CHOKeW: Bandwidth Differentiation and TCP Protection in Core Networks

Shushan Wen, Yuguang Fang
Department of Electrical and
Computer Engineering
University of Florida
Gainesville, Florida 32611, USA
Email: wen@ecel.ufl.edu, fang@ece.ufl.edu

and

Hairong Sun
Sun Microsystems
Broomfield, Colorado 80021, USA
Email: hairong.sun@sun.com

ABSTRACT

Both bandwidth differentiation and TCP protection is important for implementing Quality of Services (QoS) in TCP/IP networks. To the best of our knowledge, no other schemes have combined these two tasks together so far. In this paper we present a stateless Active Queue Management (AQM) algorithm called CHOKeW. CHOKeW uses "matched drops" created by CHOKe to control the bandwidth allocation, but excludes the RED module. Based on the congestion status and the priority of the arriving packet, CHOKeW adjusts the maximum number of packets drawn from the buffer for matched drops. Using simulations, we show that CHOKeW is capable of working in different congestion scenarios, supporting multiple bandwidth priority levels by giving high priority flows with high throughput, and restrict the throughput of high-speed unresponsive flows to protect TCP flows.

I. Introduction

Nowadays, one of the technical problems related to Quality of Services (QoS) in the Internet that need to be solved is how to provide bandwidth differentiation cost-effectively in core routers.

In the literature, bandwidth differentiation has been investigated by some researchers. A typical scheme is RED with In/Out bit (RIO) [8], which 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 that causes a smaller dropping probability. In RIO an "out" flow may be starved because there is no mechanism to guarantee the bandwidth share for low-priority traffic [3], which is a disadvantage of RIO.

In general, routers in the DiffServ architecture, similar to those proposed in Core-Stateless Fair Queueing (CSFQ) [23], 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. As [15] proposed, a set of priority services can be applied to modelling and analyzing DiffServ, by mapping the PHBs that receive better services into the higher priority levels.

When studying the method to provide bandwidth differentiation to Transmission Control Protocol (TCP), the primary transport layer protocol in the Internet, we must consider the mechanisms to protect TCP at the same time.

The importance of TCP protection has been discussed by Floyd and Fall [12]. They predicted that the Internet will collapse if there is no mechanism to protect TCP flows. In the worst case, the routers are consumed with forwarding packets even though no packet is useful for receivers.

Conventional Active Queue Management (AQM) algorithms such as Random Early Detection (RED) [10] cannot protect TCP flows. It is strongly suggested that novel AQM schemes be designed for TCP protection with in routers [4], [12]. Cho [5] proposed a mechanism which uses a "flow-valve" filter for RED to punish non-TCP-friendly flows. However, this approach requires three parameters to be reserved for each flow, which increases the memory requirement significantly. In [14], 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, inspired by the understanding that 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.

Compared with previous methods, CHOKe [19], proposed by Pan *et al.*, does not require per-flow state maintenance, which makes the implementation in core networks much simpler. 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" in short). If the two packets are not from the same flow, the arriving packet is delivered to RED and the buffered packet is kept intact.

CHOKe is simple and effective for TCP protection, but it only supports best-effort traffic. In DiffServ networks where flows have different priority, TCP protection is still an imperative task. In this paper, we use the technique of matched drops to design another scheme called CHOKeW. The letter

W represents a function that has the capability of supporting multiple weights for bandwidth differentiation.

Both TCP protection and bandwidth differentiation is quite important for providing QoS to TCP traffic in the Internet. This is the goal of our scheme. To our best of knowledge, no other schemes have combined TCP protection with bandwidth differentiation thus far.

Our scheme use matched drops to control bandwidth shares. When a low-priority TCP flow only has a small bandwidth share, the backlog in the buffer for this flow is also small, and the packets from this flow are unlikely dropped. So this TCP flow is unlikely to be starved. To avoid flow starvation successfully is one of the advantages of our scheme.

In fact, some scheduling schemes, such as Weighted Fair Queueing (WFQ) [9] and other packet approximation of the Generalized Processor Sharing (GPS) model [20], [21] may also support differentiated bandwidth allocation. However, the main disadvantage of these schemes is that they require constant per-flow state maintenance, which is not cost-effective in core networks as it will cause memory-requirement complexity O(N) and per-packet-processing complexity usually larger than O(1), where N denotes the number of flows being served by the router.[1] Our scheme is a stateless scheme, 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). The simplicity is another advantage of our scheme.

The rest of the paper is organized as follows. Section II describes our algorithm, CHOKeW. Section III presents and discusses the simulation results, including the effect of supporting two priority levels, multiple priority levels, TCP protection, and fairness comparison. We conclude our paper in Section IV.

II. CHOKeW Algorithm

CHOKeW uses the strategy of matched drops presented by CHOKe [19] to protect TCP flows. Like CHOKe, CHOKeW is a stateless algorithm that 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 heavier 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 the authors of [19] 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. Simulations were used to compare the performance of CHOKeW and that of CHOKeW-RED (i.e., CHOKeW with RED) and the results are similar. We do not show the simulation results in this paper due to the space limit.

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, to control the time as well as the maximum number of draws. For flow i ($i = 1, 2, \cdots, N$, where N is the number of active flows), the drawing factor is denoted by p_i ($p_i \geq 0$).

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

Let w_i ($w_i \geq 1$) denote the priority weight of flow i. If two flows, say, i and j, are given the same priority, then $w_i = w_j$.

Let p_0 denote the drawing factor for traffic with the lowest priority. The relationship among w_i, p_i, and p_0 can be described as

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

If flow i has a higher priority weight than flow j ($w_i > w_j$), flow i will have a smaller drawing factor than flow j ($p_i < p_j$). In other words, the maximum number of random draws upon an arrival from flow i is smaller, and hence flow i will have a lower possibility of becoming the victim of matched drops. This is the basic mechanism for supporting bandwidth differentiation in CHOKeW. In addition, from (1) we have $p_i = p_j$ when flow i and flow j are of the same priority; this is how CHOKeW provides better fairness among flows with the same priority).

Now we discuss the precise meaning of drawing factor p_i, which depends upon its value. According to the value of p_i ($p_i > 0$), the drawing process 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

$$p_i = m + f. \tag{2}$$

where $m \in \mathbb{Z}^*$ (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 represents the fractional part of p_i. In this case, at the most m or $m+1$ packets in the buffer may be drawn for comparison. Let d_{max} denote the maximum number of random draws. We have

$$\begin{cases} Prob[d_{max} = m+1] = f, \\ Prob[d_{max} = m] = 1 - f. \end{cases}$$

[1] For example, according to [22], the per-packet-processing complexity is O(N) for WFQ, O($\log N$) for WF^{2}Q [1], and O($\log \log N$) for Leap Forward Virtual Clock [24].

```
For each packet (pkt) arrival
(1) Update p_0
(2) Check the priority level of pkt
    IF it corresponds to weight w_i
    THEN p_i ← p_0/w_i, m ← ⌊p_i⌋, f = p_i − m
(3) Generate a random number v ∈ [0, 1)
    IF v < f
    THEN m ← m + 1
(4) IF L > L_th
    THEN
      WHILE m > 0
        m ← m − 1
        Draw a packet (pkt') at random
        IF pkt' and pkt are from the same flow
        THEN
          Drop both pkt' and pkt
          Return
        ELSE keep pkt' intact
    IF buffer is full
    THEN drop pkt
    ELSE let pkt enter the buffer

Parameters:
L: Queue length
L_th: Queue length threshold to activate drawing
```

Fig. 1. The algorithm of drawing packets

The algorithm of drawing packets is described in Fig.1. In this figure, we only use one variable m to record the value of m (before Step (3)) and d_{max} (after Step (3)). Note that no more packets need to be drawn as soon as matched drops occur. On the other hand, if no match is found, the drawing process will continue until the number of draws reaches d_{max}.

CHOKeW is a stateless algorithm since it does not need any variables to store per-flow states. In the DiffServ network architecture, the routers are categorized into edge routers and core routers. The edge routers are designed to support fewer flows and are able to perform more complicated functions for better service. The core routers are designed to support a vast number of flows, but only perform simpler functions for cost effectiveness. Edge routers are able to maintain some per-flow states. For instance, they may continually measure the arrival rate for each flow. If the priority of a flow is determined by end users but its arrival rate is above a threshold defined by the network, edge routers may lower the priority of this flow by remarking the DS fields of its packets. If the priority is assigned by a negotiation mechanism between the network and end users, the values in DS fields may be initialized by edge routers. In any case, the packets in excess of the rate threshold are (re)marked with lower priority and are prone to being dropped by core routers.

The congestion status of a router may become either heavier or lighter after a period of time, since circumstances (such as the number of users, the application types, and the traffic priority) constantly change. When the router is more congested, CHOKeW informs TCP senders to decrease their sending rates by dropping more packets. Otherwise, CHOKeW drops fewer packets. This adaptive function is implemented by updating

```
Initialization:
p_0 ← 0

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 to increase p_0
L^−: Queue length threshold to decrease p_0
L_th < L^− < L^+
p_0: Basic drawing factor with initial value 0
p^+: Step length required for increasing p_0
p^−: Step length required for decreasing p_0
```

Fig. 2. The algorithm of updating p_0

TABLE I
THE STATE OF CHOKEW VS. 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	$p_0 + p^+$

p_0. The updating process is shown in Fig.2, which details Step (1) of the algorithm of drawing packets that is shown in Fig.1. The combination of Fig.1 and Fig.2 provides a complete description of the CHOKeW algorithm.

The present 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 I. At any time, CHOKeW works in one of following states:

1) inactive matched drops and decreasing p_0 (unless $p_0 = 0$), when $0 \leq L \leq L_{th}$;
2) active matched drops and decreasing p_0 (unless $p_0 = 0$), when $L_{th} < L < L^-$;
3) active matched drops and constant p_0, when $L^- \leq L \leq L^+$;
4) active matched drops and increasing p_0, when $L^+ < L \leq L_{lim}$.

The implementation of CHOKeW is feasible in core routers, because CHOKeW only needs to check the DS field, without aid of the flow ID,[2] to find the priority upon each packet arrival. Therefore, when CHOKeW is used in core routers, priority becomes a packet feature. In terms of service qualities in the core network, packets from different flows shall be served the same if they have the same priority; on the other hand, packets from the same flow may be treated differently

[2]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.

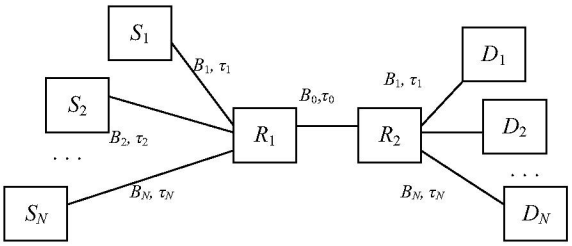

Fig. 3. Network Topology

if their priority is different (e.g., some packets are remarked by the routers).

For ease of explanation, let $w_{(k)}$ denote the priority weight corresponding to priority k ($k = 1, 2, \cdots, M$, where M is the number of predefined priority levels). If a packet from flow i is marked with priority k, for this packet $w_i = w_{(k)}$. When remarking is allowed, another packet from flow i may be remarked with priority k' ($k' \neq k$), for this packet $w_i = w_{(k')}$. If a packet from flow j ($j \neq i$) is also marked with priority k, for this packet, also, $w_j = w_{(k)}$.

Now we discuss the complexity of CHOKeW. Based on the above description, we know that CHOKeW needs to remember only $w_{(k)}$ for each predefined priority level k ($k = 1, 2, \cdots, M$), instead of w_i 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 not be a large value even in the foreseeable future, i.e., $M \ll N$. Since M is independent of N, we know that with respect to N, 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).

III. Performance Evaluation

To evaluate CHOKeW in different scenarios and to compare it with some other schemes, we implemented CHOKeW using `ns` simulator version 2 [17]. The CHOKeW code is designed as a patch for `ns`, and it is available at [26].

In this section, we use the network topology 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 1000 bytes. TCP flows are driven by FTP applications, and UDP flows are driven by CBR traffic. All TCPs are simulated as 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: $minth = 100$ packets, $maxth = 200$ packets, $gentle = true$, $pmax = 0.02$, and the EWMA weight is set to 0.002.

Parameters of RIO include those for "out" traffic and those for "in" traffic. For "out" traffic, $minth\_out = 100$ packets, $maxth\_out = 200$ packets, $pmax\_out = 0.02$. For "in" traffic, $minth\_in = 110$ packets, $maxth\_in = 210$ packets, $pmax\_in = 0.01$ (except in Subsection III-A where different parameters are specified to compare RIO with CHOKeW). Both $gentle\_out$ and $gentle\_in$ are set to $true$.

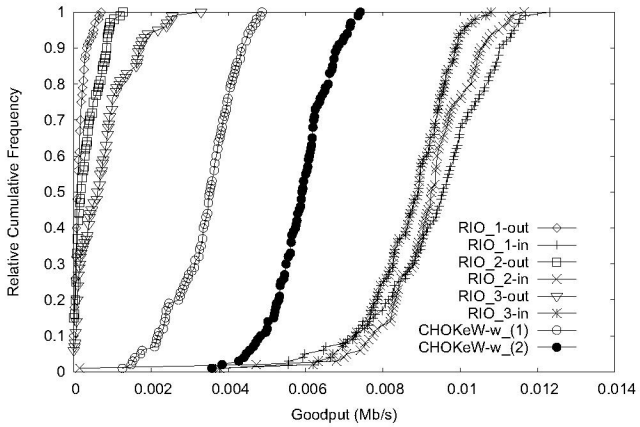

Fig. 4. The Relative Cumulative Frequency of RIO and CHOKeW

Parameters of BLUE are set as follows: $\delta_1 = 0.0025$ (the step length for increasing the dropping probability), $\delta_2 = 0.00025$ (the step length for decreasing the dropping probability), and $freeze\_time = 100$ ms.

A. Two Priority Levels

One of the main tasks of CHOKeW is supporting bandwidth differentiation for multiple priority levels when working in a stateless method. We validate the effect of supporting two priority levels in this subsection and three or more priority levels in the next subsection.

As mentioned before, flow starvation often happens in RIO but is avoidable in CHOKeW. In order to investigate seriousness of the flow starvation, we record the Relative Cumulative Frequency (RCF) of goodput for flows in the same priority. The RCF of goodput g for flows in a certain priority 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. For CHOKeW, 100 flows are of low priority with $w_{(1)} = 1$, and the other 100 flows are of high priority with $w_{(2)} = 2$. For RIO, 100 flows belong to "out" traffic and the other 100 flows belong to "in" traffic. The RCF of goodput for flows in each priority level of CHOKeW and RIO is shown in Fig.4. 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; for RIO_3 $minth\_in = 110$ packets and $maxth\_in = 210$ packets.

From Fig.4 we see that for "out" traffic of RIO_1 the RCF of goodput zero 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. Even some "in" flows of RIO

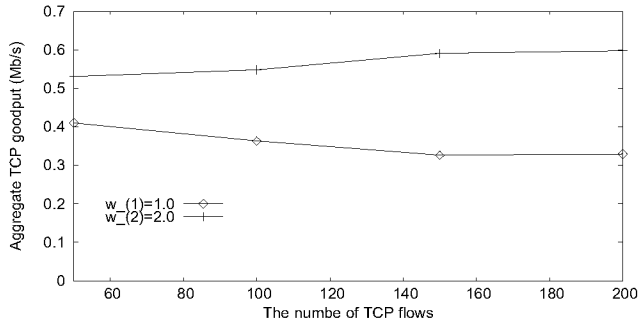

Fig. 5. The aggregate TCP goodput for two priority levels

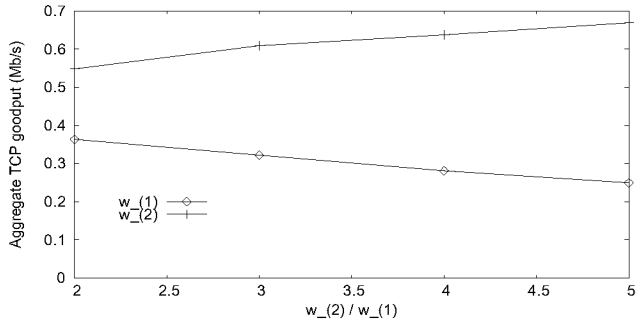

Fig. 6. The relationship between aggregate TCP goodput and $w_{(2)}/w_{(1)}$

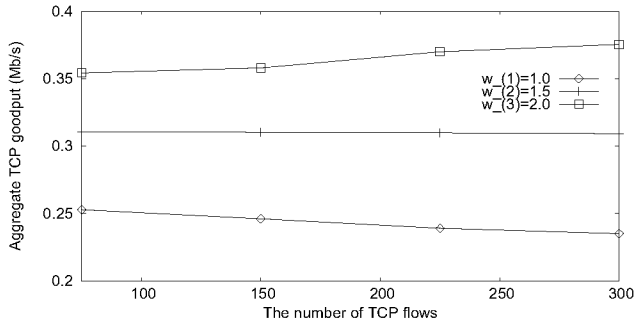

Fig. 7. The relationship between the aggregate goodput and the number of TCP flows for three priority levels

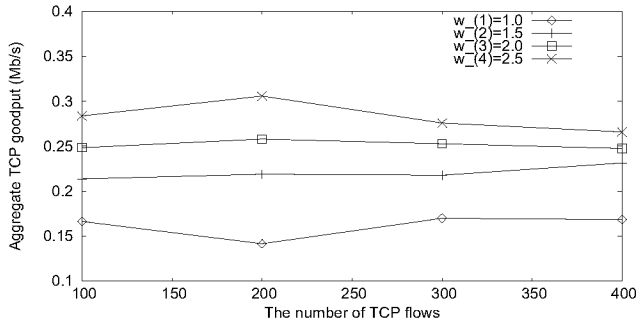

Fig. 8. The relationship between aggregate goodput and the number of TCP flows for four priority levels

may have very low goodput (e.g., the lowest goodput of "in" flows of RIO_2 is only 0.00015 Mb/s). Flow starvation is very common in RIO, but it is rarely observed 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. Here two priority levels are corresponding to $w_{(1)} = 1$ and $w_{(2)} = 2$. Half of the flows are assigned $w_{(1)}$ and the other half assigned $w_{(2)}$. As more flows are going through the CHOKeW router, the goodput difference between the higher-priority flows and the lower-priority flows changes, but high-priority flows can get higher goodput no matter how many flows exist.

The aggregate TCP goodput for each priority level versus the ratio of the higher priority weight to the lower priority weight (i.e., $w_{(2)}/w_{(1)}$) is shown in Fig.6, where 100 TCP flows is simulated and $w_{(1)}$ is set to 1. We can see that as $w_{(2)}/w_{(1)}$ ascends, the goodput difference between the flows at the higher priority level and the flows at the lower priority level is more significant. When the number of flows at the higher priority level equals that at the lower priority level, considering CHOKeW maintains good fairness among flows with the same priority, the per-flow goodput for a higher-priority flow is greater than that for a lower-priority flow.[3]

[3]Here we only show the results of the scenario in which two priority levels have the same number of flows. The results about per-flow goodput still hold true in an environment where the number of TCP flows at two priority levels is different, although the aggregate goodput of the higher-priority flows may be smaller than that of the lower-priority flows if there are much fewer flows in the higher priority.

B. 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 only supports two priority levels, the results are not compared with RIO in this subsection. Fig.7 and Fig.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. Three priority levels are configured using $w_{(1)} = 1.0$, $w_{(2)} = 1.5$, and $w_{(3)} = 2.0$. Four priority levels are configured using $w_{(1)} = 1.0$, $w_{(2)} = 1.5$, $w_{(3)} = 2.0$, and $w_{(4)} = 2.5$. Although the goodput fluctuates when the number of TCP flows changes, the flows in higher priority are still able to obtain higher goodput. In the simulations of multiple priorities, no flow starvation is observed.

C. TCP Protection

TCP protection is another task of CHOKeW. We use UDP flows at the sending rate of 10 Mb/s to simulate misbehaving flows. 100 TCP flows (50 with the $w_{(1)} = 1$ and 50 with $w_{(2)} = 2$) are simulated. In order to evaluate the performance of TCP protection, the UDP flows are assigned the high priority weight $w_{(2)} = 2$ Obviously, it would be easier to block the misbehaving flows if they only correspond to the low priority $w_{(1)}$. Hence the effectiveness of TCP protection is validated if the high-priority misbehaving flows are blocked successfully.

CHOKeW are compared with RIO in the simulations. The

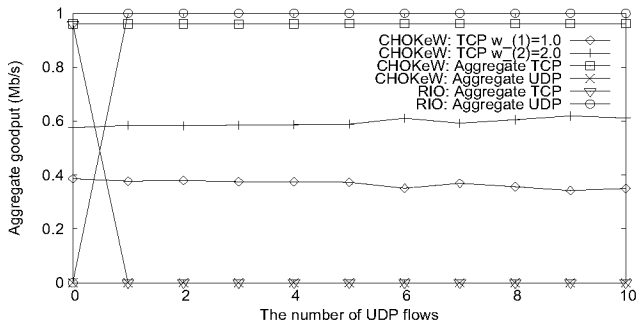

Fig. 9. The relationship between the aggregate goodput and the number of UDP flows

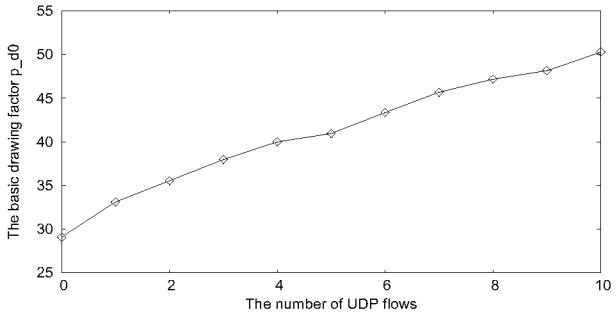

Fig. 10. p_0 versus the number of UDP flows

goodput versus the number of UDP flows is shown in Fig.9. Since no retransmission in UDP flows, goodput is equal to throughput for UDP. Even if the number of UDP flows increases from 1 to 10, for CHOKeW, the TCP goodput in each priority level (and hence the aggregate TCP throughput) 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. However, when high-speed UDP flows exist, the bandwidth shares for TCP flows in a RIO router are all nearly zero. RIO cannot protect TCP flows.

Fig.10 illustrates the relationship between p_0 and the number of UDP flows. As more UDP flows start, p_0 increases, but the value of p_0 is not high enough to block TCP flows. In this experiment, we also find that few packets of TCP flows are dropped due to buffer overflow. In fact, when the edge routers cooperate with the core routers, the high-speed misbehaving flows will be marked with lower priority at the edge routers. Therefore, CHOKeW should be able to block even more misbehaving flows than shown in Fig.9 and p_0 should also be smaller than shown in Fig.10.

D. Fairness

We compare the fairness of CHOKeW with that of RED and BLUE in this subsection. 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).

The network topology is shown 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% of the total number of flows.[4]

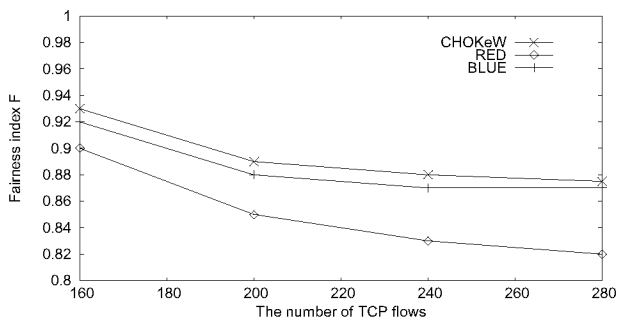

Fig. 11. The relationship between the fairness index and the number of flows for CHOKeW, RED and BLUE

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. When many flows are active, however, it is hard to measure the fairness from direct observation; in this case, we introduce the fairness index:

$$F = \frac{\left(\sum_{i=1}^{N} g_i\right)^2}{N \sum_{i=1}^{N} {g_i}^2} \tag{3}$$

where N is the number of active flows during the observation period, and g_i ($i = 1, 2, \cdots, N$) represents the goodput of flow i. From (3) we know $F \in (0, 1]$. The closer the value of F is to 1, the better the fairness. In this paper we use g_i as goodput instead of throughput so that the TCP performance evaluation will more accurately reflect the successful delivery rate. Fig.11 shows the fairness index of CHOKeW, RED, and BLUE versus the number of TCP flows ranging from 160 to 280. The fairness decreases as the number of flows increases, however, CHOKeW provides better fairness than both RED and BLUE in this situation.

ACKNOWLEDGMENT

We would like to thank the U.S. Office of Naval Research for providing grant N000140210464 (Young Investigator Award), and the U.S. National Science Foundation for providing grant ANI-0093241 (CAREER Award).

IV. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a stateless, 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 shares to flows with higher priority, maintaining good fairness among flows with the same priority, and protecting TCP against high-speed unresponsive

[4]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.

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. Our simulation provided a quantitative investigation of CHOKeW, though future understanding of network dynamics may allow for more comprehensive models.

The parameter tuning is another area of exploration for future work on CHOKeW. When the priority-weight ratio $w_{(2)}/w_{(1)}$ is higher, the bandwidth shares being allocated to the higher-priority flows will be greater; meantime, considering the total available bandwidth does not change, the bandwidth shares allocated to the lower-priority flows will be smaller. The value of $w_{(2)}/w_{(1)}$ is particular to the application, the network environment, and the users' demand. The results of this research can be incorporated with price-based DiffServ networks to provide differentiated bandwidth allocation as well as TCP protection.

REFERENCES

[1] J. Bennet and H. Zhang, *WF^2Q: Worst Case Fair Weighted Fair Queuing*, IEEE INFOCOM'96, 1996

[2] S. Blake, D. Black, M. Carlson, *et al.*, *An Architecture for Differentiated Service*, IETF RFC 2475, December 1998

[3] U. Bodin, O. Schelen, and S. Pink, *Load-Tolerant Differentiation with Active Queue Management*, ACM CCR'00. [Online]. Available: http://www.acm.org/sigcomm/ccr/archive/ccr-toc/ccr-toc-2000.html

[4] R. Braden, D. Clark, and J. Crowcroft, *et al.*, *Recommendations on Queue Management and Congestion Avoidance in the Internet*, IETF RFC 2309, April 1998

[5] K. Cho, *Flow-Valve: Embedding a Safety-Valve in RED*, IEEE GLOBECOM'99

[6] R. B. Cooper, *Introduction to Queueing Theory*, 2nd ed. Elsevier North Holland, 1981

[7] D. D. Clark, S. Shenker, and L. Zhang, *Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism*, ACM SIGCOMM'92

[8] D. D. Clark and W. Fang, *Explicit Allocation of Best Effort Packet Delivery Service*, IEEE/ACM Transactions on Networking, August 1998, 6(4):362–373

[9] A. Demers, S. Keshav, and S. Shenker, *Analysis and simulations of a Fair Queueing algorithm*, ACM SIGCOMM'89

[10] S. Floyd and V. Jacobson, *Random Early Detection Gateways for Congestion Avoidance*, IEEE/ACM Transaction on Networking, Aug.1993, 1(4):397–413

[11] S. Floyd, *RED: Discussions of Setting Parameters*, November 1997. [Online]. Available: http://www.icir.org/floyd/REDparameters.txt

[12] S. Floyd and K. Fall, *Promoting the Use of End-to-End Congestion Control in the Internet*, IEEE/ACM Transaction on Networking, Aug.1999, 7(4):458–472

[13] J. Heinanen, F. Baker, W. Weiss, *et al.*, *Assured Forwarding PHB Group*, IETF, RFC 2597, 1999

[14] R. Mahajan and S. Floyd, *Controlling High-Bandwidth Flows at the Congested Router*, ICSI Tech Report TR-01-001, April 2001. [Online]. Available: http://www.icir.org/red-pd/

[15] P. Marbach, *Pricing Differentiated Services Networks: Bursty Traffic*, IEEE INFOCOM'01

[16] N. Nichols, S. Blake, F. Baker, *et al.*, *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*, IETF RFC 2474, Dec.1998

[17] *ns-2 (Network Simulator version 2)*. [Online]. Available: http://www.isi.edu/nsnam/ns/

[18] J. Padhye, V. Firoiu, D. Towsley, *et al.*, *Modeling TCP Throughput: A Simple Model and its Empirical Validation*, ACM SIGCOMM'98

[19] R. Pan, B. Prabhakar, and K. Psounis, *CHOKe: A Stateless Active Queue Management Scheme for Approximating Fair Bandwidth Allocation*, IEEE INFOCOM'01

[20] A. K. Parekh and R. G. Gallager, *A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: the Single-Node Case*, IEEE/ACM Transaction on Networking, Aug.1993, 1(3):344–357

[21] A. K. Parekh and R. G. Gallager, *A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: the Multiple-Node Case*, IEEE/ACM Transaction on Networking, Aug.1994, 2(2):137–150

[22] S. Ramabhadran and J. Pasquale, *Stratified Round Robin: A Low Complexity Packet Scheduler with Bandwidth Fairness and Bounded Delay*, ACM SIGCOMM'03

[23] I. Stoica, S. Shenker, and H. Zhang, *Core-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocation in High Speed Networks*, ACM SIGCOMM'98

[24] S. Suri, G. Varghese, and G. Chandramenon, *Leap Forward Virtual Clock: A New Fair Queueing Scheme with Guaranteed Delay and Throughput Fairness*, IEEE INFOCOM '97, April 1997

[25] A. Tang, J. Wang, and S. H. Low, *Understanding CHOKe*, IEEE INFORCOM'03

[26] S. Wen and Y. Fang, *CHOKeW Patch on ns*, April 2005. [Online]. Available: http://www.ecel.ufl.edu/ wen/chokew.zip