinear system is locally stable if its linearized model is stable near an equilibrium.

queue holds the sum of all connections' windows minus the data in flight in the network. Thus, the expected average queue length $\bar{\ell}$ can be written in terms of $\bar{p}$ as

$$\bar{\ell} = N \frac{K}{\sqrt{\bar{p}}} - C \times RTT \tag{1}$$

where K is a constant of proportionality (i.e., $K/\sqrt{\bar{p}}$ is the expected window size).

For an AQM algorithm to be stable, a deviation from the expected average queue length at time t_k must not lead to a larger deviation at a later time $t_{k+1} = t_k + RTT$. More formally, let ℓ_k vary from $\bar{\ell}_k$ by some small amount $\delta\ell_k$, with

$$\ell_k = \bar{\ell}_k + \delta\ell_k. \tag{2}$$

The deviation in average queue length increases the drop probability, which in turn (after an RTT) reduces expected window sizes and the expected average queue length. For RED, the impact of the deviation in the average queue length is governed by the mapping function

$$p(\ell) = p_{\max} \frac{\ell - \min_{th}}{\max_{th} - \min_{th}}. \tag{3}$$

Substituting (2) into (3) at time t_k, we obtain

$$\begin{aligned} p(\ell_k) &= p_{\max} \frac{\bar{\ell}_k + \delta\ell_k - \min_{th}}{\max_{th} - \min_{th}} \\ &= \bar{p}_k + \frac{p_{\max}}{\max_{th} - \min_{th}} \delta\ell_k \\ &\equiv \bar{p}_k + \delta p_k. \end{aligned} \tag{4}$$

After an RTT, the window sizes reflect the change in p. Substituting (4) into (1) at time t_{k+1} and expanding the radical linearly around $\bar{p}_k$ (δp_k is small), we find

$$\bar{\ell}_{k+1} = N \frac{K}{\sqrt{\bar{p}_k + \delta p_k}} - C \times RTT \tag{5}$$

$$= \left(N \frac{K}{\sqrt{\bar{p}_k}} - C \times RTT \right) - N \frac{K}{\sqrt{\bar{p}_k}} \frac{\delta p_k}{2\bar{p}_k} \tag{6}$$

$$\begin{aligned} &= \bar{\ell}_k - N \frac{K}{\sqrt{\bar{p}_k}} \frac{\delta p_k}{2\bar{p}_k} \\ &= \bar{\ell}_k - N \frac{K}{\sqrt{\bar{p}_k}} \frac{1}{2\bar{p}_k} \frac{p_{\max}}{\max_{th} - \min_{th}} \delta\ell_k \\ &= \bar{\ell}_k - \frac{\bar{\ell}_k + C \times RTT}{2\bar{p}_k} \frac{p_{\max}}{\max_{th} - \min_{th}} \delta\ell_k \\ &\equiv \bar{\ell}_k + G(\bar{\ell}_k)\, \delta\ell_k. \end{aligned} \tag{7}$$

The last step makes use of (1) to rewrite $NK/\sqrt{\bar{p}_k}$. Without the linearization at fixed point $\bar{p}_k$ (or $\bar{\ell}_k$) in (6), we will have the same third degree polynomial of $\bar{\ell}^* = \bar{\ell}_{k+1} = \bar{\ell}_k$ as in [9].

The coefficient $G(\bar{\ell}_k)$ in (7) denotes the system gain, and indicates how a deviation in RED's average queue length at time t_k affects the expected average queue length at time t_{k+1}. If $|G(\bar{\ell}_k)| < 1$, the linearized system is stable because the perturbation near the fixed point decays exponentially. If $|G(\bar{\ell}_k)| \geq 1$, the system *may* be unstable. The ambiguity in the latter statement arises because use of $|G(\bar{\ell}_k)|$ represents a conservative approximation of system dynamics.

Rewriting our stability criterion as a bound on the expected average queue length helps to illuminate the reasons behind RED's instability. Starting from an expansion of $|G(\bar{\ell})| < 1$, we can solve the left hand side for $\bar{\ell}$ by using (3), and get

$$\bar{\ell} > C \times RTT + 2 \min_{th}.$$

Therefore, RED needs to keep queue size larger than $C \times RTT$ to be stable, which significantly increases queueing delay. Breaking the inequality does not necessarily mean that RED is unstable. In other words, $|G(\bar{\ell})| < 1$ is a sufficiency condition for stability but not a necessary one. The same analysis holds for ARED; manipulation of $p_{\max}$ does not affect any of the equations directly, although it can help reduce the likelihood of instability when an algorithm does not meet our stability criterion.

HRED meets the stability criterion by introducing another parameter. The value $p_{\min}$ in HRED specifies the drop probability when $\ell = \min_{th}$. The mapping function (3) becomes

$$p = (p_{\max} - p_{\min}) \frac{\ell - \min_{th}}{\max_{th} - \min_{th}} + p_{\min} \tag{8}$$

and $|G(\bar{\ell})|$ is given by

$$|G(\bar{\ell})| = \frac{\bar{\ell} + C \times RTT}{2\bar{p}} \frac{p_{\max} - p_{\min}}{\max_{th} - \min_{th}}. \tag{9}$$

Equation (9) allows the system gain to be controlled through changes to $p_{\max} - p_{\min}$ to ensure that the system is locally stable. HRED must maintain $|G(\bar{\ell})| < 1$ for local stability, independent of the current drop probability. To obtain this independence, we set $p_{\max}$ according to the equation

$$p_{\max} - p_{\min} = \frac{1}{\kappa} \frac{p_{\min}}{\max_{th}} (\max_{th} - \min_{th}). \tag{10}$$

The parameter κ adjusts the slope of the probabilities and enables tradeoff in design discussed later in this section. We substitute (10) first into (9) and then (8) to simplify the stability criterion for HRED as follows

$$\bar{\ell} > C \times RTT + 2 \min_{th} - 2\kappa \max_{th}. \tag{11}$$

The additional threshold term makes it substantially easier for HRED to achieve stability than for RED. For instance, by picking $\kappa \max_{th} - \min_{th} > C \times RTT/2$, we can make HRED stable for any feasible value of $\bar{\ell}$. Besides, the stability holds regardless of number of connections unlike other stability criteria in [9] and [14]. If we relax this criterion[2] for stability within the operating range, $\bar{\ell} \geq \min_{th}$, we need only $\kappa \max_{th} - \frac{1}{2} \min_{th} > C \times RTT/2$ for guaranteed local stability. Substituting reasonable values of the thresholds in terms of the buffer size into the last inequality ($\max_{th} = Q_{\max}/2$ and $\min_{th} = Q_{\max}/4$), we obtain $Q_{\max} > \frac{4}{4\kappa - 1} C \times RTT$. The probability slope factor κ thus allows us to reduce the amount of memory required for local stability.

[2]Using the approach of [9], HRED should obey $\bar{\ell} + C \times RTT < 2\kappa \max_{th}$ to avoid *border collision* and to achieve stability. This gives a tighter bound than (11) because $\bar{\ell} \geq \min_{th}$.

if $(q > \max_{th})$
$p_{\min} = p_{\min} + K_\alpha p(q - \max_{th})$
if $(q < \min_{th})$
$p_{\min} = p_{\min} - K_\beta p(\min_{th} - q)$
$p_{\max} = p_{\min} + \frac{1}{\kappa} \frac{p_{\min}}{\max_{th}} (\max_{th} - \min_{th})$

Fig. 1. Adjustment algorithm of HRED.

PIC achieves local stability by controlling the gain and phase margins. In our analysis, we consider only the gain, permitting a relatively simple if somewhat conservative stability criterion. The analysis developed for PIC in [8] can be used to validate the use of our criterion.

Rewriting the DC gain of the plant function $\frac{(C \times RTT_0)^3}{(2N)^2}$, where RTT_0 represents the propagation delay and the queueing delay, and letting[3] $K = \sqrt{2}$ in (1), we obtain $\frac{\bar{\ell} + C \times RTT}{2\,\bar{p}}$, which is the first term of $|G(\bar{\ell})|$. Since the second term coming from the mapping function of RED is the same as the DC gain of transfer-function L_{red} in [8], the DC gain of the system $\frac{(C \times RTT_0)^3}{(2N)^2} L_{\text{red}}$ is identical to $|G(\bar{\ell})|$. In the frequency range of interest, the system approximates to a single pole model and its gain is less than or equal to its DC gain.

C. Adjustment to Dynamic Traffic Loads

Stability in steady state for a given traffic load is an important aspect of AQM algorithms, but a good algorithm must also maintain stability in dynamic loads. Most AQM algorithms use the same control mechanism for both steady and dynamic traffic loads, HRED separates them to get fast response time while retaining other good properties in steady load. The local stability conditions derived in the previous section are inadequate for dynamic traffic loads, as the queue may make buffer overflows or be empty as traffic changes. In either case, RED and HRED will lose stability and suffer from high loss rates and low link utilization [11], [12], [24].

A simple way to adjust to dynamic loads, motivated by ARED, is to adjust $p_{\min}$ and $p_{\max}$. However, ARED loses even local stability under heavy loads because it adjusts only $p_{\max}$ [17]. HRED adjusts both probabilities in tandem satisfying (10). Pseudocode for this adjustment appears in Fig. 1. Before specifying the algorithm in detail, we present other features of HRED distinguishing it from RED.

The introduction of the adjustment algorithm separated from the mapping function not only decouples queue length and drop probability, but also allows HRED to discriminate between temporal variations and changes in traffic load. For temporal variation, HRED gets p using (8) and adjusts the range of mapping to traffic loads by changing $p_{\min}$ and $p_{\max}$. RED averages the queue length to avoid short-term variations from bursty traffic or temporal congestion [5]. In HRED, however, these short-term changes can also be handled through adjustment of the probabilities, thus HRED does not need to average queue length. HRED effectively sets $w_q = 1$, making ℓ equal to the actual queue length q. We replace ℓ and $\bar{\ell}$ in previous equations of HRED with q or $\bar{q}$. The removal of averaging is helpful to avoid TCP synchronization [20] and reduces complexities in analysis.

[3]The authors of [8] take ${W_0}^2\, p_0 = 2$ for a single TCP connection where W_0 and p_0 are expected window size of TCP and drop probability of RED at the operating point.

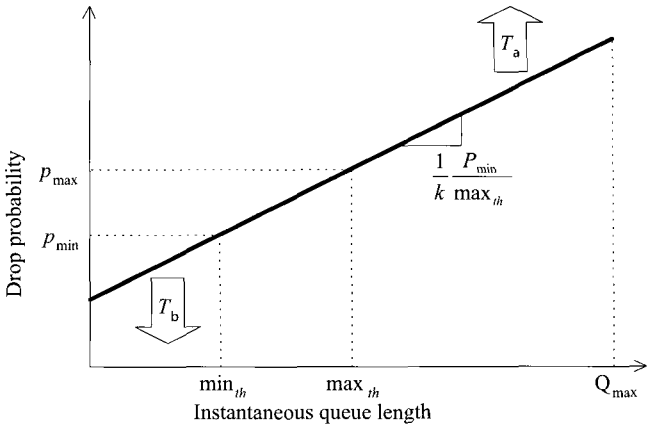


Fig. 2. Drop probability mapping function of HRED.

Another important change from RED is the extension of linear range in the mapping function. From an observation that the discontinuity of p at $\max_{th}$ occasionally causes excessive packet drops and/or leads to instability, we designed HRED to apply (8) for all values of q. Fig. 2 illustrates the mapping function of HRED.

With removal of EWMA and extension of the linear function, it is easier to analyze the adjustment algorithm. Since the adjustment algorithm affects response time and can influence global stability, HRED may exhibit unstable behavior if it is overly aggressive in adjustments. Conservative adjustments, on the other hand, can lead to lack of responsiveness to change. Stability and response time thus offer a tradeoff.

Assume that the system is stable with N connections and expected queue length $\bar{q}_k$. Let the number of connections increase abruptly by ϵN. We consider the necessary change to drop probability in order to maintain $\bar{q}_{k+1}$ at its current value $\bar{q}_k$. From (1), we find the desired drop probability $\bar{p}'_{k+1}$ to be given by

$$\sqrt{\bar{p}'_{k+1}} = (1+\epsilon)\, N\, \frac{K}{\bar{q}_k + C \times RTT} \qquad (12)$$
$$\bar{p}'_{k+1} = (1+\epsilon)^2\, \bar{p}_k.$$

Then, the drop probability should increase by $(1+\epsilon)^2 p - p = \epsilon(2+\epsilon)p$.

We estimate the increment on a packet arrival from the algorithm in Fig. 1. The estimation will show that the algorithm is designed to control response time independent of current status and the amount of traffic changes.

Given an increase of N to $(1+\epsilon)N$, the total incoming traffic rate increases from C to $(1+\epsilon)C$ and, in turn, the queue length for an interval T_α increases by $\epsilon C T_\alpha$, which corresponds to $(q - \max_{th})$ in Fig. 1.[4] As drop probability is updated on every packet arrival, the increment of drop probability for T_α must

[4]We implicitly make two assumptions. First, we assume a SUDDEN increase in the traffic rate. If a gradual increase is assumed, the added queue length should be halved. Second, we assume that $\bar{p}$ does not change, which does not hold if $T_\alpha > RTT$. We restate it in the next section.

satisfy

$$K_\alpha p\left(\epsilon CT_\alpha\right)\frac{(1+\epsilon)CT_\alpha}{S}=\epsilon\left(2+\epsilon\right)p \tag{13}$$

where S is the average packet size, and the fractional factor in the equation gives the average number of arrivals for T_α, and the right hand side $\epsilon\left(2+\epsilon\right)p$ is the targeted increment of drop probability. From (13), we then obtain

$$K_\alpha \Leftarrow \frac{S}{\left(CT_\alpha\right)^2}\frac{2+\epsilon}{1+\epsilon} \tag{14}$$

which can be approximated by $\frac{2S}{(CT_\alpha)^2}$ for $\epsilon \ll 1$.

Similarly,

$$K_\beta \Leftarrow \frac{S}{\left(CT_\beta\right)^2}\frac{2-\epsilon}{1-\epsilon}\approx\frac{2S}{\left(CT_\beta\right)^2}. \tag{15}$$

From (14) and (15), the response time of HRED can be configured through T_α and T_β, and independent of the amount of traffic change.

D. Configuration

In this section, we discuss detailed configuration of parameters in HRED focusing on stability of the adjustment algorithm. Although response times are presented by T_α and T_β, they are not equal if T_α and T_β are larger than one RTT. An RTT after traffic changes, the incoming rate is adjusted and (13) no longer holds. The adjusted rate is less than $(1+\epsilon)C$ and closer to the link capacity C. It takes longer than T_α and T_β for HRED to get the target drop probability. We loosely represent the response time, nevertheless, in terms of T_α and T_β even if they are larger than one RTT.

Let RTT^+ be the largest average round-trip time in the system. Conservative configuration of HRED bounds the minimum of T_α and T_β to RTT^+ in order to remove possible overshoots in the drop probability.[5] Otherwise, HRED may be unstable and make large variations of queue length.

If there is no packet drop from buffer overflow, the conservative configuration is also helpful to avoid oscillation. The oscillation in this case comes from wrong rate control. Overly increased or decreased drop probability makes sources react in their rates exceedingly, and causes repeated oscillations of queue length with large variation.

However, when packets are dropped from buffer overflow, the conservative configuration is not sufficient to achieve stability. The instability may occur because the drop probability calculated from the queue length is not consistent with the actual drop probability, which mainly depends on packet drops from buffer overflow. This can happen when a large number of connections are suddenly established and the adjustment algorithm fails to follow. When the drop probability p is severely underestimated, the overly increased traffic can bring packet drops from buffer overflow before HRED adjusts p correctly. Packet drops result in a sudden traffic decrease due to AIMD of TCP congestion algorithm. The oscillation will be repeated unless the gap between the estimated drop probability and the actual drop probability shrinks. On the other hand, a sudden removal of established connections does not trigger the oscillation because overestimated drop probability does not result in buffer overflow.

Asymmetric configuration of HRED provides a simple way to remove the instability. Since the increment of the drop probability depends on $Q_{\max}-\max_{th}$ and T_α, while the decrement depends on $\min_{th}$ and T_β, we can make the estimated drop probability catch up with the actual drop probability by keeping $(Q_{\max}-\max_{th})>\min_{th}$. After a few rises and falls of queue length, the gap between them disappears. One alternative to achieve stability is to give additional increment of the drop probability to every buffer overflow like Blue. Another is to monitor the actual drop probability and set the estimated drop probability to it when there is a significant difference between them.

We use the asymmetric configuration of $(Q_{\max}-\max_{th})=2\ \min_{th}$ and $T_\alpha=2\ T_\beta$ throughout our simulations and tests. The former removes instability, and the latter speeds up responsiveness in case of traffic decrease, helps HRED achieve full link utilization, and compensates for relative delay in response resulting from the former. Although the heuristics of conservative and asymmetric configurations do not guarantee global stability of HRED, we use them as an alternative to improve stability.

Previous discussions are implicitly restricted to the case of traffic of long-lived connections, but in reality there are many short-lived connections such as HTTP in the Internet. In HRED, it is possible to estimate the amount of short-lived traffic that can be handled.

Since HRED retains the advantages of AQM provided that q remains in $[0,\ Q_{\max}]$, we can estimate the tolerance of system based on the extended linear range and the removal of averaging. Suppose that the system is stable with N connections and the expected queue length $\overline{q}_k$ at time t_k. Let σN be an amount of traffic added or removed abruptly. We use (1) to obtain the maximum of σ that does not make buffer overflow. In the new equilibrium at time t_{k+1}, we have $\overline{q}_{k+1}<\frac{(1+\sigma)NK}{\sqrt{\overline{p}_k}}-C\times RTT$ because $\overline{p}_{k+1}$ is less than $\overline{p}_k$. Since $\overline{q}_{k+1}$ should be less than $Q_{\max}$ to avoid overflow, the right hand side of the inequality is bounded by $Q_{\max}$. Replacing $\frac{NK}{\sqrt{\overline{p}_k}}$ with $\overline{q}_k+C\times RTT$ from (1), we can express a bound on σ as $\frac{Q_{\max}-\overline{q}_k}{\overline{q}_k+C\times RTT}$. Similarly we get a different bound $\frac{\overline{q}_k}{\overline{q}_k+C\times RTT}$ that prevents the queue from being empty. Combining these two bounds, we have

$$\sigma\le\ \min\left(\frac{Q_{\max}-\overline{q}}{\overline{q}+C\times RTT},\frac{\overline{q}}{\overline{q}+C\times RTT}\right).$$

The left restriction prevents buffer overflow and the right underflow. Minimizing the equation over $\overline{q}$ within the operating range of $[\min_{th},\ \max_{th}]$, we obtain

$$\sigma\le\ \min\left(\frac{Q_{\max}-\max_{th}}{\max_{th}+C\times RTT},\frac{\min_{th}}{\min_{th}+C\times RTT}\right). \tag{16}$$

This shows the algorithm's limit in handling rapid changes. HRED maintains stable control and retains the advantages of AQM as far as changes in load satisfy (16).

[5] At the same time, they also should be as small as possible for HRED to be effective following the controlled system [29].

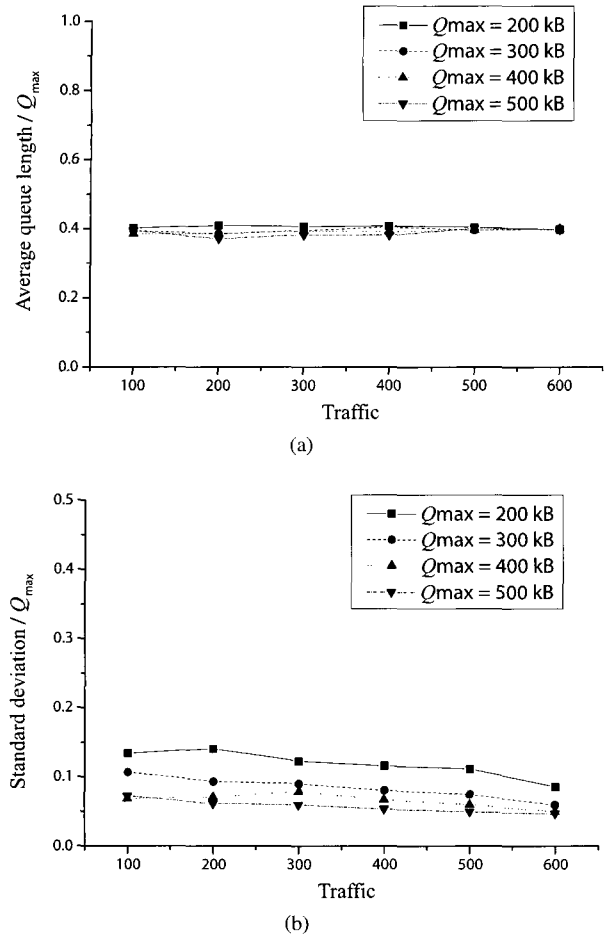


Fig. 3. Steady state properties of HRED with various buffer sizes: (a) Average queue length, (b) standard deviation of queue length.

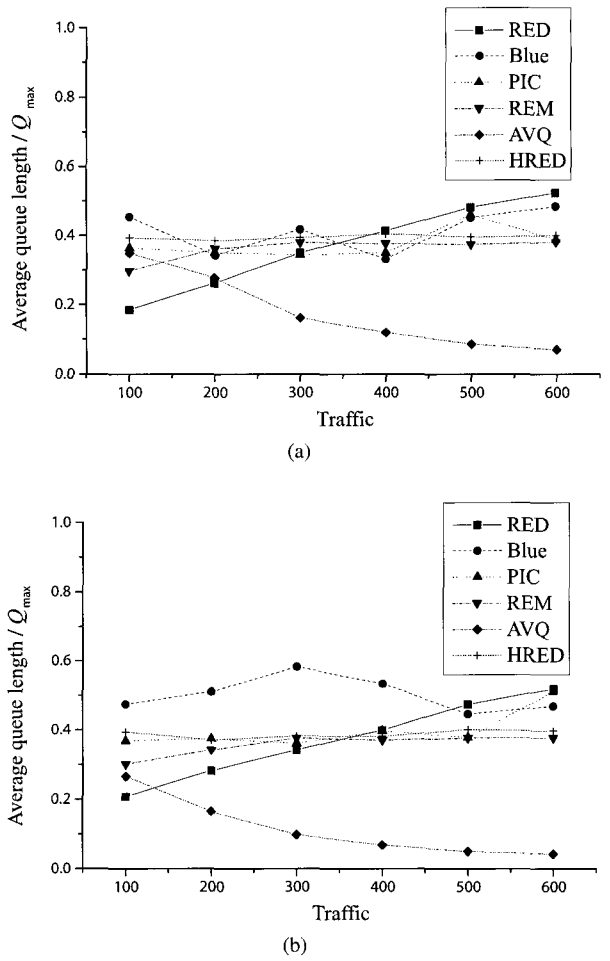


Fig. 4. Average queue length: (a) $Q_{\max} = 300$ kB, (b) $Q_{\max} = 500$ kB.

The tradeoff in queue parameters is now clear. If we bring $\max_{th}$ closer to $Q_{\max}$ to improve local stability by using limited memory from (11), we do so at the expense of some amount of short-lived traffic and global stability. Increasing $\max_{th}$ from $\frac{1}{2}\,Q_{\max}$ to $\frac{3}{4}\,Q_{\max}$, for example, reduces the buffer size requirement for local stability to $\frac{4}{6\kappa-1}\,C \times RTT$, but it raises the risk of instability and cuts the maximum value of σ by more than two thirds.

E. Summary

Although HRED introduces an additional parameter for control, its behavior is an intuitive and predictable function of its parameters. Equation (10) specifies the general form of the relationship between the probability parameters, with which an administrator can trade between buffer size, local stability, and variation of queue length.

For stability and quick response, HRED removes EWMA of RED and extends linear range in the mapping function. Local stability is analytically proven using a linearized model of TCP. Response time is designed to be independent of the amount of traffic changes and represented by T_α and T_β, which are configured conservatively and asymmetrically for global stability. Guaranteeing global stability of the system is left for future work.

IV. SIMULATION COMPARISON

This section uses the ns-2 simulator [25] to compare HRED with RED, Blue, PIC, REM, and AVQ. We first explore queueing delay by measuring queue length and standard deviation in steady load. The results corroborate the analytic claims of the previous section, demonstrating that RED fails to impose adequate control on queueing delay. We also investigate robust properties of AQM algorithms and avoidance of TCP lock-out behavior. We next compare the dynamic behaviors of AQM algorithms in terms of link utilization and response time to loading changes. We observe that HRED provides faster and more predictable response times without hurting utilization.

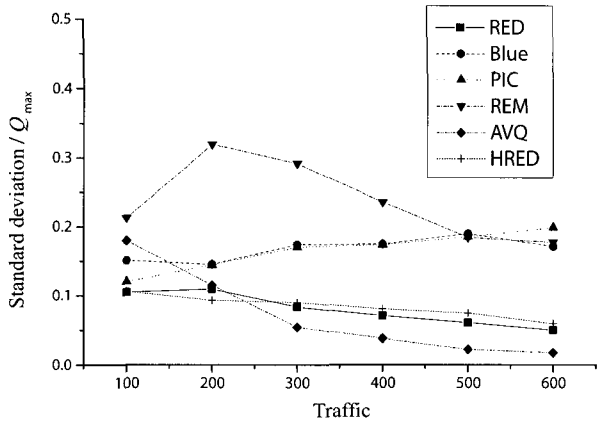
Except where otherwise specified, the simulations are based on a dumbbell topology in which all connections traverse a single bottleneck link with capacity 45 Mbps. The bottleneck link queue operates in byte mode with buffer size of 300 kB. We assign a random round-trip time of 40 to 200 ms to each connection (uniform distribution), which includes everything except queueing delay at the bottleneck. One set of random RTTs is used in all experiments. Average packet length is 500 bytes. AQMs are configured as recommended[7] in [5], [13], [14], [16],

[6] RTT^+ should be equal to or larger than RTT_0 for all TCP connections, where RTT_0 includes propagation delay and queueing delay [14].

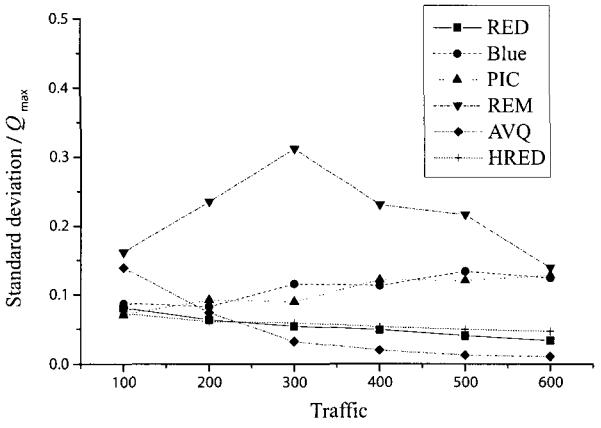
[7] Blue is configured to have finer granularity of the drop probability than the

Table 1. Configurations of active queue management algorithms for simulation.

	Queue parameters	Criteria for adjustment
RED	$\min_{th} = \frac{1}{2}\max_{th} = \frac{1}{4}Q_{\max}$	$w_q = 0.002$, $p_{\max} = 0.1$
Blue		$d1 = 0.0002$, $d2 = 0.00002$, $freeze\_time = 10$ ms
PIC	$q_{\text{ref}} = \frac{3}{8}Q_{\max}$	$RTT^+ = 200$ ms,[6] $N^- = 100$, 160 Hz sampling
REM	$b^* = \frac{3}{8}Q_{\max}$	$\alpha = 0.1$, $\gamma = 0.001$, $\phi = 1.001$, 1 kHz sampling
AVQ		$\alpha = 0.15$, $\gamma = 1.0$
HRED	$\min_{th} = \frac{1}{2}\max_{th} = \frac{1}{4}Q_{\max}$	$\kappa = 2$, $T_\alpha = 400$ ms, $T_\beta = 200$ ms



(a)



(b)

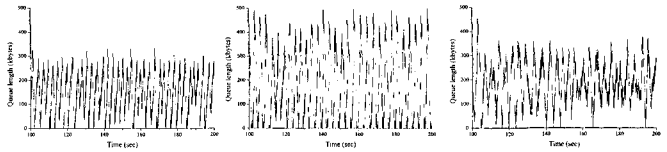
Fig. 5. Standard deviation of queue length: (a) $Q_{\max}$ = 300 kB, (b) $Q_{\max}$ = 500 kB.

[26], [27], with the `gentle_` option enabled for RED. The details of configuration are given in Table 1.

A. Steady State Behavior

We monitored average queue length and standard deviation over a period of 50 seconds after the initial 150 seconds. The load consists of a combination of long-lived greedy FTP connections and WEB sessions. Each WEB session consists of 10 HTTP transfers and its length follows a Pareto distribution with the mean of 10 packets and the shaping parameter of 1.2. The simulation results for different traffic loadings, in which the loss rates range from 0.005 to more than 0.1, appear in Figs. 3–5.

recommended configuration, with which Blue has oscillatory behaviors due to coarse granularity.



(a) (b) (c)

Fig. 6. Instantaneous queue length of REM with $Q_{\max}$ = 500 kB in different traffic loads: (a) 100 *traffic*, (b) 300 *traffic*, (c) 600 *traffic*.

The horizontal axis with the unit *traffic* reports both the number of FTP connections and the arrival rate of WEB sessions with Poisson distribution. For example, 300 *traffic* means that 300 long-lived FTP connections are established and WEB sessions arrives at $300\frac{1}{12}$ sessions per second.

Fig. 3 illustrates the properties of HRED in steady load. With various buffer sizes, it succeeds in keeping average queue length constantly to the value of $\frac{1}{2}(\min_{th} + \max_{th})$ corresponding to 0.375 after normalization. HRED also has small deviation gradually decreasing as traffic and buffer size increase. Its queueing properties are independent of traffic loads and queue size in comparison with other AQM algorithms. The independence leads to less jitter as the load fluctuates in various network environments.

We also run simulations for comparision with other AQM algorithms. Figs. 4 and 5 respectively present average queue length and standard deviation when the buffer size is 300 kB and 500 kB. We can confirm that RED does not decouple congestion measure from performance measure [15], [19]. Average queue length of RED increases linearly as traffic increases. However, in terms of deviation, RED presents better performance than Blue, PIC, and REM.

Since Blue does not change a drop probability unless the queue is overflow or underflow, its average queue length is arbitrary and independent of the amount of traffic. In standard deviation, Blue has large variation and can not control queue length. Therefore, neither average queue length nor standard deviation is under control in Blue.

Fig. 4 also shows that PIC and REM successfully maintain queue length at a target value. Both always try to find a proper drop probability for the target queue length using the same controller. However, in Fig. 5, they have as large deviation as or even more than Blue and show their limit in controlling queue length. The non-monotonic behavior of REM in deviation comes from the nonlinear characteristics of the system. As the drop probability $\overline{p}_k$ gets larger, a perturbation δp_k takes less effect on queue length ℓ_{k+1} from (6). Fig. 6 illustrates instanta-

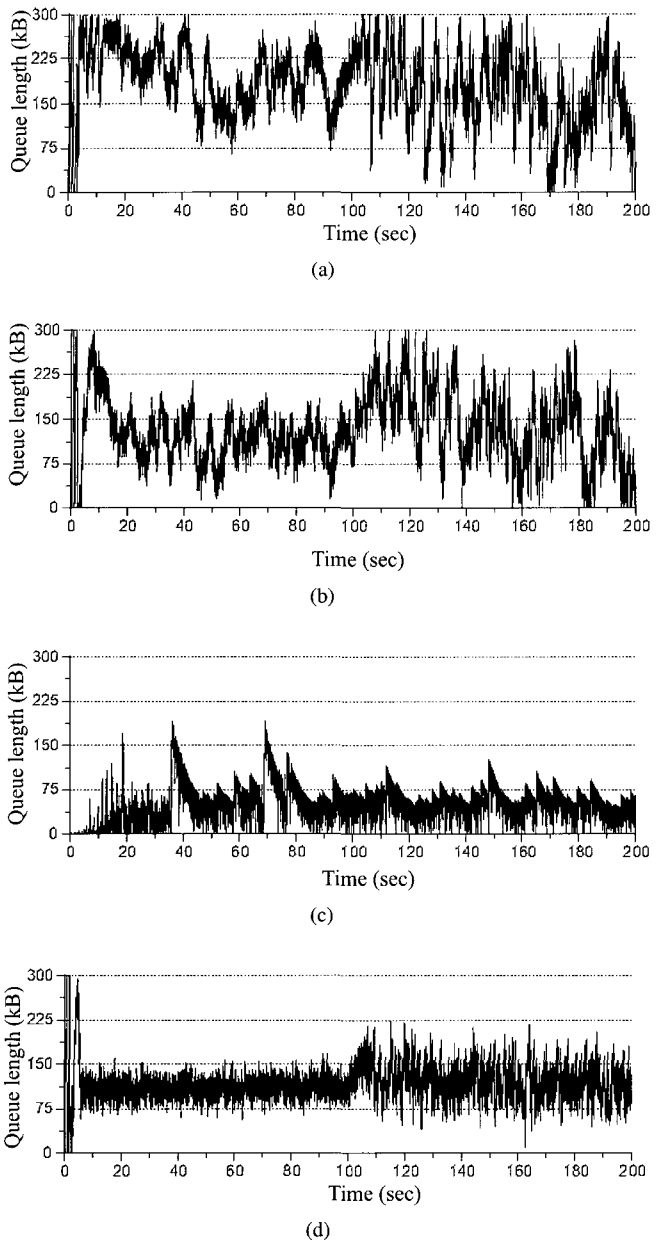


Fig. 7. Instantaneous queue length of AQM algorithms. WEB traffic is added to FTP traffic at 100 sec: (a) Blue, (b) PIC, (c) AVQ, (d) HRED.

neous queue lengths of REM with 500 kB buffer size in different traffic loads. In light load of 100 *traffic*, queue length hardly increases over 300 kB. It largely decreases at an increase of drop probability due to a small $\overline{p}_k$. As $\overline{p}_k$ increases with traffic loads, the reduced effect of δp_k on queue length allows it to swing from top to bottom of the buffer as shown in Fig. 6(b). It results in the largest deviation in Fig. 5(b). With more traffic loads, however, REM gets control of queue length and reduces oscillatory behaviors.

The difference between PIC and RED comes from different mapping of congestion measure to drop probability. Another reason is that the recommended configuration of REM in [27] is not suitable to this network environment. This implies that REM still has configuration problems. Throughout extended simulations, whose results are not shown due to lack of space, we observe that REM can have similar standard deviations to PIC with smaller γ. We exclude REM afterward from simulation results because its large deviation is intolerant, and with smaller γ, its properties are close to that of PIC.

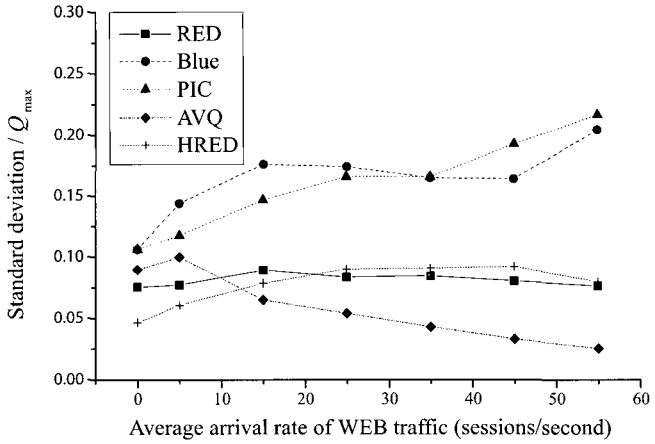


Fig. 8. Robustness of AQMs algorithms to WEB traffics.

The behavior of AVQ is interesting. It always tries to keep queue length as small as possible without hurting utilization. Therefore, queue length of AVQ is the smallest in Fig. 4, but can not be controlled to a desired queue length. Another weakness of AVQ resides in scalability to light loads. Since it fails to keep small deviation at light loads, it increases average queue length in order to not hurt utilization. To make things worse, this can not be cured by increasing buffer size as shown in Fig. 5, because AVQ is a totally rate-based control algorithm.

In the next experiment, we verify robustness of AQM algorithms to short-lived TCP connections such as WEB traffic. It is important because most AQM algorithms including HRED proved their local stability through the linearization of TCP congestion control algorithm. We start simulation with 300 long-lived FTP connections and add, after 100 sec, WEB traffic that has average arrival rate of 25 sessions per second.

Fig. 7 illustrates instantaneous queue lengths of Blue, PIC, AVQ, and HRED. We omit RED that has similar results with HRED. The variations before 100 sec comes from heterogeneous network environments of different RTTs and packet sizes between connections. Following buffer overflow at 100 sec, Blue and PIC have large swings of queue length. On the contrary, AVQ has rather less variation after addition of WEB traffic. It makes AVQ more attractive with heavy load of WEB traffic. In case of HRED (and RED), it still holds queue length mostly in the operating range of $[\min_{th}, \max_{th}]$.

To make the difference clear, we measure standard deviation of queue length changing the amount of WEB traffic. The simulation results are shown in Fig. 8. With no WEB traffic, all AQM algorithms operate stably with normalized standard deviation of less than 0.11. AVQ shows the most outstanding performance. Even though traffic variation increases with additional WEB traffic, it has less deviation. Blue and PIC are unable to manage increased traffic variation and have large deviation as an outcome. It can be critical especially in dynamic traffic loads of Internet. RED and HRED show good performance. They manage queue length by preventing it from going beyond $\min_{th}$ and $\max_{th}$, and thereby restrict deviation while controlling average

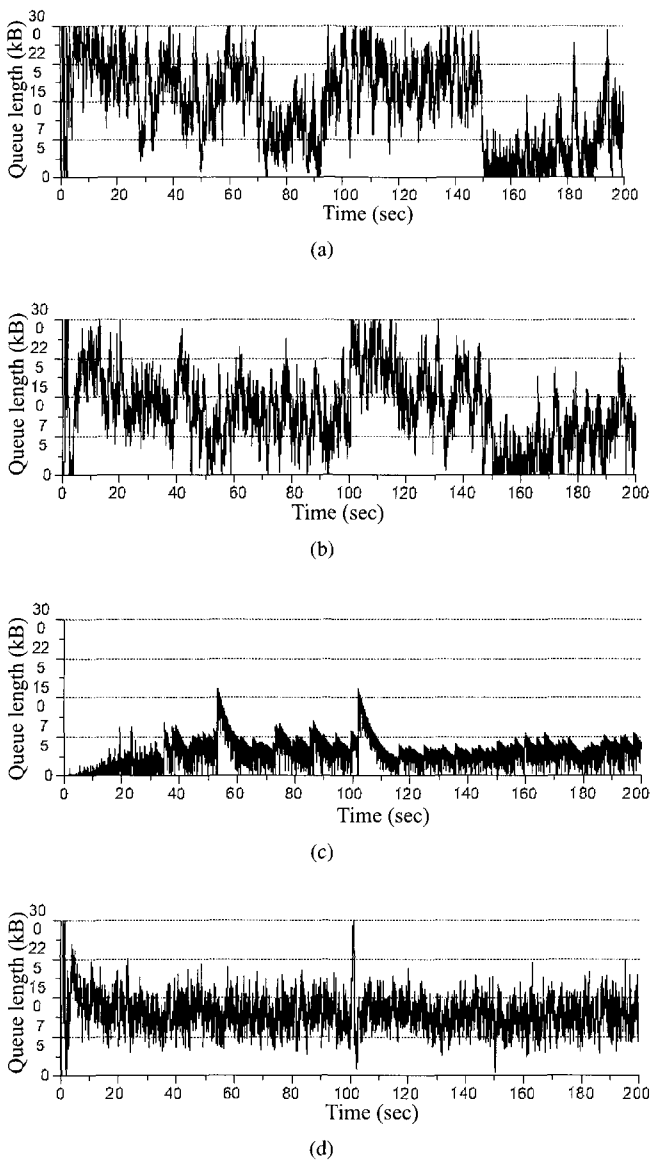


Fig. 9. Instantaneous queue length of AQM algorithms. Traffic increases at 100 sec and returns at 150 sec: (a) Blue, (b) PIC, (c) AVQ, (d) HRED.

Table 2. Goodput percent of TCP connections of 20 ms RTT.

	RED	Blue	PIC	AVQ	HRED
%	76.9	73.8	76.2	89.7	76.0

queue length.

As the last experiment for steady state behavior, we compare AQM algorithms in relative goodput between connections of two different RTTs. We establish two groups of connections. One has 20 ms RTT and the other 200 ms RTT. Each group consists of 150 FTP connections and all connections share a single bottleneck. All AQM algorithms achieve the total goodput of more than 99% of bandwidth.

We illustrate the percent goodput of short RTT connections out of total goodput in Table 2. AVQ weighs short RTT connections than the other AQM algorithms. Since AVQ does not main-

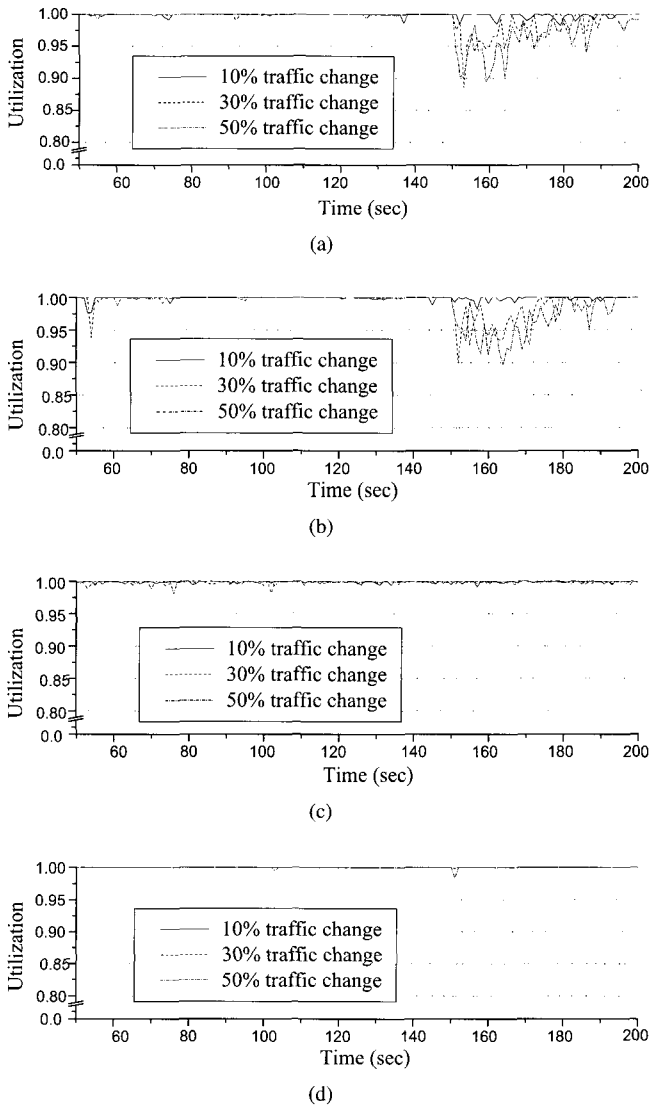


Fig. 10. Link utilization of AQM algorithms. Traffic increases at 100 sec and returns at 150 sec: (a) Blue, (b) PIC, (c) AVQ, (d) HRED.

tain drop probability internally and drops a packet when the virtual queue overflows instead of random dropping, it more likely drops a packet from bursty traffic. Comparing with tail-drop algorithm, in which the goodput of short RTT connections occupies 91.6% of the total, AVQ without random dropping does not relieve unfairness that comes from different RTTs and leads to TCP lock-out behavior, in which TCP connections of short RTT occupy most queue space preventing others from getting in.

The experiments demonstrate that only HRED manages to provide stable, predictable queueing delay. RED, Blue, PIC, REM, and AVQ fail in some network environments through oscillatory behavior and/or lack of queue-length tractability.

B. Dynamic Behavior

We next examine behaviors of AQM algorithms in dynamic traffic loads. In order to have intuition about the effects of dynamic loads on performance, it is worth while to observe in-

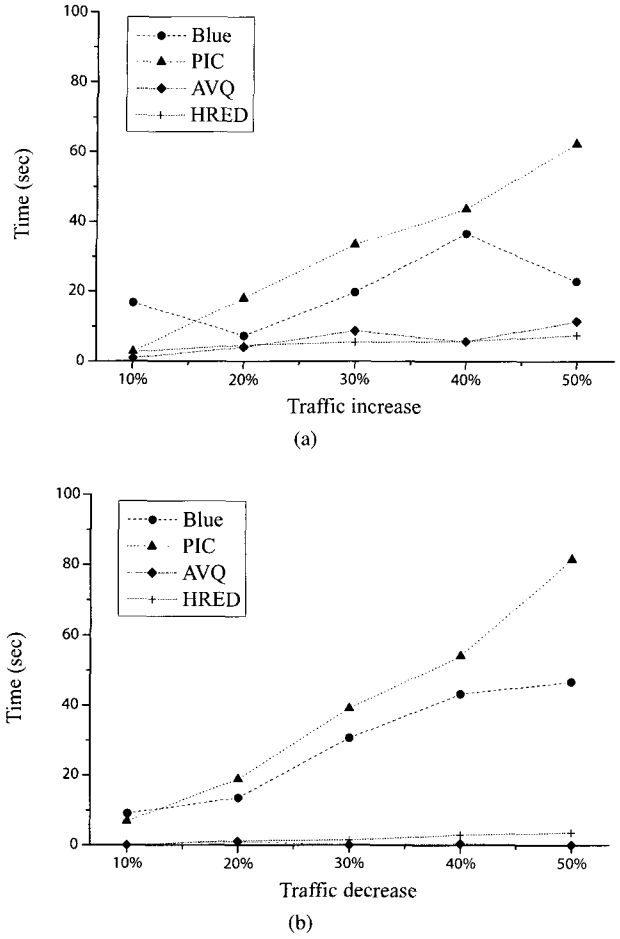


Fig. 11. Response time of AQM algorithms: (a) After traffic increase, (b) after traffic decreas.

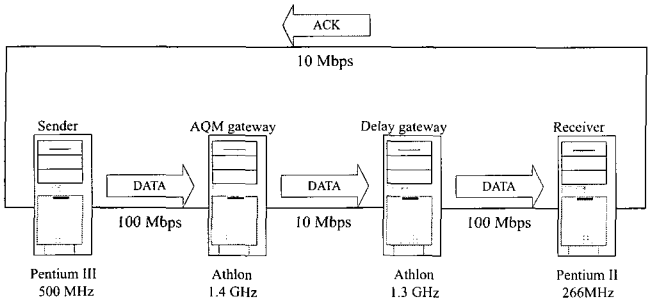


Fig. 12. Topology of implementation test.

stantaneous queue length of each algorithm in dynamic loads. We start our simulation with 300 long-lived FTP connections and WEB traffic having the arrival rate of 25 sessions per second. The total traffic increases by 30% at 100 sec and returns at 150 sec. Fig. 9 illustrates instantaneous queue length of each AQM algorithm.

With traffic increase, Blue builds up queue length and adjusts the drop probability when buffer overflows. After the adjustment, it still keeps large queue length because it can not handle queue length. For the same reason, it has small queue length after traffic decrease at 150 sec. PIC, though it can handle queue length at traffic changes, adjusts drop probability so slowly that it takes queue length away from q_{ref} for a while.

During the adjusting period, there are significant performance degradations in Blue and PIC. They lose advantages of AQM such as low delay and avoidance of lock-out behavior while buffer overflows, and fail to achieve full link utilization while the queue is empty. Figs. 10(a) and 10(b) ensure that the traffic decrease hurts link utilization in Blue and PIC.[8] The larger the amount of change is, the longer it takes for them to achieve full utilization.

On the contrary, AVQ and HRED have quick responses and advantages of AQM. In Fig. 9(c), we can observe that AVQ has a sharp rise of queue length at 100 sec but it decreases soon without buffer overflow. At 150 sec, it is even hard to figure out the effect of traffic decrease but there is a slight increase of variation. In case of HRED in Fig. 9(d), a sharp rise and fall of queue length at 100 sec and 150 sec disappear quickly without performance degradation. Figs. 10(c) and 10(d) also confirm that AVQ and HRED fully utilize the bottleneck link.

Since fast response is essential to avoid performance degradation, we further concentrate on response time. We estimate response time by measuring the time when standard deviation of queue length after traffic change is less than 1.5 times of that in steady state. The results are shown in Fig. 11. We measure and average the response times of five simulation runs. As shown in Fig. 11, AVQ and HRED respond quickly regardless of the magnitude of traffic changes. However, PIC not only responds slowly but also shows proportional increase of response time to traffic changes. The latter states that it responds slowly with large traffic changes even though it is reconfigured for quick response at the cost of stability and robustness. Hence, we conclude that AVQ and HRED are superior to PIC in terms of response time. The response time of Blue varies irregularly because it has large deviation even in steady load.

V. IMPLEMENTATION

In this section, we explore operability of AQM implementations in testbed. The testbed consists of 4 machines as shown in Fig. 12. AQM algorithms operate in AQM Gateway before the bottleneck link of 10 Mbps. The delay gateway plays a role of increasing the network capacity by inserting 100 ms delay into connections. Since all links are Ethernet, a separate feedback link is added for ACKs to return to the sender without colliding with data packets at the link layer. AQM algorithms are implemented on Linux kernel 2.4.9. Fixed-point approximation is applied to remove floating-point computations and to reduce processing loads. The configurations are the same as for the simulation except $RTT^{+} = 100$ ms, $N^{-} = 30$, 100 Hz sampling in PIC; 100 Hz sampling in REM; $\gamma = 0.975$ in AVQ; and $T_{\alpha} = 200$ ms, $T_{\beta} = 100$ ms in HRED.

We generate traffic as a mixture of long-lived greedy FTP connections and short-lived FTP connections. The short-lived connections arrive with Poisson distribution and each transmits 10 packets. All packets are 500 bytes in length. The x-axis in Fig. 13, which illustrates the test results, presents number of

[8] We omit the results before initial 50 seconds. Since we can not set initial parameter for AQM algorithms on equal terms, we can not fairly compare their initial behavior. For example, though we can set initial drop probability for PIC, we can not for AVQ.

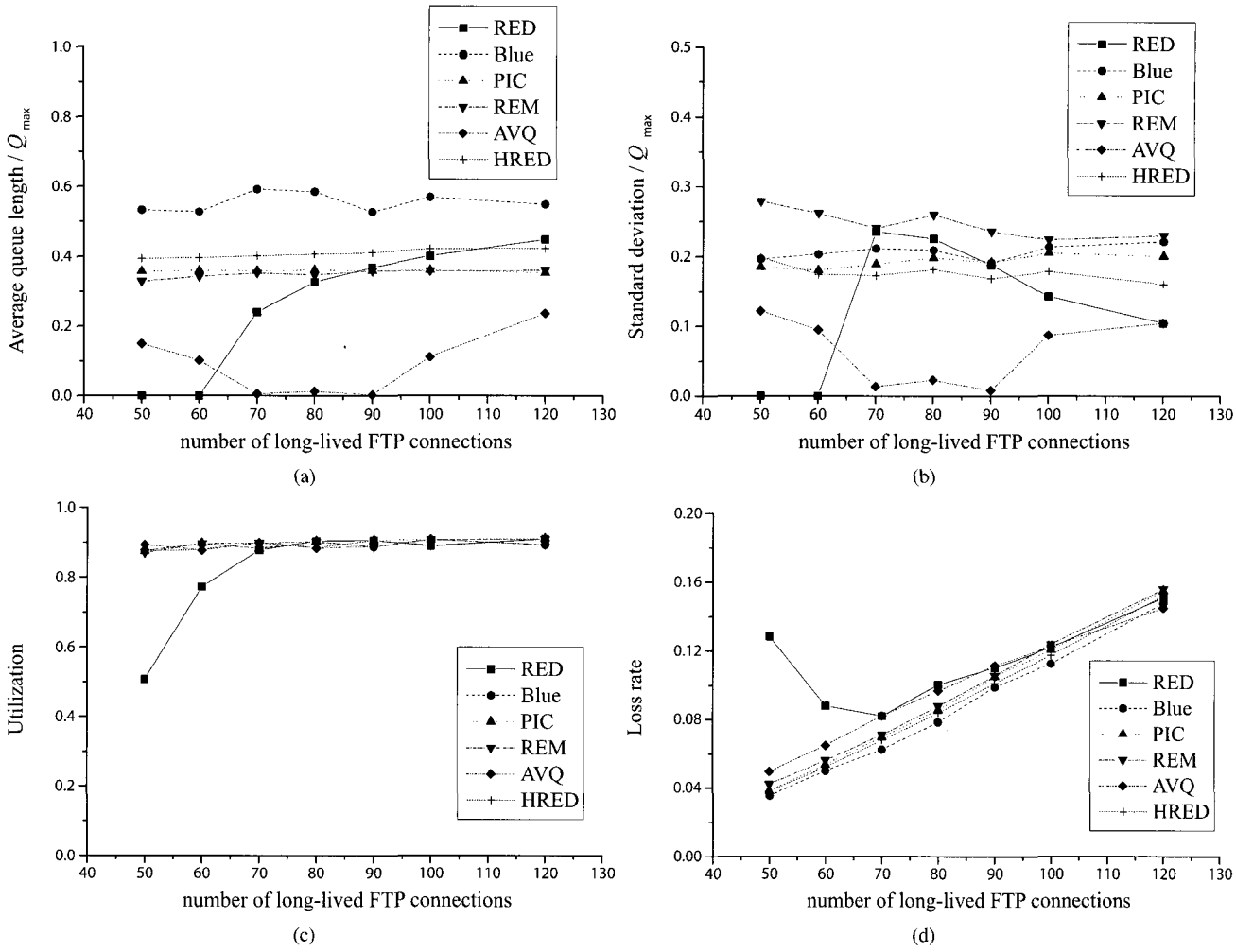


Fig. 13. Characteristics of AQM implementations with buffer size of 50 kB: (a) Average queue length, (b) standard deviation of queue length, (c) link utilization, (d) loss rate.

long-lived connections along with average arrival rate of short-lived connections: 50 in x-axis implies that 50 long-lived connections are being serviced and $50\frac{1}{2}$ short-lived connections are arriving per second.

Generally, (a) average queue length and (b) standard deviation are similar to those of Fig. 4, and we can lead the same conclusion as before. The differences from simulations come from fixed-point approximations of AQM algorithms. One is that RED does not build up queue length at all with 50 and 60 connections, and fails to achieve full link utilization with high loss rate. The other is that AVQ increases queue length again with 100 and 120 connections.

Figs. 13(c) and 13(d) show link utilization and loss rate. The link utilization is measured by dividing the number of forwarded bytes for a given period by the link capacity. It can not reach up to 1.0 due to the packet overhead of MAC and TCP/IP header. As shown in Fig. 13(c), all AQM implementations achieve full utilization in steady state except RED. The loss rate is estimated by dividing the number of packet drops by the number of packet arrivals. From Fig. 13(d), it is evident that RED and AVQ have higher loss rate than the others. While the high loss rate of RED results from fixed-point approximation, that of AVQ comes from small queue length and its bias against bursty traffic.

VI. DISCUSSION

AQM marks packets instead of dropping when it is used with ECN. There are studies investigating the effects of AQM that is ECN capable [13], [14], [28], which can be also applied to HRED. It is observed that AQM with ECN operates similarly but rarely has packet drops. AQM and ECN can be compared with explicit control protocol (XCP) proposed by Katabi *et al.* in [29]. XCP generalizes ECN and decouples utilization control from fairness control with help of additional information in congestion header. Both ECN and XCP need cooperation of end-host and routers, and can be effective when all routers in the path have the capability.

There are other new AQM algorithms that we do not treat in this paper. An AQM based on sliding mode variable structure control (SMVS) is one of them [21]. Since SMVS is insensitive to system dynamic parameters, it outperforms PIC in responsivity and robustness against the disturbance. A modified ARED is also proposed in [19]. By using AIMD algorithm to adapt $p_{\max}$ and controlling queue length at a target value, it maintains a predictable average queue size and reduces parameter sensitivity.

VII. CONCLUSIONS

We have presented the design and analysis of hybrid RED. HRED can be easily configured to meet requirements for queueing delay, jitter, and response time through its parameters, which are easier to understand and more intuitively meaningful than other AQM algorithms. HRED takes over dual-threshold model to control the queueing delay, and employs a linear mapping from instantaneous queue length to drop probability across all values of queue length. It also decouples queue length management from adaptation to dynamic load, handling changes in load with a simple control algorithm.

We provided a simple analysis to derive stability criteria for HRED, illustrating that the conservative and asymmetric configuration can be used for stable operation. We also derived a relationship of control parameters with stability, the tolerant amount of short-lived connections, and the response time to changes in load, allowing tradeoffs among them.

Simulation and implementation results for several AQM algorithms demonstrated that HRED is less dependent on traffic loads than other AQM algorithms, and is robust to WEB traffic while providing stable, predictable queueing delay. The simulation results also illuminate the differences in response time between Blue, PIC, AVQ, and HRED. While PIC and HRED are provably stable, response times of HRED are much better than those of PIC in dynamic traffic loads. Although AVQ is also stable and has quick response time, it does not provide a simple means to control delay and jitter, and it has queue properties that are dependent on traffic loads, and can not avoid TCP lock-out behavior.

ACKNOWLEDGEMENTS

This research was supported in part by the University IT Research Center Project and the ubiquitous Autonomic Computing and Network Project, Ministry of Information and Communication, in Korea, and by National Science Foundation grant ACI-9984492.

REFERENCES

[1] B. Braden, Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang, "Recommendations on queue management and congestion avoidance in the internet," RFC 2309, Apr. 1998.
[2] M. May, J. Bolot, C. Diot, and B. Lyles, "Reasons not to deploy RED," in *Proc. IWQoS'99*, June 1999.
[3] M. Christiansen, K. Jeffay, D. Ott, and F. Smith, "Tuning RED for web traffic," in *Proc. SIGCOMM 2000*, Sept. 2000.
[4] V. Paxson, "End-to-end Internet packet dynamics," *IEEE/ACM Trans. Networking*, vol. 7, no. 3, June 1999.
[5] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Trans. Networking*, vol. 1, no. 4, Aug. 1993.
[6] V. Jacobson, "Congestion avoidance and control," in *Proc. SIGCOMM'88*, Aug. 1988.
[7] K. Ramakrishnan and S. Floyd, "A proposal to add explicit congestion notification (ECN) to IP," RFC 2481, Jan. 1999.
[8] C. Hollot, V. Misra, D. Towsley, and W. Gong, "A control theoretic analysis of RED," in *Proc. INFOCOM 2001*, Apr. 2001.
[9] P. Ranjan, E. Abed, and R. La, "Nonlinear instabilities in TCP-RED," in *Proc. INFOCOM 2002*, June 2002.
[10] S. Low, F. Paganini, J. Wang, S. Adlakha, and J. Doyle, "Dynamics of TCP/RED and a scalable control," in *Proc. INFOCOM 2002*, June 2002.
[11] T. J. Ott, T. V. Lakshman, and L. H. Wong, "SRED: Stabilized RED," in *Proc. INFOCOM'99*, Mar. 1999.
[12] W. Feng, D. Kandlur, D. Saha, and K. Shin, "A self-configuring RED gateway," in *Proc. INFOCOM'99*, Mar. 1999.
[13] W. Feng, D. Kandlur, D. Saha, and K. Shin, "Blue: An alternative approach to active queue management algorithms," in *Proc. NOSSDAV 2001*, June 2001.
[14] C. Hollot, V. Misra, D. Towsley, and W. Gong, "On designing improved controllers for AQM routers supporting TCP flows," in *Proc. INFOCOM 2001*, Apr. 2001.
[15] S. Athuraliya, V. Li, S. Low, and Q. Yin, "REM: Active queue management," *IEEE Network*, May 2001.
[16] S. Kunniyur and R. Srikant, "Analysis and design of an adaptive virtual queue (AVQ) algorithm for active queue management," in *Proc. SIGCOMM 2001*, Aug. 2001.
[17] C. Joo, and S. Bahk, "Scalability problems of RED," *IEE Electron. Lett.*, vol. 38, no. 21, Oct., 2002.
[18] S. Floyd, "Recommendation on using the gentle variant of RED," available at http://www.aciri.org/floyd/red/gentle.html, Mar. 2000.
[19] S. Floyd, R. Gummadi, and S. Shenker "Adaptive RED: An algorithm for increasing the robustness of RED's active queue management," under submission, Aug. 2001.
[20] M. May, T. Bonald, and J. Bolot, "Analytic evaluation of RED performance," in *Proc. INFOCOM 2000*, Mar. 2000.
[21] F. Y. Ren, X. H. Yin, Y. Ren, and F. B. Wang, "A robust active queue management algorithm based on sliding mode variable structure control," in *Proc. INFOCOM 2002*, June 2002.
[22] G. Franklin, J. Powell, and A. Emami-Naeini, *Feedback Control of Dynamic Systems*, Addison Wesley, 3rd ed., 1994.
[23] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP throughput: A simple model and its empirical validation," in *Proc. SIGCOMM'98*, Sept. 1998.
[24] V. Firoiu and M. Borden, "A study of active queue management for congestion control," in *Proc. INFOCOM 2000*, Mar. 2000.
[25] The UCB/LBNL/VINT Network Simulator (NS), available at http://www-mash.cs.berkeley.edu/ns.
[26] S. Floyd, "RED: Discussions of setting parameters," available at http://www.aciri.org/floyd/REDparameters.txt, Nov., 1997.
[27] S. Athuraliya, "A Note on parameter values of REM with Reno-like algorithms," available at http://netlab.caltech.edu/pub/papers/REMparameter.pdf.
[28] P. Bagal, S. Kalyanaraman, and B. Packer, "Comparative study of RED, ECN and TCP rate control," Technical Report, Mar. 1999.
[29] D. Katabi, M. Handley, and C. Rohrs, "Internet congestion control for future high bandwidth-delay product environments," in *Proc. SIGCOMM 2002*, Aug. 2002.

Changhee Joo received B.S., M.S., and Ph.D. degrees in electrical engineering and computer science from Seoul National University in 1998, 2000, and 2005, respectively. From 2005, he is working with the department of electrical and computer engineering, Purdue University as a researching staff. His interests are protocol design and performance analysis in communication networks.

Saewoong Bahk received B.S. and M.B. degrees in electrical engineering from Seoul National University in 1984 and 1986, respectively, and the Ph.D. degree from the University of Pennsylvania in 1991. From 1991 through 1994, he was with the department of network operations systems at AT&T Bell Laboratories as an MTS where he worked for AT&T network management. In 1994, he joined the school of electrical engineering at Seoul National University and currently serves as a professor. His areas of interests include performance analysis of communication networks and network security.

Steven S. Lumetta received A.B. in Physics in 1994 and M.S. and Ph.D. degrees in computer science from University of California, Berkeley in 1994 and 1998. He is an associate professor of electrical and computer engineering, an affiliate associate professor of computer science, and a research associate professor in the Coordinated Science Laboratory at the University of Illinois at Urbana-Champaign. Lumetta's research interests are in networking, computer systems, and digital system testing.