Differentiated Bandwidth Allocation with TCP Protection in Core Routers

Shushan Wen, *Student Member*, *IEEE*, Yuguang Fang, *Fellow*, *IEEE*, and Hairong Sun, *Senior Member*, *IEEE*

Abstract—Differentiated Services (DiffServ) networks have received wide attention for several years. They categorize routers into edge routers and core routers. In core routers, one of the technological challenges is to implement differentiated bandwidth allocation and TCP protection with low complexity, where conventional per-flow queueing is costly. In this paper, we present an Active Queue Management (AQM) scheme called CHOKeW, named after previous work CHOKe that is effective to protect TCP flows. A method is borrowed from CHOKe that draws a packet at random from the buffer, compares it with the arriving packet, and drops both if they are from the same flow (we call this "matched drops"). CHOKeW enhances the drawing function by adjusting the maximum number of draws based on the priority of the new arrival and the current status of network congestion. The number of parameters that CHOKeW needs to maintain is determined by the number of priority levels being supported by the network, which usually has a small limited value. With respect to the number of flows (say N) going through the router in the core networks, both the memory-requirement complexity and the per-packet-processing complexity for CHOKeW is O(1), as compared to $\mathrm{O}(N)$ and usually greater than O(1), respectively, which has been seen in conventional per-flow schemes. In order to explain the features of CHOKeW, an analytical model is used, followed by running a series of simulations to evaluate the performance. We show that under a variety of congestion scenarios, CHOKeW is able to 1) support differentiated bandwidth allocation by affording a larger bandwidth share to higher priority flows, 2) provide the flows in the same priority with better fairness than conventional stateless AQM schemes such as RED and BLUE, 3) maintain high link utilization as well as short queue length, and 4) protect TCP flows by restricting the bandwidth share of high-speed unresponsive flows.

Index Terms—Router, network operations.

1 Introduction

Problems associated with Quality of Services (QoS) in the Internet have been investigated for years but have not been solved completely. One of the technological challenges is to introduce a reliable and a cost-effective method to support multiple services at different priority levels within core networks that can support thousands of flows.[1]

In recent years, many QoS models, such as Service Marking [1], Label Switching [2], [3], Integrated Services/RSVP [4], [5], and Differentiated Services (DiffServ) [6] have been proposed. Each of these models has their own unique features and flaws.

In Service Marking, a method called "precedence marking" is used to record the priority value within a packet header. However, the service request is only associated with each individual packet and does not consider the aggregate forwarding behavior of a flow. The flow behavior is nevertheless critical to implement QoS. The second model, Label Switching, including Multiprotocol Label Switching (MPLS) [7], is designed in a way that supports packet delivery. In this model, finer granularity resource allocation is available, but scalability becomes a problem in large networks. In the worst scenario, it scales in proportion with the square of the number of edge routers. In addition, the basic infrastructure of Label Switching is built by Asynchronous Transfer Mode (ATM) and Frame Relay technology. In order to upgrade current IP routers to Label Switching routers, more overhead (such as address mapping, assembly and disassembly of IP packets, and adding and removing cell/frame control fields) must be handled appropriately. Integrated Services/RSVP relies upon traditional datagram networks, but it also has a scalability problem due to the necessity to establish packet classification and to maintain the forwarding state of the concurrent reservations on each router. DiffServ is a refinement to Service Marking, and it provides a variety of services for IP packets based on their per-hop behaviors (PHBs) [6]. Because of its simplicity and scalability, DiffServ has caught the most attention nowadays.

In general, routers in the DiffServ architecture, similar to those proposed in Core-Stateless Fair Queueing (CSFQ) [8], are divided into two categories: edge (boundary) routers and core (interior) routers. Sophisticated operations, such as per-flow classification and marking, are implemented at edge routers. In other words, core routers do not necessarily maintain per-flow states; instead, they only need to forward

1. The meaning of a flow depends on the definition. An IPv6 header already includes a flow-ID field; for an IPv4 packet, one practical definition, for example, is the combination of source and destination addresses.

- *S. Wen and Y. Fang are with the Department of Electrical and Computer Engineering, University of Florida, 435 Engineering Building, P.O. Box 116130, Gainesville, FL 32611. E-mail: wen@winet.ece.ufl.edu, shushanwen@gmail.com, fang@ece.ufl.edu.*
- *H. Sun is with Sun Microsystems, Broomfield, CO 80021. E-mail: hairong.sun@sun.com.*

Manuscript received 7 May 2006; revised 5 May 2007; accepted 22 Apr. 2008; published online 30 Apr. 2008.
Recommended for acceptance by Y. Oruc.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0117-0506.
Digital Object Identifier no. 10.1109/TPDS.2008.71.

the packets according to the indexed PHB values that are predefined. These values are marked in the Differentiated Services fields (DS fields) in packet headers [6], [9]. For example, Assured Forwarding [10] defined a PHB group, and each packet is assigned a level of drop precedence. Thus, packets with primary importance based on their PHB values encounter relatively low-dropping probability. The implementation of an Active Queue Management (AQM) to conduct the dropping, however, is not specified in [10].

When we design an AQM scheme, the performance has to be investigated along with Transmission Control Protocol (TCP), taking account of the fact that almost all error-sensitive data in the Internet are transmitted by TCP, and the dynamics of TCP has unavoidable interactions with the dropping scheme.

In order to incorporate the priority services[2] of DiffServ into TCP, the following technical problems must be solved: 1) TCP protection and 2) bandwidth differentiation. We discuss them in the following sections.

1.1 Previous Work

The importance of TCP protection has been discussed by Floyd and Fall [12]. They predicted that the Internet would collapse if there was no mechanism to protect TCP flows. In the worst case, the resources of routers would be consumed with packet forwarding, even though no packet is useful for receivers because the long delay and heavy losses suffered by the flows are beyond the acceptable limits of the applications. In the meantime, the bandwidth would be completely occupied by unresponsive senders that do not reduce the sending rates even after their packets are dropped by the congested routers [12].

Conventional AQM algorithms such as Random Early Detection (RED) [13] and BLUE [14] cannot protect TCP flows. It is strongly suggested that novel AQM schemes be designed for TCP protection in routers [12], [15]. Cho [16] proposed a mechanism, which uses a "flow-valve" filter for RED to punish non-TCP-friendly flows. However, this approach has to reserve three parameters for each flow, which significantly increases the memory requirement. In [17], Mahajan and Floyd described a simpler scheme, known as RED with Preferential Dropping (RED-PD), in which the drop history of RED is used to help identify non-TCP-friendly flows based on the observation where flows at higher speeds usually have more packet drops in RED. RED-PD is also a per-flow scheme and at least one parameter needs to be reserved for each flow to record the number of drops.

When compared with previous methods including conventional per-flow schemes, the implementation design of CHOKe [18], proposed by Pan et al., is simple, and it does not require per-flow state maintenance. CHOKe serves as an enhancement filter for RED in which a buffered packet is drawn at random and compared with an arriving packet. If both packets come from the same flow, they are dropped as a pair (hence, we call this "matched drops"); otherwise, the arriving packet is delivered to RED. Note that a packet that has passed CHOKe may still be dropped by RED. The validity of CHOKe has been explained using an analytical model by Tang et al. [19].

Besides TCP protection, some research has investigated the relationship between the priority of flows and the magnitude of bandwidth differentiation. RED with In/Out bit (RIO) [20], presented by Clark and Fang, is one of the most popular schemes designed for bandwidth differentiation. It uses two sets of RED parameters to differentiate high-priority traffic (marked as "in") from low-priority traffic (marked as "out"). The parameter set for "in" traffic usually includes higher queue thresholds, which results in a smaller dropping probability.

Some scheduling schemes, such as Weighted Fair Queueing (WFQ) [21] and other packet approximation of the Generalized Processor Sharing (GPS) model [22] may also support differentiated bandwidth allocation.

1.2 Our Contributions

Even though the functions of TCP protection and bandwidth differentiation are of crucial importance for the current Internet, we have not seen that any previous work has a mechanism to support both functions simultaneously and effectively. Thus, we design a novel scheme and our main contributions are

1. integrating TCP protection and bandwidth differentiation in one single scheme, with low complexity,
2. investigating TCP protection in network scenarios that support priority,
3. preventing TCP starvation, and
4. improving fairness using our scheme.

CHOKe is simple and works well for TCP protection, but it supports only best-effort traffic. In this paper, we use the concept of matched drops to design another scheme called CHOKeW. The letter W represents a function that supports multiple weights for bandwidth differentiation.

In DiffServ networks, TCP protection has to be investigated along with priority of flows. It has three scenarios: first, protecting TCP flows in higher priority from high-speed unresponsive flows in lower priority, second, protecting TCP flows from high-speed unresponsive flows in the same priority, and third, protecting TCP flows in lower priority from high-speed unresponsive flows in higher priority. Since CHOKeW is designed for allocating a greater bandwidth share to higher priority flows, if TCP protection is effective in the third scenario, it should also be effective in the first and second scenarios.

In RIO, an "out" flow may be starved because there is no mechanism to guarantee the bandwidth share for low-priority traffic [23], which is a major disadvantage of RIO. Our scheme uses matched drops to control the bandwidth share. When a low-priority TCP flow only has a small bandwidth share, the responsiveness of TCP can lead to a small backlog for this flow in the buffer. The packets from this flow will unlikely be dropped, so this flow will not be starved.

The main disadvantage of scheduling schemes such as WFQ is that they require constant per-flow state maintenance, which is not cost-effective in core networks as it causes memory-requirement complexity $O(N)$ and

2. As [11] proposed, a set of priority services can be applied to modeling and analyzing DiffServ by mapping the PHBs that receive better services into the higher priority levels. In the rest of this paper, we use "priority levels" to represent PHBs for general purposes.

per-packet-processing complexity that is usually larger than O(1), where N denotes the number of flows being served by the router.[3] Our scheme does not maintain per-flow states, and the packet processing time is independent of N. Both the memory-requirement complexity and the per-packet-processing complexity of CHOKeW is O(1).

Moreover, CHOKeW uses First-Come-First-Served (FCFS) scheduling, which shortens the tail of the delay distribution [28], and lets packets arriving in a small burst be transmitted in a burst. Many applications in the Internet, such as TELNET, benefit from this feature. Schedulers similar to WFQ or DRR, however, interweave the packets from different queues in the forwarding process, which diminishes this feature.

In addition, by using CHOKeW, we expect better fairness among the flows with the same priority.

The rest of the paper is organized as follows: Section 2 describes the CHOKeW algorithm. Section 3 derives the equations for the steady state and explains the features and effectiveness of CHOKeW, such as fairness and bandwidth differentiation. Section 4 presents and discusses the simulation results, including the effect of supporting two and even more priority levels, TCP protection, the performance of TCP Reno in CHOKeW, a comparison with CHOKeW-RED (CHOKeW with RED module), and a comparison with CHOKeW-avg (CHOKeW with a module to calculate the average queue length by EWMA). Section 5.1 discusses the issues concerning the implementation and proposes an extended matched drop algorithm for CHOKeW designed for some special scenarios. We conclude our paper in Section 6.

2 CHOKeW Algorithm

CHOKeW uses the strategy of matched drops presented by CHOKe [18] to protect TCP flows. Like CHOKe, without the necessity of maintaining per-flow states, CHOKeW is capable of working in core networks where a myriad of flows are served.

More importantly, CHOKeW supports differentiated bandwidth allocation for traffic with different priority weights. Each priority weight corresponds to one of the priority levels; a greater priority weight represents a higher priority level.

Although CHOKeW borrows the idea of matched drops from CHOKe for TCP protection, there are significant differences between these two algorithms. First of all, the goal of CHOKe is to block high-speed unresponsive flows with the help of RED to inform TCP flows of network congestion, whereas CHOKeW is designed for supporting differentiated bandwidth allocation with the assistance of matched drops that are also able to protect TCP flows.

While Pan et al. [18] suggested to draw more than one packet if there are multiple unresponsive flows, they did not provide further solutions. In CHOKeW, the adjustable number of draws is not only used for restricting the bandwidth share of high-speed unresponsive flows but also used as signals to inform TCP of the congestion status. In order to avoid functional redundancy, CHOKeW is not combined with RED since RED is also designed to inform TCP of congestion. Thus, we say that CHOKeW is an independent AQM scheme, instead of an enhancement filter for RED. To demonstrate that RED is not an essential component for the effectiveness of CHOKeW, the comparison between the performance of CHOKeW and that of CHOKeW-RED (i.e., CHOKeW with RED) is shown in Section 4.6.

2.1 Drawing Factor

In order to determine when to draw a packet (or packets) and how many packets are possibly drawn from the buffer, we introduce a variable, called the drawing factor.

Roughly speaking, we may interpret p_i as the maximum number of random draws from the buffer upon a packet arrival from flow i. The precise meaning is discussed below.

Assume that the number of active flows served by a CHOKeW router is N, and the number of priority levels supported by the router is M. Let w_i $(w_i \geq 1)$ be the priority weight of flow i $(i = 1, 2, \cdots, N)$, and $w_{(k)}$ $(k = 1, 2, \cdots, M)$ be the weight of priority level k. If flow i is at priority level k, then $w_i = w_{(k)}$. All flows at the same priority level have the same priority weight. If $w_{(k)} > w_{(l)}$, we say that flows at priority level k have higher priority than flows at priority level l, or simply, priority level k is higher than priority level l.

Let p_0 denote the basic drawing factor. The drawing factor used for flow i is calculated as follows:

$$p_i = p_0 / w_i. \tag{1}$$

As $w_i \geq 1$, we know that p_0 is the upper bound of p_i.

From (1), we also know that if flow i has higher priority than flow j $(w_i > w_j)$, flow i will get a smaller drawing factor $(p_i < p_j)$ and, hence, will have a lower possibility of becoming the victim of matched drops. This is the basic mechanism for supporting bandwidth differentiation in CHOKeW (further explained in Section 3.4).

The precise meaning of drawing factor p_i depends upon its value. It can be categorized into two cases:

Case 1. When $0 \leq p_i < 1$, p_i represents the probability of drawing one packet from the buffer at random for comparison.

Case 2. When $p_i \geq 1$, p_i consists of two parts, and we may rewrite p_i as

$$p_i = m_i + f_i, \tag{2}$$

where $m_i \in \mathbb{N}$ (the set of nonnegative integers) represents the integral part with the value of $\lfloor p_i \rfloor$ (the largest integer $\leq p_i$) and f_i the fractional part of p_i. In this case, at the most m_i or $m_i + 1$ packets in the buffer may be drawn for comparison. Let d_i denote the maximum number of random draws. We have

$$\begin{cases} Prob[d_i = m_i + 1] = f_i, \\ Prob[d_i = m_i] = 1 - f_i. \end{cases}$$

3. For example, according to Ramabhadran and Pasquale [24], the per-packet-processing complexity is $O(N)$ for WFQ, $O(\log N)$ for WF^2Q [25], and $O(\log \log N)$ for Leap Forward Virtual Clock [26]. Deficit Round Robin (DRR) [27] reduces the per-packet-processing complexity to $O(1)$, but its memory-requirement complexity is still $O(N)$ when the number of logic queues is comparable to the number of active flows, in order to obtain desired performance. By contrast, stateless AQM schemes such as RED and BLUE usually have complexity O(1).

```
Initialization:
p_0 ← 0

For each packet pkt arrival
(1) L ← L + l_a
(2) Update p_0 (see Fig.2)
(3) IF pkt is at priority level k
    THEN p ← p_0/w_(k), m ← ⌊p⌋, f = p − m
(4) Generate a random number v ∈ [0, 1)
    IF v < f
    THEN m ← m + 1
(5) IF L > L_th
    THEN
      WHILE m > 0
        m ← m − 1
        Draw packet pkt' from the buffer at random
        IF ξ_a=ξ_b
        THEN
          L ← L − l_a − l_b, drop pkt' and pkt
          RETURN /*wait for the next arrival*/
        ELSE keep pkt' intact
(6) IF L > L_lim /*buffer is full*/
    THEN L ← L − l_a, drop pkt
    ELSE let pkt enter the buffer

Parameters:
ξ_a: Flow ID of the arriving packet
ξ_b: Flow ID of the packet drawn from buffer
l_a: Size of the arriving packet
l_b: Size of the packet drawn from the buffer
L: Queue length
L_lim: Buffer limit
L_th: Queue length threshold
```

Fig. 1. The CHOKeW algorithm.

2.2 Algorithm Description

The algorithm of drawing packets is described in Fig. 1. One may notice that in this figure, we use p, f, and m, instead of p_i, f_i, and m_i, since those variables can be reused by all flows, as well as all priority levels. For simplicity of implementation, we use m to represent m_i (before step 4) and d_i (after step 4), rather than create one more variable for d_i.

The congestion status of a router may become either heavier or lighter after a period of time, since circumstances (e.g., the number of users, the application types, and the traffic priority) constantly change. In order to cooperate with TCP and to improve the system performance, an AQM scheme such as RED [13], needs to inform TCP senders to lower their sending rates by dropping more packets when the network congestion becomes worse. Unlike CHOKe [18], CHOKeW does not have to work with RED in order to function properly. Instead, CHOKeW can adaptively update p_0 based on the congestion status. The updating process is shown in Fig. 2, which details step 2 in Fig. 1. The combination of Figs. 1 and 2 provides a complete description of the CHOKeW algorithm.

CHOKeW updates p_0 upon each packet arrival but activates matched drops only when the queue length L is longer than the threshold L_{th} (step 5 in Fig. 1). Three queue length thresholds are applied to CHOKeW: L_{th} is the threshold of activating matched drops, L^+ is that of increasing p_0, and L^- is that of decreasing p_0.

As the buffer is used to absorb bursty traffic [15], we set $L_{th} > 0$, so that the short bursty traffic can enter the buffer without suffering any packet drops when the queue length L is less than L_{th} (although p_0 may be larger than 0 for historical reasons). A large L_{th} can absorb more bursty traffic but may lead to a slow reaction to the congestion, while a small L_{th} can activate matched drops in the early stage of congestion but may cause low link utilization.

```
IF L < L^−
THEN
  p_0 ← p_0 − p^−
  IF p_0 < 0
  THEN p_0 ← 0
IF L > L^+
THEN p_0 ← p_0 + p^+

Parameters:
L: Queue length
L^+: Queue length threshold of increasing p_0
L^−: Queue length threshold of decreasing p_0
L_th < L^− < L^+
p_0: Basic drawing factor
p^+: Step length of increasing p_0
p^−: Step length of decreasing p_0
```

Fig. 2. The algorithm of updating p_0.

TABLE 1
The State of CHOKeW versus the Range of L

State of CHOKeW	Range of L			
	$[0, L_{th}]$	(L_{th}, L^-)	$[L^-, L^+]$	$(L^+, L_{lim}]$
Matched Drops	Inactive	Active		
$p_0 \leftarrow$	$max\{0, p_0 - p^-\}$		p_0	$p_0 + p^+$

When $L \in [L^-, L^+]$, the network congestion status is considered to be stable, and p_0 maintains the same value as before (i.e., the algorithm shown in Fig. 2 does not adjust the value of p_0). Only when $L > L^+$, the congestion is considered to be heavy, and p_0 is increased by p^+ each time. The alleviation of network congestion is represented by $L < L^-$, and as adaptation, p_0 is reduced by p^- each time. We keep $L_{th} < L^-$ so that the matched drops are still active when p_0 starts becoming smaller, which prevents matched drops from being completely turned off suddenly and endows the algorithm with higher stability.

The state of CHOKeW can be described by the activation of matched drops and the process of updating p_0, which is further determined by the range the current queue length L falls into, shown in Table 1.

2.3 Complexity

One advantage of using CHOKeW is that it is easily able to prioritize each packet based on the value of the DS field, without the aid of the flow ID.[4] Therefore, when CHOKeW is applied in core routers, priority becomes a packet feature. In terms of service qualities in the core network, packets from different flows shall equally be served if they have the same priority; on the other hand, packets from the same flow may be treated differently if their priority is different (e.g., some packets are remarked by edge routers).

4. In CHOKeW, the flow ID is only used to check whether two packets are from the same flow. This operation (XOR) can be executed efficiently by hardware.

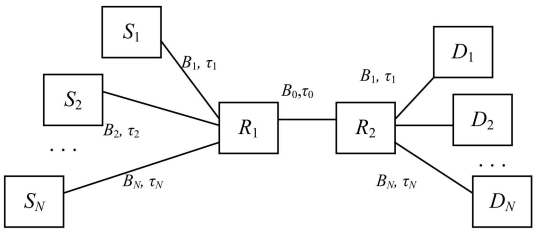


Fig. 3. Network topology.

CHOKeW needs to remember only $w_{(k)}$ for each predefined priority level k ($k = 1, 2, \cdots, M$), instead of some variables for each flow i ($i = 1, 2, \cdots, N$). The complexity of CHOKeW is only affected by M. In DiffServ networks, it is reasonable to expect that M will never be a large value in the foreseeable future, i.e., $M \ll N$. Therefore, with respect to N, which can be quite large in core networks, the memory-requirement complexity, as well as the per-packet-processing complexity, of CHOKeW is O(1), while for conventional per-flow schemes, the memory-requirement complexity is $O(N)$, and the per-packet-processing complexity is usually larger than O(1) [24].

3 Model

In previous work, Tang et al. [19] proposed a model to explain the effectiveness of CHOKe. Using matched drops, CHOKe produces a "leaky buffer," where packets may be dropped when they move forward in the queue, which may result in a flow that maintains many packets in the queue but can obtain only a small portion of bandwidth share. In this way, TCP protection takes effect on high-speed unresponsive flows [19].

For CHOKeW, we need a model to explain not only how to protect TCP flows (as shown in [19]) but also how to differentiate the bandwidth share.

The network topology shown in Fig. 3 is used for our model. In this figure, two routers, R_1 and R_2, are connected to N source nodes (S_i, $i = 1, 2, \cdots, N$) and N destination nodes (D_i), respectively. The R_1–R_2 link, with bandwidth B_0 and propagation delay τ_0, allows all flows to go through. B_i and τ_i denotes the bandwidth and the propagation delay of each link connected to S_i or D_i, respectively. As we are interested in the network performance under a heavy load, we always let $B_0 < B_i$, so that the link between the two routers becomes a bottleneck.

The symbols that are used for performance analysis are listed in Table 2.

TABLE 2
List of Symbols

Symbol	Definition
α_i	combination of factors other than q_i that affect flow i
η_i	entry probability for packets from flow i
λ	average aggregate arrival rate
λ'	average aggregate downstream bandwidth
λ_i	average arrival rate of flow i
λ_i'	average downstream bandwidth used by flow i
μ_i	average entry rate of flow i
ρ_i	passing probability for packets from flow i
D	average waiting time
f_i	fractional part of p_i
L	queue length
L'	queue length only counting survivable packets
L_i	packet backlog from flow i
L_i'	backlog for survivable packet from flow i
m_i	integral part of p_i
p_0	basic drawing factor
p_i	drawing factor for flow i
q_i	dropping probability for flow i
r_i	matching probability for flow i
s_i	survival probability for packets from flow i
w_i	priority weight of flow i

3.1 Some Useful Probabilities

In the CHOKeW router, for flow i, let r_i be the probability that matched drops occur at one draw (matching probability in short), which is dependent of the current queue length L and the number of packets from flow i in the queue (i.e., the packet backlog from flow i, denoted by L_i). The following equation presented in [19] can also be used for CHOKeW:

$$r_i = L_i/L. \tag{3}$$

Now, we focus on the features of matched drops. Assuming that the buffer has an unlimited size and thus packet dropping is due to matched drops instead of overflow, we can calculate the probability that a packet from flow i is allowed to enter the queue upon its arrival, denoted by η_i:

$$\eta_i = (1 - r_i)^{m_i}(1 - f_i r_i), \tag{4}$$

where $m_i = \lfloor p_i \rfloor$, and $f_i = p_i - m_i$, as mentioned before.

In (4), $(1 - r_i)^{m_i}$ is the probability of no matched drops in the first m_i draws. After the completion of the first m_i draws, the value of f_i stochastically determines whether one more packet is drawn. The probability of no further draw is $(1 - f_i)$, and the probability that one more packet is drawn, but no matched drops occur is $f_i(1 - r_i)$. Therefore, the probability that no matched drops occur is $(1 - f_i) + f_i(1 - r_i) = 1 - f_i r_i$.

We rewrite $(1 - r_i)^{f_i}$ as its Maclaurin Series:

$$(1 - r_i)^{f_i} = 1 - f_i r_i + o(r_i^2).$$

Assuming the core network serves a vast number of flows, it is reasonable to say $r_i \ll 1$ for each responsive flow i.[5] We have $(1 - r_i)^{f_i} \approx 1 - f_i r_i$, and (4) can be rewritten as $\eta_i = (1 - r_i)^{m_i + f_i}$, or

$$\eta_i = (1 - r_i)^{p_i}. \tag{5}$$

5. We call a flow responsive if it avoids sending data at a high rate when the network is congested. A TCP flow is responsive, while a UDP flow is not.

For a packet from flow i, let s_i denote the probability that it survives the queue and q_i the dropping probability (either before or after the packet enters the queue).[6] We have

$$q_i = 1 - s_i. \tag{6}$$

For each packet arrival from flow i, the probability that it is dropped before entering the queue is $1 - \eta_i$. According to the rule of matched drops, a packet from the same flow, which is already in the buffer, should also be dropped if the arriving packet is dropped. Thus, in a steady state, we obtain $q_i = 2(1 - \eta_i)$, $0.5 \leq \eta_i \leq 1$. When $q_i = 1$, $\eta_i = 0.5$. In other words, when flow i is starved, the router still needs to let half of the arriving packets from flow i enter the queue, and the packets in the queue will be used to match the new arrivals in the future. On the other hand, if $\eta_i < 0.5$ temporarily, the number of packets that enter the queue cannot compensate the backlog for the packet losses from the queue, which causes the reduction of r_i until $r_i = 1 - 2^{-1/p_i}$ and accordingly $\eta_i = 0.5$.

By using (5) in it, we get

$$q_i = 2 - 2(1 - r_i)^{p_i}, \tag{7}$$

and from (6),

$$s_i = 2(1 - r_i)^{p_i} - 1. \tag{8}$$

After a packet enters the queue, it will either pass the queue successfully, or be dropped from the queue due to matched drops. The passing probability ρ_i satisfies $s_i = \eta_i \rho_i$. Using (5) and (8) in it, we get

$$\rho_i = 2 - \frac{1}{(1 - r_i)^{p_i}}.$$

In order for TCP protection, CHOKeW requires $0 \leq q_i < 1$ if flow i uses TCP. Equation (7) shows that as long as p_i does not exceed a certain range, CHOKeW can guarantee that flow i will not be starved, even if it may only have low priority. This feature offers CHOKeW advantages over RIO [23], which neither protects TCP flows, nor prevents the starvation of low-priority flows.

The CHOKeW algorithm ensures the satisfaction of the lower bound of q_i ($q_i \geq 0$). Now, we discuss the range of p_i to satisfy the upper bound of q_i (i.e., $q_i < 1$). From (7), we have

$$p_i < \frac{-1}{\log_2(1 - r_i)}. \tag{9}$$

Now, we discuss flow starvation base on the value of p_i. Given r_i, the upper bound of p_i shown in (9) corresponds to the threshold to starve flow i.

When all flows are TCP, they are able to reduce their sending rates and cooperate with each other when the network is only in mild congestion, which prevents p_i from further increasing. For instance, when r_i is equal to 0.01 (imagine 100 TCP flows share the queue length evenly), the upper bound of p_i is 68.9; in other words, the algorithm has to draw up to 69 packets before a TCP flow is starved, but due to the congestion avoidance mechanism in TCP, CHOKeW rarely needs to raise p_i to such a high value to control the congestion.

When some high-speed UDP flows exist with many TCP flows, which is the realistic scenario of the current Internet, r_j for UDP flow j can be much larger than r_i for TCP flow i, and thus, p_j can easily reach the threshold to block flow j. For example, in a simulation where five high-speed UDP flows exist with 100 TCP flows, each UDP flow has $r_j = 0.017$, so the threshold of p_j to block a UDP flow is 40.4. On the other hand, a TCP flow has $r_i = 0.009$ on the average, which corresponds to 76.7 for the upper bound of p_i. When all high-speed UDP flows are blocked, TCP flows are still very safe.

Formula (9) also explains why a flow in CHOKe (where $p_i \equiv 1$) that is not starved must have a backlog shorter than half of the queue length. This result is consistent with the conclusion in [19].

3.2 Steady-State Features of CHOKeW

In this section, assume that there are N independent flows going through the CHOKeW router, and the packet arrivals for each flow are Poisson.[7]

The packets arriving at the router can be categorized into two groups: 1) those that will be dropped and 2) those that will pass the queue. Let λ denote the average aggregate arrival rate for all flows, and λ' the average aggregate arrival rate for the packets that will pass the queue. Then, the average arrival rate for the packets that will be dropped is equal to $\lambda - \lambda'$. Similarly, L denotes the queue length as mentioned above, and L' the queue length only counting the packets that will not be dropped ("survivable packets" in short). Compared to the time that it takes to transmit (serve) a packet, the processing time to drop a packet is negligible. Little's Theorem [29] shows

$$D = L'/\lambda', \tag{10}$$

where D is the average waiting time for packets in the queue.

In order to analyze the feature of individual flows, for flow i ($i = 1, 2, \cdots, N$), let λ_i be the average arrival rate, and λ_i' be the average arrival rate only counting the packets that will survive the queue. Then

$$\lambda_i' = \lambda_i(1 - q_i). \tag{11}$$

As mentioned above, L_i denotes the backlog from flow i. Let L_i' be the backlog for the survivable packets from flow i. Then, these per-flow measurements have the following relationship with their aggregate counterparts: $\lambda = \sum_{i=1}^{N} \lambda_i$, $\lambda' = \sum_{i=1}^{N} \lambda_i'$, $L = \sum_{i=1}^{N} L_i$, and $L' = \sum_{i=1}^{N} L_i'$.

Based on the Poisson Arrivals See Time Average (PASTA) property of Poisson arrivals, packets from all flows have the same average waiting time in the queue (i.e., $D_i = D$, $i = 1, 2, \cdots, N$). Using Little's Theorem again, for flow i, we get

$$D = L_i'/\lambda_i'. \tag{12}$$

Using (10) in (12),

$$\frac{L_i'}{L'} = \frac{\lambda_i'}{\lambda'}. \tag{13}$$

6. Note that it is possible for a packet to become a matched-drop victim even after it has entered the queue.

7. Strictly speaking, the Poisson distribution is not a perfect representation of Internet traffic; nevertheless, it can provide some insights into the features of our algorithm.

The average number of packet drops from flow i during period D is $D\lambda_i q_i$. As packets from a flow are dropped in pairs (one before entering the queue and one after entering the queue), flow i has $D\lambda_i q_i/2$ packets dropped after entering the queue on the average. Thus, we have

$$\begin{cases} L_i = L_i' + D\lambda_i q_i/2, \\ L = L' + \frac{D}{2}\sum_{j=1}^{N} \lambda_j q_j. \end{cases}$$

Using (7), (11), (12), and (13) in it, we obtain

$$\begin{cases} L_i = D\lambda_i (1-r_i)^{p_i}, \\ L = D\sum_{j=1}^{N} \lambda_j (1-r_j)^{p_j}. \end{cases} \tag{14}$$

For flow i, let μ_i denote the average arrival rate only counting the packets entering the queue. Then, μ_i is determined by λ_i and η_i, i.e., $\mu_i = \lambda_i \eta_i$. Considering (5), we rewrite (14) as

$$\begin{cases} L_i = D\mu_i, \\ L = D\sum_{j=1}^{N} \mu_j \end{cases} \tag{15}$$

and rewrite (3) as $r_i = \mu_i \Big/ \left(\sum_{j=1}^{N} \mu_j\right)$.

Equation (15) can be interpreted as Little's Theorem used for a leaky buffer, where packets may be dropped before they reach the front of the queue, which is not a classical queueing system. From (15), we get an interesting result: even in a leaky buffer, the average waiting time is determined by the average queue length for flow i (or the average aggregate queue length) and the average arrival rate from flow i (or the average aggregate arrival rate) that only counts the packets entering the queue. The average waiting time is meaningful to the packets surviving the queue exclusively, whereas packets that are dropped after entering the queue still contribute to the queue length.

Below are the group of formulas that describe the steady-state features of CHOKeW:

$$\begin{cases} r_i = \frac{\mu_i}{\sum_{j=1}^{N} \mu_j}, & \text{(16a)} \\ q_i = 2 - 2(1-r_i)^{p_i}, & \text{(16b)} \\ \mu_i = \lambda_i (1-r_i)^{p_i}, & \text{(16c)} \\ \lambda_i' = \lambda_i (1-q_i). & \text{(16d)} \end{cases}$$

Note that they are based on Maclaurin approximation and thus are not exact, but we can use them for further analysis and obtain some insightful results.

3.3 Fairness

To demonstrate the fairness of our scheme, we study two TCP flows, i and j ($i \neq j$ and $i, j \in \{1, 2, \cdots, N\}$). In this section, we analyze the fairness under circumstances where flows have the same priority and, hence, the same drawing factor, denoted by p. The discussion of multiple priority levels is left to the next section.

From (16a), $r_i/r_j = \mu_i/\mu_j$. Using (16c) in it, we get

$$\frac{r_i}{r_j} = \frac{\lambda_i (1-r_i)^p}{\lambda_j (1-r_j)^p}. \tag{17}$$

Previous research (for example, [12] and [30]) has shown that the approximate sending rate of TCP is affected by the dropping probability, packet size, Round Trip Time (RTT), and other parameters such as the TCP version and the speed of users' computers. In this paper, we describe TCP sending rate as

$$\lambda_i = \alpha_i R(q_i), \tag{18}$$

where q_i is the dropping probability, and α_i ($\alpha_i > 0$) denotes the combination of other factors.[8] Because the sending rate of TCP decreases as network congestion worsens (indicated by a higher dropping probability), we have

$$\partial R/\partial q_i < 0. \tag{19}$$

When flow i and flow j have the same priority, our discussion covers two cases, distinguished by the equivalence of α_i and α_j.

Case 1. $\alpha_i = \alpha_j$.

When $\alpha_i = \alpha_j$, an intuitive solution to (17) is $\lambda_i = \lambda_j$ and $r_i = r_j$. From (16b) and (16d), we have $\lambda_i' = \lambda_j'$, i.e., flow i and flow j get the same amount of bandwidth share. We will show that this is the only solution.

Let

$$G = \frac{\lambda_i (1-r_i)^p}{\lambda_j (1-r_j)^p} - \frac{r_i}{r_j}.$$

Then, any solution to (17) is also a solution to $G = 0$.

In the core network, a router usually has to support a great number of flows and the downstream link bandwidth is shared among them. It is reasonable to assume that fluctuations in the backlog of a TCP flow do not significantly affect the backlog of other flows (i.e., $\partial r_j/\partial r_i \approx 0$, $i \neq j$). Then, we have

$$\frac{\partial G}{\partial r_i} = \frac{1}{\lambda_j (1-r_j)^p}\left[\frac{\partial \lambda_i}{\partial r_i}(1-r_i)^p - \lambda_i p (1-r_i)^{p-1}\right] - \frac{1}{r_j}.$$

Because $\frac{\partial \lambda_i}{\partial r_i} = \frac{\partial \lambda_i}{\partial q_i} \cdot \frac{\partial q_i}{\partial r_i} = 2\frac{\partial \lambda_i}{\partial q_i} p (1-r_i)^{p-1}$, and $\partial \lambda_i/\partial q_i < 0$ (derived from (18) and (19)), for any values of r_i ($0 < r_i < 1$) and p ($p > 0$), $\partial G/\partial r_i < 0$. In other words, G is a monotone decreasing function with respect to r_i. As a result, if there is a value of r_i satisfying $G = 0$, it must be the only solution to $G = 0$. Thus, the only steady state is maintained by $\lambda_i = \lambda_j$ and $r_i = r_j$, when TCP flows i and j have the same priority and the same factor α. This indicates that CHOKeW is capable of providing good fairness to flow i and flow j.

Case 2. $\alpha_i \neq \alpha_j$.

Let $(\lambda_i'/\lambda_j')_C$ and $(\lambda_i'/\lambda_j')_R$ denote the ratio of the average throughput of flow i to that of flow j for CHOKeW and for conventional stateless AQM schemes, respectively. By comparing $(\lambda_i'/\lambda_j')_C$ to $(\lambda_i'/\lambda_j')_R$, we will show that CHOKeW is able to provide better fairness, when $\alpha_i \neq \alpha_j$.

Among conventional stateless AQM schemes, RED determines the dropping probability according to the

8. The construction of (18) results from previous work. In [12], for instance, when a TCP session works in the nondelay mode, the sender's rate can be estimated by $\lambda_i = \frac{P_i}{T_i}\sqrt{\frac{3}{2q_i}}$, where P_i and T_i denote packet size and RTT of this flow, respectively.

average queue length, and BLUE calculates the dropping probability from packet loss and link idle events. In a steady state, for AQM schemes such as RED and BLUE, every flow has the similar dropping probability, denoted by q. Therefore, flow i has an average throughput of

$$\lambda_i' = \lambda_i(1-q) = (1-q)\alpha_i R(q),$$

and flow j has an average throughput of

$$\lambda_j' = (1-q)\alpha_j R(q).$$

For RED and BLUE,

$$\left(\lambda_i'/\lambda_j'\right)_R = \alpha_i/\alpha_j. \tag{20}$$

If $\alpha_i > \alpha_j$, $(\lambda_i'/\lambda_j')_R > 1$. Given an AQM scheme, the closer λ_i'/λ_j' is to 1, the better the fairness will be. For CHOKeW, if $(\lambda_i'/\lambda_j')_C$ is closer to 1 than $(\lambda_i'/\lambda_j')_R$, i.e., $(\lambda_i'/\lambda_j')_R > (\lambda_i'/\lambda_j')_C > 1$, we say CHOKeW is capable of providing better fairness.

From (16b), $R(q_i)$ in (18) can be rewritten as

$$R(q_i) = R(2-2(1-r_i)^{p_i}).$$

When flow i and flow j have the same priority, p, we define

$$\Lambda(r_l) \triangleq R(2-2(1-r_l)^p), \quad \text{for } l \in \{i,j\}.$$

Then, (18) can be rewritten as

$$\lambda_l = \alpha_l \Lambda(r_l), \quad \text{for } l \in \{i,j\}.$$

From (16b) and (16d),

$$\frac{\lambda_i'}{\lambda_j'} = \frac{\alpha_i \Lambda(r_i)[2(1-r_i)^p - 1]}{\alpha_j \Lambda(r_j)\left[2(1-r_j)^p - 1\right]}. \tag{21}$$

Our goal is to show that the right-hand side of (21) is less than α_i/α_j, if $\alpha_i > \alpha_j$. From (16a) and (16c),

$$r_i = \frac{\alpha_i \Lambda(r_i)(1-r_i)^p}{\alpha_i \Lambda(r_i)(1-r_i)^p + \sum_{k=1,k\neq i}^{N} \mu_k}$$

and, hence,

$$\frac{\partial r_i}{\partial \alpha_i} = \beta \Lambda(r_i)(1-r_i)^p \left[1 - \beta\alpha_i \frac{\partial \Lambda}{\partial r_i}(1-r_i)^p + \beta\alpha_i \Lambda(r_i) p (1-r_i)^{p-1}\right]^{-1},$$

where $\beta = \sum_{k=1,k\neq i}^{N} \mu_k \Big/ \left(\sum_{k=1}^{N} \mu_k\right)^2$.

From

$$\frac{\partial \Lambda}{\partial r_i} = \frac{\partial R}{\partial q_i} \cdot \frac{\partial q_i}{\partial r_i} = \frac{\partial R}{\partial q_i}\left[2p(1-r_i)^{p-1}\right] < 0,$$

we see $\partial r_i/\partial \alpha_i > 0$, which means when $\alpha_i > \alpha_j$, we have $r_i > r_j$ and $\Lambda(r_i) < \Lambda(r_j)$. Using these results in (21), for CHOKeW,

$$\left(\lambda_i'/\lambda_j'\right)_C < (\alpha_i/\alpha_j). \tag{22}$$

A comparison between (20) and (22) proves that CHOKeW provides better fairness than RED and BLUE.

3.4 Bandwidth Differentiation

For any two TCP flows i and j $(i \neq j)$, if $\alpha_i = \alpha_j$, and $w_i < w_j$ (from (1), $p_i > p_j$), CHOKeW allocates a smaller bandwidth share to flow i than to flow j, i.e., $\lambda_i' < \lambda_j'$. This seems to be an intuitive strategy, but we also noticed that the interaction among p_i, r_i, and q_i may cause some confusion. The dropping probability of flow i in CHOKeW, q_i, is not only determined by p_i but also by r_i. Furthermore, the effects of r_i and p_i are inverse—a larger value of p_i results in a larger q_i, but at the same time it leads to a smaller r_i, which may produce a smaller q_i. To clear up the confusion, we only need to show that a larger value of p_i leads to a smaller value of bandwidth share, λ_i', which is equivalent to showing $\partial\lambda_i'/\partial p_i < 0$. From (16d), (18), and (19), we get $\partial\lambda_i'/\partial q_i < 0$. From the Chain Rule

$$\frac{\partial \lambda_i'}{\partial p_i} = \frac{\partial \lambda_i'}{\partial q_i} \cdot \frac{\partial q_i}{\partial p_i},$$

we only need to show

$$\frac{\partial q_i}{\partial p_i} > 0.$$

Proof. According to the Chain Rule, we know

$$\frac{\partial q_i}{\partial p_i} = \frac{\partial q_i}{\partial r_i} \cdot \frac{\partial r_i}{\partial p_i} + \frac{\partial q_i}{\partial u} \cdot \frac{\partial u}{\partial p_i}, \tag{23}$$

where $u = p_i$. We introduce the symbol u to distinguish $\partial q_i/\partial u$ from $\partial q_i/\partial p_i$. We consider r_i as a constant in $\partial q_i/\partial u$ but not in $\partial q_i/\partial p_i$. From (16b),

$$\partial q_i/\partial u = -2(1-r_i)^{p_i}\ln(1-r_i), \tag{24}$$

and

$$\partial q_i/\partial r_i = 2p_i(1-r_i)^{p_i-1}. \tag{25}$$

According to (16a) and (16c), we have

$$\frac{\partial r_i}{\partial p_i} = \frac{\gamma_1}{\sum_{k=1}^{N} \mu_k + \lambda_i p_i (1-r_i)^{p_i-1}}, \tag{26}$$

where

$$\gamma_1 = (1-r_i)^{p_i}\left[\frac{\partial \lambda_i}{\partial q_i} \cdot \frac{\partial q_i}{\partial p_i} + \lambda_i \ln(1-r_i)\right].$$

Using (24), (25), and (26) in (23), we get

$$\frac{\partial q_i}{\partial p_i} = -\frac{2(1-r_i)^{p_i}\ln(1-r_i)\sum_{k=1}^{N}\mu_k}{\gamma_2} > 0,$$

where

$$\gamma_2 = \sum_{k=1}^{N}\mu_k + \lambda_i p_i (1-r_i)^{p_i-1} - 2\frac{\partial \lambda_i}{\partial q_i}(1-r_i)^{2p_i-1}p_i.$$

□

4 Performance Evaluation

To evaluate CHOKeW in various scenarios and to compare it with some other schemes, we implemented CHOKeW using ns simulator version 2 [31]. The metrics that are used in this section include TCP goodput, link utilization, average queue length, fairness index, and Relative Cumulative Frequency (RCF). The meanings of these metrics, if not self-explanatory, will be given in the sections when we use them.

The network topology is shown in Fig. 3, where $B_0 = 1$ Mb/s and $B_i = 10$ Mb/s $(i = 1, 2, \cdots, N)$. Unless specified otherwise, the link propagation delay $\tau_0 = \tau_i = 1$ ms. The buffer limit is 500 packets, and the mean packet size is 1,000 bytes. TCP flows are driven by FTP applications, and UDP flows are driven by CBR traffic. All TCPs are TCP SACK. Each simulation runs for 500 seconds.

Parameters of CHOKeW are set as follows: $L_{th} = 100$ packets, $L^- = 125$ packets, $L^+ = 175$ packets, $p^+ = 0.002$, and $p^- = 0.001$.

Parameters of RED are set as follows: the threshold of indicating light congestion $minth = 100$ packets, the threshold of indicating heavy congestion $maxth = 200$ packets, the EWMA weight is set to 0.002, the dropping probability threshold $pmax = 0.02$ (except in Section 4.6, where different values of $pmax$ are tested), and $gentle = true$ (which lets the dropping probability changes from $pmax$ to 1 when the average queue length increases from $maxth$ to $2maxth$).

Parameters of RIO include two sets of RED parameters, one for "out" traffic and the other one for "in" traffic. For "out" traffic, $minth\_out = 100$ packets, $maxth\_out = 200$ packets, and $pmax\_out = 0.02$. For "in" traffic, $minth\_in = 110$ packets, $maxth\_in = 210$ packets, and $pmax\_in = 0.01$ (except in Section 4.1, where we use multiple sets of parameters for comparison). Both $gentle\_out$ and $gentle\_in$ are set to $true$.

For parameters of BLUE, we set $\delta_1 = 0.0025$ (the step length of increasing the dropping probability), $\delta_2 = 0.00025$ (the step length of decreasing the dropping probability), and $freeze\_time = 100$ ms (the minimum time interval between two successive updates of the dropping probability).

4.1 Two Priority Levels with the Same Number of Flows

One of the main tasks of CHOKeW is supporting bandwidth differentiation for multiple priority levels without the need for per-flow states. We validate the effect of supporting two priority levels with the same number of flows in this section, two priority levels with different number of flows in the next section, and three or more priority levels in Section 4.3.

For TCP traffic, goodput is a well-known criterion to measure the performance. Here, we use the same definition for "goodput" as described in [12], i.e., "the bandwidth delivered to the receiver, excluding duplicate packets." On the other hand, for TCP, the bandwidth that the receiver gets is highly related to the number of duplicate packets generated by the flow. There is good evidence that the more congested the traffic in a TCP flow becomes, the higher dropping rate packets will have, and the more duplicate packets will be produced [32], [33]. Since TCP decreases the sending rate when packet drops are detected, it is reasonable to believe that a TCP flow with a larger bandwidth share also has higher goodput.

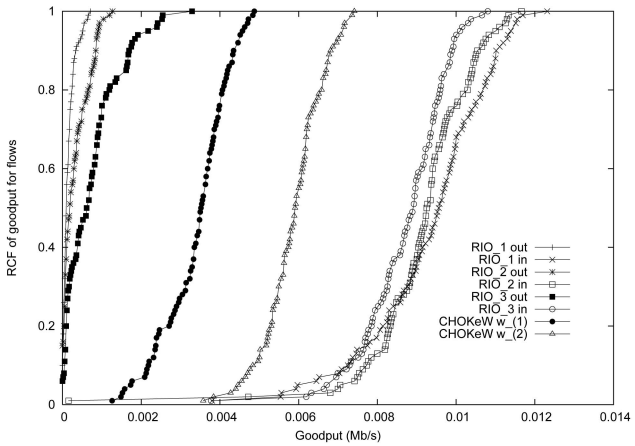


Fig. 4. The RCF of RIO and CHOKeW under a scenario of two priority levels.

As mentioned before, flow starvation often happens in RIO but is avoidable in CHOKeW. In order to quantify and compare the severity of flow starvation among different schemes, we record the RCF of goodput for flows at each priority level. For a scheme, the RCF of goodput g for flows at a specific priority level represents the number of flows that have goodput lower than or equal to g divided by the total number of flows in this priority.

We simulate 200 TCP flows. When CHOKeW is used, $w_{(1)} = 1$ and $w_{(2)} = 2$ are assigned to equal number of flows. When RIO is used, the number of "out" flows is also equal to the number of "in" flows. Fig. 4 illustrates the RCF of goodput for flows at each priority level of CHOKeW and RIO. Here, we show three sets of results from RIO, denoted by RIO_1, RIO_2, and RIO_3, respectively. For RIO_1, we set $minth\_in = 150$ packets and $maxth\_in = 250$ packets, for RIO_2, $minth\_in = 130$ packets and $maxth\_in = 230$ packets, and for RIO_3, $minth\_in = 110$ packets and $maxth\_in = 210$ packets.

In Fig. 4, we see that the RCF of goodput zero for "out" traffic of RIO_1 is 0.1. In other words, 10 of the 100 "out" flows are starved. Similarly, for RIO_2 and RIO_3, 15 and 6 flows are starved, respectively. It is observed that some "in" flows of RIO also have very low goodput (e.g., the lowest goodput of "in" flows of RIO_2 is only 0.00015 Mbps) due to a lack of TCP protection. Flow starvation is very common in RIO, but it rarely happens in CHOKeW.

Now, we investigate the relationship between the number of TCP flows and the aggregate TCP goodput for each priority level. The results are shown in Fig. 5, where the curves of $w_{(1)} = 1$ and $w_{(2)} = 2$ correspond to the two priority levels. Half of the flows are assigned $w_{(1)}$ and the other half assigned $w_{(2)}$. As the number of flows changes, the results shows a little fluctuation owing to the stochastic characteristics of the network, but high-priority flows can get higher goodput no matter how many flows exist.

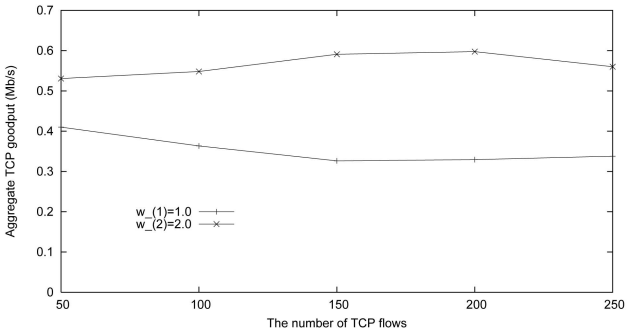


Fig. 5. The aggregate TCP goodput versus the number of TCP flows under a scenario of two priority levels.

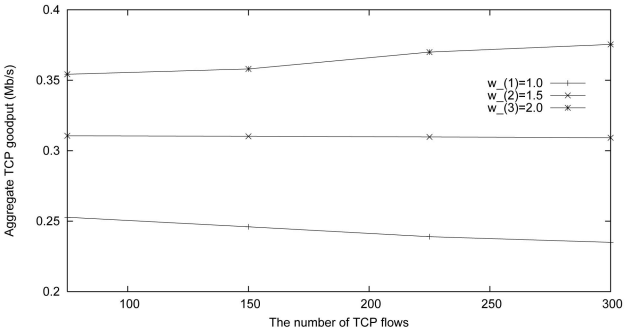


Fig. 7. The aggregate goodput versus the number of TCP flows under a scenario of three priority levels.

4.2 Two Priority Levels with Different Number of Flows

When the number of flows in each priority level is different, CHOKeW is still capable of differentiating bandwidth allocation on flow basis. In the following experiment, among the total 100 TCP flows, 25 flows are assigned fixed priority weight $w_{(1)} = 1.0$, and 75 flows are assigned $w_{(2)}$. As $w_{(2)}$ varies from 1.5 to 4.0, the average per-flow goodput is collected in each priority level and shown in Fig. 6. The results are compared with those of WFQ working in an aggregate flow mode, i.e., in order to circumvent the per-flow complexity, flows at the same priority level are merged into an aggregate flow before entering WFQ, and WFQ buffers packets in the same queue if they have the same priority, instead of using strict per-flow queueing. In WFQ, the buffer pool of 500 packets is split into two queues: the queue for $w_{(1)}$ has a capacity of 125 packets, and the queue for $w_{(2)}$ has a capacity of 375 packets.

In Fig. 6, it is readily to see that for both CHOKeW and WFQ, the goodput of high-priority flows rises as the weight ratio (i.e., $w_{(2)}/w_{(1)}$) increases, and accordingly, the goodput of low-priority flows falls. However, when the weight ratio is less than 3, the average per-flow goodput of high-priority flows is even lower than that of low-priority flows for WFQ. We say that WFQ does not guarantee to offer higher per-flow goodput to higher priority if the priority is taken by more flows, when aggregate flows are used. For CHOKeW, bandwidth differentiation works effectively in the whole range of $w_{(2)}$, even though all packets are mixed in one single queue.

This feature is developed based on the fact that CHOKeW does not require multiple queues to isolate flows; by contrast, conventional packet approximation of GPS, such as WFQ, cannot avoid the complexity caused by per-flow nature and give satisfactory bandwidth differentiation on a flow basis at the same time.

4.3 Three or More Priority Levels

In situations where multiple priority levels are used, the results are similar to those of two priority levels, i.e., the flows with higher priority achieve higher goodput. Since RIO supports only two priority levels, the results are not compared with those of RIO in this section. Figs. 7 and 8 demonstrate the aggregate TCP goodput for each priority level versus the number of TCP flows for three priority levels and for four priority levels, respectively. At each level, the number of TCP flows ranges from 25 to 100. In Fig. 7, three priority levels are configured using $w_{(1)} = 1.0$, $w_{(2)} = 1.5$, and $w_{(3)} = 2.0$. One more weight, $w_{(4)} = 2.5$, is added to the simulations corresponding to Fig. 8 for the fourth priority level. Even though the goodput fluctuates when the number of TCP flows changes, the flows in higher priority are still able to obtain higher goodput. Furthermore, no flow starvation is observed.

4.4 TCP Protection

TCP protection is another task of CHOKeW. We use UDP flows at the sending rate of 10 Mbps to simulate misbehaving flows. A total of 100 TCP flows are generated in the simulations. Priority weights $w_{(1)} = 1$ and $w_{(2)} = 2$ are assigned to equal number of flows. In order to evaluate

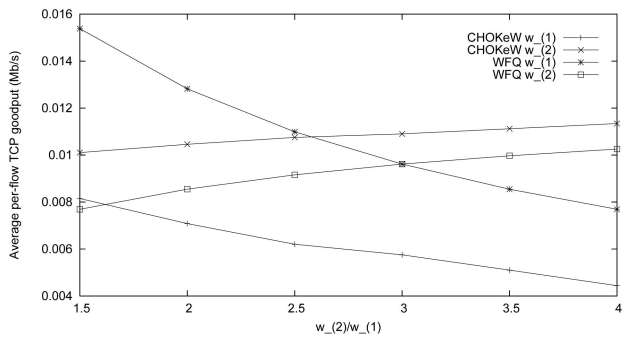


Fig. 6. The average per-flow TCP goodput versus $w_{(2)}/w_{(1)}$ when 25 flows are assigned $w_{(1)} = 1$ and 75 flows $w_{(2)}$.

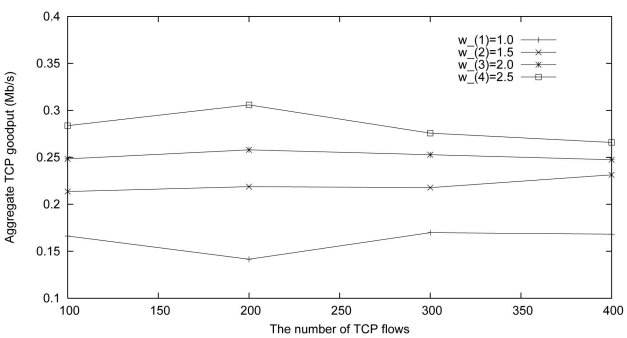


Fig. 8. The aggregate goodput versus the number of TCP flows under a scenario of four priority levels.

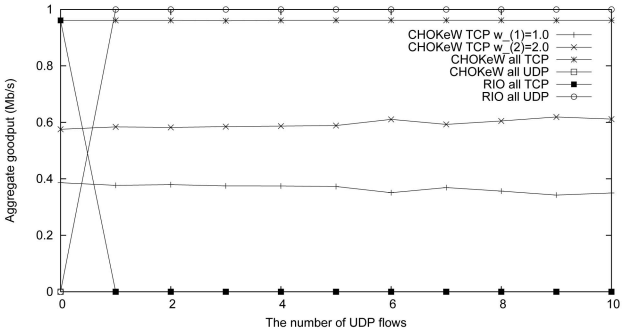


Fig. 9. The aggregate goodput versus the number of UDP flows under a scenario to investigate TCP protection.

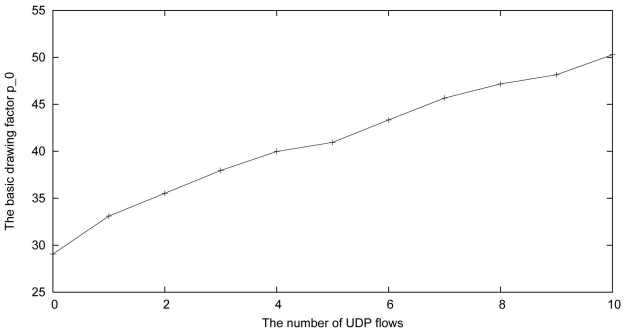


Fig. 10. The basic drawing factor p_0 versus the number of UDP flows under a scenario to investigate TCP protection.

the performance of TCP protection, the UDP flows are assigned the high-priority weight $w_{(2)} = 2$. As discussed before, if TCP protection functions properly in situations where misbehaving flows have higher priority, it should also work well when the misbehaving flows have lower priority. Hence, the effectiveness of TCP protection is validated as long as the high-priority misbehaving flows are blocked successfully.

The goodput versus the number of UDP flows is shown in Fig. 9, where CHOKeW is compared with RIO. Since no retransmission is provided by UDP flows, goodput is equal to throughput for UDP. For CHOKeW, even if the number of UDP flows increases from 1 to 10, the TCP goodput in each priority level (and hence, the aggregate goodput of all TCP flows) is quite stable. In other words, the link bandwidth is shared by these TCP flows, and the high-speed UDP flows are completely blocked by CHOKeW. By contrast, the bandwidth share for TCP flows in a RIO router is nearly zero, as high-speed UDP flows occupy almost all the bandwidth.

Fig. 10 illustrates the relationship between p_0 of CHOKeW and the number of UDP flows. As more UDP flows start, p_0 increases, but it rarely reaches the high value of starting to block TCP flows, while those high-speed UDP flows are blocked. In this experiment, we also find that few packets of TCP flows are dropped due to buffer overflow. In fact, when edge routers cooperate with core routers, the high-speed misbehaving flows will be marked with lower priority at the edge routers. Therefore, CHOKeW should be able to block misbehaving flows more easily, and p_0 should also be smaller than shown in Fig. 10.

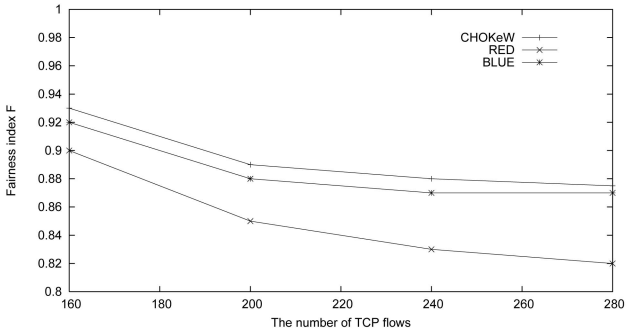


Fig. 11. The fairness index versus the number of flows for CHOKeW, RED and BLUE.

4.5 Fairness

In Section 3.3, we use the analytical model to explain how CHOKeW can provide better fairness among the flows in the same priority than conventional stateless AQM schemes such as RED and BLUE. We validate this attribute by simulations in this section. Since RED and BLUE do not support multiple priority levels and are only used in best-effort networks, we let CHOKeW work in one priority state (i.e., $w_{(1)} = 1$ for all flows) in this section.

In the simulation network illustrated in Fig. 3, the end-to-end propagation delay of a flow is set to one of 6, 60, 100, or 150 ms. Each of the four values is assigned to 25 percent of the total number of flows.[9]

When there are only a few (e.g., no more than three) flows under consideration, the fairness can be evaluated by directly observing the closeness of the goodput or throughput of different flows. In situations where many flows are active, however, it is hard to measure the fairness by direct observation; in this case, we use the fairness index:[10]

$$F = \frac{\left(\sum_{i=1}^{N} g_i\right)^2}{N \sum_{i=1}^{N} {g_i}^2}, \tag{27}$$

where g_i $(i = 1, 2, \cdots, N)$ represents the goodput of flow i. From (27), we know $F \in (0, 1]$. The closer the value of F is to 1, the better the fairness is. In this paper, we use g_i as goodput instead of throughput so that the TCP performance evaluation can reflect the successful delivery rate more accurately. Fig. 11 shows the fairness index of CHOKeW, RED, and BLUE versus the number of TCP flows ranging from 160 to 280. Even though the fairness decreases as the number of flows increases for all schemes, CHOKeW still provides better fairness than RED and BLUE. We also see that BLUE has better fairness than RED. When a great number of TCP connections exist, BLUE is capable of maintaining a more stable queue length and preventing global synchronization more effectively than RED, given the same buffer limit [14]. Global synchronization is the phenomenon that all TCP flows increase and decrease the

9. For flow i, the end-to-end propagation delay is $4\tau_i + 2\tau_0$. Since τ_0 is constant for all flows in Fig. 3, the propagation delay can be assigned a desired value given an appropriate τ_i.

10. This fairness index was defined by Jain [34] using throughput. Since then, it has been widely used for research on TCP/IP congestion control.

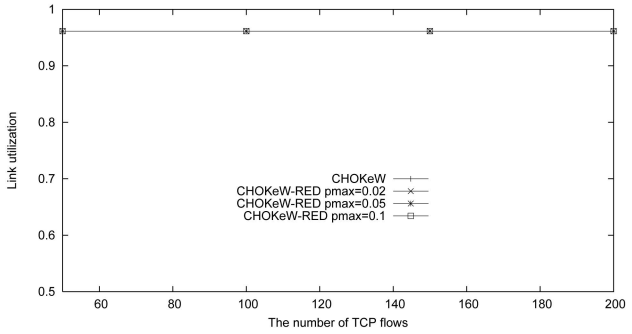


Fig. 12. Link utilization of CHOKeW and CHOKeW-RED.

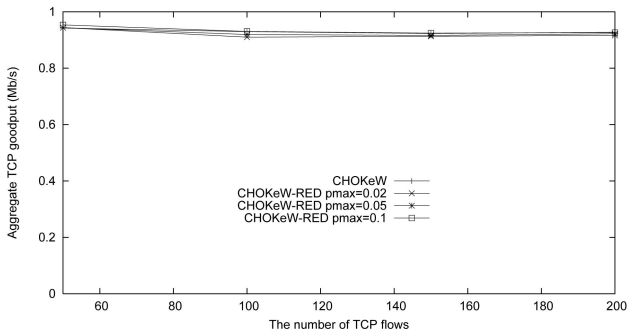


Fig. 14. Aggregate TCP goodput of CHOKeW and CHOKeW-RED.

sending rates simultaneously [13], and it can invalidate the effect of random dropping and hence harm the fairness significantly.

4.6 CHOKeW versus CHOKeW-RED

An adaptive drawing algorithm has been incorporated into the design of CHOKeW, where TCP flows can get network congestion notifications from matched drops. Simultaneously, the bandwidth share of high-speed unresponsive flows are also brought under control by matched drops. As a result, the RED module is no longer required in CHOKeW. We compared the average queue length, link utilization, and TCP goodput of CHOKeW with those of CHOKeW-RED (i.e., CHOKeW working with the RED module) with $pmax$ ranging from 0.02 to 0.1.

The relationship between the number of TCP flows and the values of link utilization, the average queue length, and the aggregate TCP goodput is shown in Figs. 12, 13, and 14, respectively. In each figure, the performance results of CHOKeW-RED are indicated by three curves, each corresponding to one of the three values for $pmax$ (0.02, 0.05, and 0.1).

Fig. 12 shows that all schemes maintain an approximate link utilization of 96 percent (shown by the curves overlapping each other), which is considered sufficient for the Internet. In Fig. 13, we can see that the average queue length for CHOKeW-RED increases as the number of TCP flows increases. In contrast, the average queue length can be maintained at a steady value within the normal range between L^- (125 packets) and L^+ (175 packets) for CHOKeW. In situations where the number of TCP flows is larger than 100, CHOKeW has the shortest queue length. In CHOKeW-RED, if $L \leq L^+$ is maintained by random drops from RED (for example, this may happen when all flows use TCP), p_0 does not have an opportunity to increase its value (p_0 is initialized to 0 and updated to $p_0 + p^+$ only when $L > L^+$), which causes a longer queue in CHOKeW-RED.

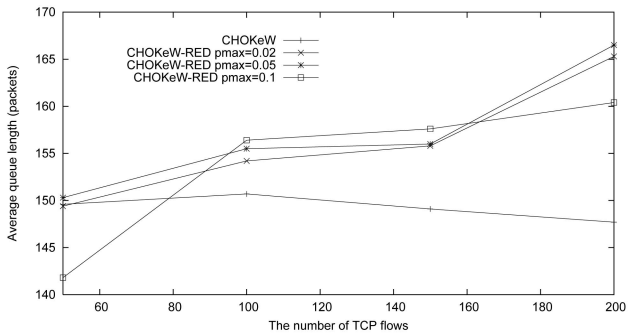


Fig. 13. Average queue length of CHOKeW and CHOKeW-RED.

Besides the link utilization and the average queue length, the aggregate TCP goodput is always of interest when we evaluate TCP performance. The comparison of TCP goodput between CHOKeW and CHOKeW-RED is shown in Fig. 14. In this figure, all of the schemes have similar results. In addition, when the number of TCP flows is larger than 100, CHOKeW rivals the best of CHOKeW-RED (i.e., $pmax = 0.1$).

In a special environment, if the network has not experienced heavy congestion and the queue length $L < L^+$ has been maintained by random drops of RED since the beginning, CHOKeW-RED cannot achieve the goal of bandwidth differentiation as $p_0 = 0$, and thus, $p_i = p_j = 0$ even if $w_i \neq w_j$. In other words, CHOKeW independent of RED works best.

5 Implementation Considerations

5.1 Buffer for Flow IDs

One of the implementation considerations is the buffer size. As discussed in [15], the objective of using buffers in the Internet is to absorb data bursts and transmit them during subsequent periods of silence. Maintaining normally small queues does not necessarily generate poor throughput if appropriate queue management is used; instead, it may help maintain good throughput, as well as lower end-to-end delay.

When used in CHOKeW, this strategy, however, may cause a problem that no two packets in the buffer are from the same flow, although this is an extreme case and is unlikely to happen so often due to the bursty nature of flows. In this case, no matter how large p_i is, packets drawn from the buffer will never match an arriving packet from flow i. In order to improve the effectiveness of matched drops, we consider a method that uses a FIFO buffer for storing the flow IDs of forwarded packets in the history. When the packets are forwarded to the downstream link, their flow IDs are also copied into the ID buffer. If the ID buffer is full, the oldest ID is deleted, and its space is reallocated to a new ID. Since the size of flow IDs is

```
IF L > L_th
THEN
  Generate a random number v' ∈ [0,1)
  IF v' < L_ID/(L_ID + L)
  THEN
    m ← 2 × m
    WHILE m > 0
      m ← m − 1
      Draw ξ_c from ID buffer at random
      IF ξ_a=ξ_c
      THEN
        L ← L − l_a, drop pkt
        RETURN /*wait for the next arrival*/
    GOTO Step (6) in Fig.1

Parameters:
ξ_a: Flow ID of arriving packet
ξ_c: Flow ID drawn from ID buffer at random
l_a: Size of the arriving packet
L_ID: Queue length of ID buffer
```

Fig. 15. The extended matched drop algorithm with ID buffer.

constant and much smaller than packet size, the implementation does not require additional processing time or large memory space. We generalize matched drops by drawing flow IDs from a "unified buffer," which includes the ID buffer and the packet buffer. This modification is illustrated in Fig. 15, interpreted as a step inserted between steps (4) and (5) in Fig. 1.

Let L_{ID} denote the number of IDs in the buffer when a new packet arrives. Draws can happen either in the regular packet buffer or in the ID buffer. The probabilities that the draws happen in the ID buffer and the packet buffer are $\frac{L_{ID}}{L_{ID}+L}$ and $\frac{L}{L_{ID}+L}$, respectively. If the draws are from the ID buffer, only one packet (i.e., the new arrival) is dropped each time, and hence, the maximum number of draws is set to $2 \times p_i$, implemented by $m \leftarrow 2 \times m$ in Fig. 15.

5.2 Parallelizing the Drawing Process

Another implementation consideration is how to shorten the time of the drawing process. When $p_0 > 1$, CHOKeW may draw more than one packets for comparison upon each arrival. In Section 2, we use a serial drawing process for the description (i.e., packets are drawn one at a time) to let the algorithm be easily understood. If this process does not meet the time requirement of the packet forwarding in the router, a parallel method can be introduced.

Let ξ_a be the flow ID of the arriving packet, ξ_b^i $(i = 1, 2, \cdots, m)$ be the flow IDs of the packets drawn from the buffer. The logical operation of matched drops can be represented by bitwise XOR $(\oplus)$ and Boolean AND $(\wedge)$ as follows:[11] If

$$\bigwedge_{i=1}^{m} (\xi_a \oplus \xi_b^i) = 0(false),$$

then conduct matched drops. Note that the above equation is satisfied if and only if some term of $\xi_a \oplus \xi_b^i$ is $false$. On one hand, any ξ_b^i drawn from the buffer can provoke matched drops if it is equal to ξ_a. On the other hand, matched drops are triggered only when some ξ_b^i is equal to ξ_a. Besides the arriving packet, we can simply drop any one of the buffered packets with flow ID ξ_b^i that makes $\xi_a \oplus \xi_b^i = 0$.

11. Here, bitwise operations treat bit 1 as $true$ and bit 0 as $false$, while Boolean operations treat all nonzero values as $true$ and value 0 as $false$.

6 Conclusions and Future Work

In this paper, we proposed a cost-effective AQM scheme called CHOKeW that provides bandwidth differentiation among flows at multiple priority levels. Both the analytical model and the simulations showed that CHOKeW is capable of providing higher bandwidth share to flows in higher priority, maintaining good fairness among flows in the same priority, and protecting TCP against high-speed unresponsive flows when network congestion occurs. The simulations also demonstrate that CHOKeW is able to achieve efficient link utilization with a shorter queue length than conventional AQM schemes.

Our analytical model was designed to provide insights into the behavior of CHOKeW and gave a qualitative explanation of its effectiveness. Further understanding of network dynamics affected by CHOKeW needs more comprehensive models in the future.

The parameter tuning is another area of exploration for future work on CHOKeW. As indicated in Fig. 6, when the priority-weight ratio $w_{(2)}/w_{(1)}$ is higher, the bandwidth share being allocated to the higher-priority flows will be greater. In the meantime, considering that the total available bandwidth does not change, the bandwidth share allocated to the lower priority flows will be smaller. The value of $w_{(2)}/w_{(1)}$ should be tailored to the needs of the applications, the network environments, and the users' demands. This research can also be incorporated with price-based DiffServ networks to provide differentiated bandwidth allocation, as well as TCP protection.

Acknowledgments

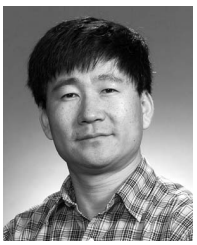
The authors are grateful to Masha S.H. Lam and Frank Goergen for the editorial assistance. This work was partially supported by the US National Science Foundation (NSF) under Grants ANI-0093241 (CAREER Award) and CNS-0721744. The work was also partially supported by the National Science Council (NSC), R.O.C., under the NSC Visiting Professorship with Contract NSC-96-2811-E-002-010 and Chunghwa Telecom under Contract NBY970147.

References

[1] P. Almquist, *Type of Service in the Internet Protocol Suite,* IETF RFC 1349, July 1992.
[2] *DSSI Core Aspects of Frame Rely,* ANSI T1S1, Mar. 1990.
[3] "ATM Traffic Management Specification Version 4.0" *ATM Forum,* Apr. 1996.
[4] R. Braden, D. Clark, and S. Shenker, "Integrated Services in the Internet Architecture: An Overview," *IETF RFC 1633,* July 1994.
[5] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource Reservation Protocol (RSVP): Version 1 Functional Specification," *IETF RFC 2205,* Sept. 1997.

[6] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An Architecture for Differentiated Service," *IETF RFC 2475,* Dec. 1998.
[7] E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol Label Switching Architecture," *IETF RFC 3031,* Jan. 2001.
[8] I. Stoica, S. Shenker, and H. Zhang, "Core-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocation in High Speed Networks," *Proc. ACM SIGCOMM,* 1998.
[9] N. Nichols, S. Blake, F. Baker, and D. Black, *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers,* IETF RFC 2474, Dec. 1998.
[10] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski, "Assured Forwarding PHB Group," *IETF RFC 2597,* June 1999.
[11] P. Marbach, "Pricing Differentiated Services Networks: Bursty Traffic," *Proc. IEEE INFOCOM,* 2001.
[12] S. Floyd and K. Fall, "Promoting the Use of End-to-End Congestion Control in the Internet," *IEEE/ACM Trans. Networking,* vol. 7, no. 4, pp. 458-472, Aug. 1999.
[13] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Trans. Networking,* vol. 1, no. 4, pp. 397-413, Aug. 1993.
[14] W. Feng, K. Shin, D. Kandlur, and D. Saha, "The BLUE Active Queue Management Algorithm," *IEEE/ACM Trans. Networking,* vol. 10, no. 4, pp. 513-528, Aug. 2002.
[15] R. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, and D. Estrin, *Recommendations on Queue Management and Congestion Avoidance in the Internet,* IETF RFC 2309, Apr. 1998.
[16] K. Cho, "Flow-Valve: Embedding a Safety-Valve in RED," *Proc. IEEE Global Telecomm. Conf. (GLOBECOM '99),* Dec. 1999.
[17] R. Mahajan and S. Floyd, "Controlling High-Bandwidth Flows at the Congested Router," ICSI Technical Report TR-01-001, http://www.icir.org/red-pd/, Apr. 2001.
[18] R. Pan, B. Prabhakar, and K. Psounis, "CHOKe: A Stateless Active Queue Management Scheme for Approximating Fair Bandwidth Allocation," *Proc. IEEE INFOCOM,* 2001.
[19] A. Tang, J. Wang, and S. Low, "Understanding CHOKe," *Proc. IEEE INFOCOM,* 2003.
[20] D. Clark and W. Fang, "Explicit Allocation of Best Effort Packet Delivery Service," *IEEE/ACM Trans. Networking,* vol. 6, no. 4, pp. 362-373, Aug. 1998.
[21] A. Demers, S. Keshav, and S. Shenker, "Analysis and Simulations of a Fair Queueing Algorithm," *Proc. ACM SIGCOMM,* 1989.
[22] A. Parekh and R. Gallager, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case," *IEEE/ACM Trans. Networking,* vol. 1, no. 3, pp. 344-357, June 1993.
[23] U. Bodin, O. Schelen, and S. Pink, "Load-Tolerant Differentiation with Active Queue Management," *ACM Computer Comm. Rev.,* http://www.acm.org/sigcomm/ccr/archive/ccr-toc/, 2000.
[24] S. Ramabhadran and J. Pasquale, "Stratified Round Robin: A Low Complexity Packet Scheduler with Bandwidth Fairness and Bounded Delay," *Proc. ACM SIGCOMM '03,* Aug. 2003.
[25] J. Bennet and H. Zhang, "WF^2Q: Worst Case Fair Weighted Fair Queuing," *Proc. IEEE INFOCOM '96,* Mar. 1996.
[26] S. Suri, G. Varghese, and G. Chandramenon, "Leap Forward Virtual Clock: A New Fair Queueing Scheme with Guaranteed Delay and Throughput Fairness," *Proc. IEEE INFOCOM '97,* Apr. 1997.
[27] M. Shreedhar and G. Varghese, "Efficient Fair Queuing Using Deficit Round Robin," *IEEE/ACM Trans. Networking,* vol. 4, no. 3, pp. 375-385, June 1996.
[28] D. Clark, S. Shenker, and L. Zhang, "Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism," *Proc. ACM SIGCOMM,* 1992.
[29] R. Cooper, *Introduction to Queueing Theory,* second ed. Elsevier North, 1981.
[30] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP Throughput: A Simple Model and Its Empirical Validation," *Proc. ACM SIGCOMM,* 1998.
[31] *ns-2 (Network Simulator Version 2),* http://www.isi.edu/nsnam/ns/, 2008.
[32] M. Allman, V. Paxson, and W. Stevens, "TCP Congestion Control," *IETF RFC 2581,* Apr. 1999.
[33] W. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols.* Addison-Wesley, 1994.
[34] R. Jain, *The Art of Computer Systems Performance Analysis.* John Wiley & Sons, 1991.
[35] X. Chen, H. Zhai, J. Wang, and Y. Fang, "TCP Performance over Mobile Ad Hoc Networks," *Canadian J. Electrical and Computer Eng.,* vol. 29, no. 1/2, pp. 129-134, Jan.-Apr. 2004.
[36] S. Wen, Y. Fang, and H. Sun, "Differentiated Bandwidth Allocation and TCP Protection within Core Routers," *Proc. IEEE Military Comm. Conf. (Milcom),* 2005.
[37] H. Zhai and Y. Fang, "Distributed Flow Control and Medium Access in Multihop Ad Hoc Networks," *IEEE Trans. Mobile Computing,* vol. 5, no. 11, pp. 1503-1514, Nov. 2006.
[38] H. Zhai, X. Chen, and Y. Fang, "Improving Transport Layer Performance in Multihop Ad Hoc Networks by Exploiting MAC Layer Information," *IEEE Trans. Wireless Comm.,* vol. 6, no. 5, pp. 1692-1701, May 2007.

Shushan Wen received the BS degree and the PhD degree in electronic engineering from the University of Electronic Science and Technology of China (UESTC) in 1996 and September 2002, respectively, and the PhD degree in electrical and computer engineering from the University of Florida in December 2006. He is with the Department of Electrical and Computer Engineering, University of Florida, Gainesville. His research interests include traffic engineering, congestion control, TCP protection, fairness and QoS in wireless, and wired communication networks. He is a student member of the IEEE.



Yuguang (Michael) Fang received the PhD degree in systems engineering from Case Western Reserve University in January 1994 and the PhD degree in electrical engineering from Boston University in May 1997. He was an assistant professor in the Department of Electrical and Computer Engineering, New Jersey Institute of Technology from July 1998 to May 2000. He then joined the Department of Electrical and Computer Engineering, University of Florida, in May 2000 as an assistant professor, got an early promotion to an associate professor with tenure in August 2003 and to a full professor in August 2005. He holds a University of Florida Research Foundation (UFRF) Professorship from 2006 to 2009. He has published more than 200 papers in refereed professional journals and conference proceedings. He received the US National Science Foundation (NSF) Faculty Early Career Award in 2001 and the Office of Naval Research Young Investigator Award in 2002. He is the recipient of the Best Paper Award in IEEE International Conference on Network Protocols (ICNP) in 2006 and the IEEE TCGN Best Paper Award in the IEEE High-Speed Networks Symposium, IEEE Globecom in 2002. He is also active in professional activities. He is a fellow of the IEEE and a member of ACM. He has served on several editorial boards of technical journals including the *IEEE Transactions on Communications*, *IEEE Transactions on Wireless Communications*, *IEEE Wireless Communications Magazine*, and *ACM Wireless Networks*. He was an editor for *IEEE Transactions on Mobile Computing* and currently serves on its Steering Committee. He has been actively participating in professional conference organizations such as serving as the Steering Committee Co-Chair for QShine, the Technical Program Vice Chair for the IEEE INFOCOM 2005, Technical Program Symposium Cochair for IEEE Globecom 2004, and a member of Technical Program Committee for IEEE INFOCOM (1998, 2000, and 2003-2009).

Hairong Sun received the BS and MS degrees in electronic engineering from the Shanghai Jiaotong University, China, in 1988 and 1991, respectively, and the PhD degree in electronic engineering from the University of Electronic Science and Technology of China (UESTC) in 1993. He is a staff engineer in the System Group, Sun Microsystems. He is responsible for reliability, availability, and performability assessment of storage products. He worked as an associate professor with UESTC in 1994 and a professor in 1997. He is a senior member of the IEEE.