A TCP-Friendly Stateless AQM Scheme for Fair Bandwidth Allocation

Cheng-Yuan Ho[a], Yi-Cheng Chan[b], and Yaw-Chung Chen[a]
[a] Department of Computer Science and Information Engineering, National Chiao Tung University
[b] Department of Computer Science and Information Engineering, National Changhua University of Education
cyho@csie.nctu.edu.tw, ycchan@cc.ncue.edu.tw, ycchen@csie.nctu.edu.tw

Abstract

Queue management, bandwidth share, and congestion control are very important to both robustness and fairness of the Internet. In this article, we investigate the problem of providing a fair bandwidth allocation to those flows that share congested link in a router. A new TCP-friendly router-based AQM (active queue management) scheme, termed WARD, is proposed to approximate the fair queueing policy. WARD is a simple packet dropping algorithm with a random mechanism, and discriminates against the flows which transmit more packets than allowed. By doing this, it not only protects TCP connections from UDP flows, but also solves the problem of competing bandwidth among different TCP versions such as TCP Vegas and Reno. In addition, WARD works quite well for TCP flow isolation even with different round trip times. In other words, WARD improves the unfair bandwidth allocation properties. Furthermore, it is stateless and easy to implement, so WARD controls unresponsive or misbehaving flows with a minimum overhead.

1 Introduction

The Internet provides a connectionless, best effort, end-to-end packet service using the IP protocol. Its performance and stability depend on congestion avoidance algorithm and implemented in the transport layer protocols to provide good service under heavy load. In practice, TCP (Transmission Control Protocol) has been widely deployed to carry the majority of the traffic in the Internet. TCP helps a traffic source to determine how much bandwidth is available in the network, adjust its transmission rate accordingly, keeps the network from being overloaded and has become a crucial factor in the robustness and stability of the Internet.

However, a lot of TCP implementations do not include the congestion avoidance mechanism either deliberately or by accident [1]. Moreover, some applications utilize the UDP (User Datagram Protocol) instead, which does not employ end-to-end flow and congestion control. Rather, the sending rate is set by the application and normally a little or even no consideration of network congestion is taken into account during the transmission. As a result, UDP flows aggressively use up more bandwidth than other TCP compatible flows. This could eventually cause "Internet Meltdown" or result in two major problems already identified in the Internet: *unfairness* and *congestion collapse*. Therefore, it is necessary to have router mechanisms to shield responsive flows from unresponsive or aggressive flows and to effectively detect congestion in the network.

Besides, even there is no UDP flow in the network, an unfairness problem may still occur when the connections with different TCP versions such as TCP Vegas [2] and TCP Reno [3] coexist [4]. It is because their slow start, congestion avoidance, and fast retransmit mechanisms are different. For example, TCP Vegas uses the difference between the expected and actual throughput, while TCP Reno detects the packet loss as an indicator, to estimate the available bandwidth in the network, to control the throughput, and to avoid congestion. TCP Vegas achieves much higher throughput, and has a fairer and stabler bandwidth share than TCP Reno does. However, TCP Reno is an aggressive control scheme in which each connection captures more bandwidth until the transmitted packets are lost. Meanwhile, TCP Vegas is a conservative scheme in which each connection obtains a proper bandwidth. Thus, the TCP Reno connections take bandwidth from the TCP Vegas connections when they coexist. To solve this unfairness problem, several end-system-based solutions such as TCP NewVegas, TCP Vegas-A, TCP Vegas+ and so on have been proposed. Nevertheless, these mechanisms merely solve the problem in some particular network environments. As a result, to regulate the flows causing the unfairness problem, the router-based schemes are more appropriate.

The rest of this paper is organized as follows. In Section 2, related work is reviewed. Section 3 briefly explains the WARD's goals and describes its algorithm in detail. The simulation results are presented in Section 4. Finally, a summary of this work is addressed in Section 5.

Proceedings of the Joint International Conference on Autonomic and Autonomous Systems
and International Conference on Networking and Services (ICAS/ICNS 2005)
0-7695-2450-8/05 $20.00 © 2005 IEEE

2 Related Works

One of the most important objectives of a router-based scheme is its ability to achieve fairness among competing flows, particularly in the case where unresponsive high-bandwidth best-effort flows share the bottleneck link.

Drop Tail: The simplest buffer management scheme for a FIFO queue is Drop Tail, which just drops an incoming packet when the queue is already full. However, it was pointed out that each congestion period introduces global synchronization in the network. When the queue overflows, packets are often dropped from several TCP connections, and these connections decrease their windows at the same time. This results in a loss of throughput at the gateway.

RED: A router implementing RED (Random Early Detection) [5] maintains a single FIFO to be shared by all the flows, and drops an arriving packet randomly during periods of congestion. The drop probability increases with the level of congestion. Since RED acts in anticipation of congestion, it does not suffer from the "lock out" and "full queue" problems inherent in the widely deployed Drop Tail mechanism. By keeping the average queue-size small, RED reduces the delays experienced by most flows. We would like to describe the details of RED because the same or alike schemes are also used in FRED and CHOKe.

The RED gateway calculates the average queue size, using a low-pass filter with an exponential weighted moving average. The average queue size is compared to two thresholds, the minimum threshold and the maximum threshold. When the average queue size is less than the minimum threshold, no packets are dropped. When the average queue size is greater than the maximum threshold, every arriving packet is dropped. When the average queue size is between the minimum and the maximum threshold, each arriving packet is dropped with probability, which is a function of the average queue size.

Although RED successfully solves the "global synchronization" problem, it doesn't provide fair queuing when there exist unresponsive UDP flows. It is because UDP traffic doesn't adjust its transmission rate according to congestion in the network. As a result, they aggressively use up more bandwidth than other TCP compatible flows.

FRED: FRED (Fair Random Early Detection) [6] is a modified version of RED meant to solve the unfairness problem and to improve RED's ability for distinguishing unresponsive users. Two global parameters,the minimum and maximum number of packets allowed to be queued at the router for each flow, are added. Each active flow needs to maintain the number of packets buffered and the number of times it was failed to respond to congestion notification. Lastly, FRED introduces an average per-flow queue size as a global variable. It protects low-speed flows by guaranteeing them a minimum buffer space. In order to manage unresponsive flows, FRED enforces per-flow queueing limits. Overall, FRED can achieve fairness under several situations with minimal modifications to RED. However, the main drawback of FRED is the amount of per-flow state information that needs to be maintained, this limits its scalability.

CHOKe: A design goal of CHOKe (CHOose and Keep for responsive flows) [1] is to keep the mechanism as simple as possible while controlling unresponsive flows. A small modification is proposed over a plain FIFO queue with RED active queue management to achieve this goal. When a packet arrives at a congested router, CHOKe draws a packet randomly from the FIFO buffer and compares it with the arriving packet. If they both belong to the same flow, then they are both dropped, else the randomly chosen packet is left intact and the arriving packet is admitted into the buffer with a probability that depends on the level of congestion. The reason for doing this is that a FIFO queue is more likely to have packets that belong to unresponsive flows more than those of responsive ones, and they are more likely to be chosen for comparison. Therefore, packets from unresponsive flows are likely to be dropped more often. However, CHOKe can control unresponsive flows only if there are more packets from those flows in the buffer at the time of congestion. This is due to the fact that CHOKe does not keep track of those unresponsive flows. In addition, some responsive flows might be punished unnecessarily as a result of its probabilistic algorithm.

Some mechanisms have rarely been implemented in a commercial high speed or backbone router because they still have a high space complexity. Thus, we propose a stateless and TCP-friendly queue management scheme.

3 Motivation, Goal, and WARD

Our work is motivated by the need for a simple, TCP-friendly, and stateless algorithm that can achieve fair bandwidth allocation and flow isolation. Although RED, FRED, and CHOKe do not estimate the number of active flows, they still have their own limits. For example, how to set the value of minimum and maximum thresholds in these algorithms to suit a topology with variable connections is a tough problem because RED, FRED, and CHOKe are sensitive to parameter settings. Moreover, improper parameter settings may lead to unsatisfactory TCP performance. We are seeking a solution to avoid these problems in the context of the Internet. In addition, we are motivated to find a solution that could penalize the "unresponsive" or "unfriendly" flows, such as UDP-based and bad implementations of TCP. Further, we hope that our solution could avoid global syn-

chronization, ensure low delays, keep buffer occupancies small, and bias against bursty traffic. Thus, we propose a stateless queue management algorithm, called WARD, which is discussed in the following paragraphs.

3.1 Mechanism Description

Suppose that a router maintains a single FIFO buffer for queueing the packets of all the flows that share an outgoing link. We describe an algorithm, WARD, that differentially penalizes unresponsive and unfriendly flows. In addition, even though the idea of WARD is similar to CHOKe, the method of WARD is different from CHOKe. The state, taken to be the number of active flows and the flow ID of each parameter packets, is assumed to be unknown to the algorithm. The only observable for the algorithm is the total occupancy of the buffer.

Before describing the WARD's algorithm, we give a weighted value to every position in the FIFO buffer. First, the index of every position is divided by the FIFO buffer size. Second, the weighted value of every position equals the first number beyond a decimal point of above result. For example, assuming the FIFO buffer size is 100, then the weighted values of the 1^{st} to 9^{th} position are 0.0, the 10^{th} to 19^{th} are 0.1, ..., the 90^{th} to 99^{th} are 0.9, and the last position is 1.0. Since the packets entering the beginning or the head of a buffer means that there is no congestion yet. On the other hand, the congestion may occur when the buffer is becoming full.

When a packet k, which may be queued into the position P, arrives at the buffer, we choose a uniformly distributed random decimal number U, which is no larger than 1. If U is bigger than the weighted value of position P, this packet k is queued into the FIFO buffer. Otherwise, the packet k is compared with two packets i and j which are randomly selected from the FIFO buffer. First, if these three packets have the same flow ID, they will be all dropped. Second, the flow ID of either packet i or packet j is same as that of the packet k, these two packets with same ID will be both dropped. Third, if packets i and j have the same flow ID, which is different from the flow ID of the packet k, both packets i and j will be dropped too. Otherwise, the randomly selected packets i and j are kept in the buffer (in the same position as before) and the arriving packet k is queued in the position P. Besides, in the sensitive case, the buffer is full when the packet k comes in. WARD will do the above steps except queueing the packet k in the last step. A flow chart of the WARD's algorithm is given in Fig. 1. WARD is a truly stateless algorithm. It does not require any special data structure. Compared to a pure FIFO queue, WARD just needs to perform few simple extra operations. We may say that WARD is embedded in Drop Tail, which is a commonly used scheme, so there is no big problem in using WARD.

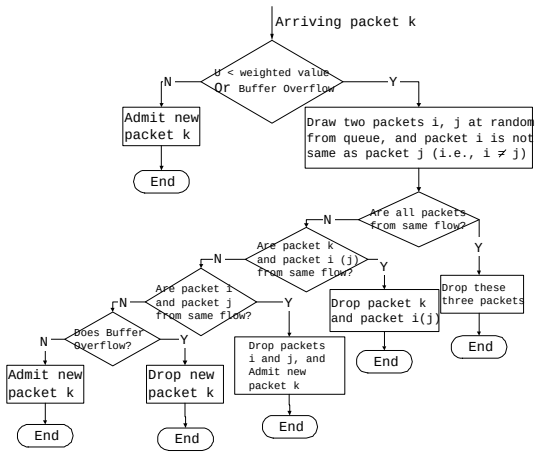

Figure 1. The WARD Algorithm.

Because the space is limited, the reasons for choosing two packets randomly, the a deterministic fluid model of TCP/WARD system and the UDP throughput behavior with WARD please refer to [7]. In the following Sections, we demonstrate that the proposed scheme improves the fairness of bandwidth allocation based on the numerical results.

4 Simulation Results

This section presents simulation results of WARD's performance in penalizing misbehaving flows and thus approximating fair bandwidth allocation. We use the RED, CHOKe, and Drop Tail schemes, whose complexities are close to that of WARD, for comparison. The simulations range over a spectrum of network configurations and traffic mixes. The results are presented in four parts: Single Unresponsive Flow, Multiple Unresponsive Flows, TCP Sources With Different Versions, and TCP Sources With Different Round Trip Times. Since the space is limited, we only show the main simulation results. Other simulation results please refer to [7].

4.1 Simulation Setup

We use the network simulator *ns2* [8], version 2.26, and the dumbbell topology shown in Fig. 2 to assess the performance of WARD, which will be compared with Drop Tail, RED, and CHOKe. The congested link in this network is between the router R1 and R2. The link, with capacity of 1 Mbps, is shared by m TCP (with one version) and n UDP, or m TCP Vegas and n TCP Reno flows. An end host is connected to the routers using a 10 Mbps link, which is ten times the bottleneck link bandwidth. All links have a small propagation delay of 1 ms except the last scenario, so that

the delay experienced by a packet is mainly caused by the buffer delay rather than the transmission delay. The maximum window size of TCP is set to 500 segment such that it doesn't become a limiting factor of a flow's throughput. The TCP flows are derived from FTP sessions which transmit large sized files. The UDP hosts send packets at a CBR (constant bit rate) of γ Kbps, where γ is variable. The size of all packets are set to 1 K Bytes.

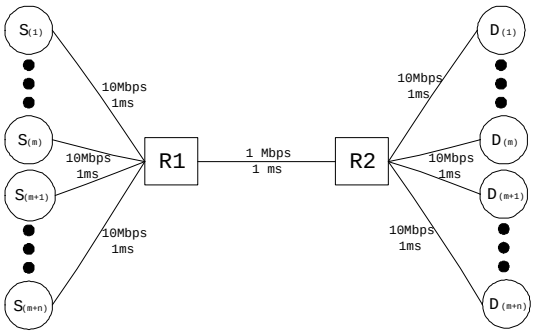

Figure 2. Simulation Topology.

4.2 Single Unresponsive Flow

To study how much bandwidth a single nonadaptive UDP source can obtain when the routers use different queue management schemes, we set up the simulation with 32 TCP sources (*Flow*1 to *Flow*32) and 1 UDP source (*Flow*33) in the network. The UDP source sends packets at a rate of 2 Mbps, twice the bandwidth of the bottleneck link, such that the link R1–R2 becomes congested.

To observe how WARD achieves fair bandwidth allocation, the individual throughput of each of the 33 connections with buffer size 132 (4 packets per flow), along with their ideal fair shares, are plotted in Fig. 3. Although the throughput of the UDP flow (*Flow*33) is still higher than the rest of the TCP flows, it can be seen that each TCP is allocated a bandwidth relatively close to its fair share. Furthermore, the dropping probability of UDP flow is about 96%. Since a packet may be dropped because of a match or buffer overflow in WARD. A misbehaving flow, which has a high arrival rate and a high buffer occupancy, incurs packet dropping mostly due to matches. On the other hand, the packets of a responsive flow are unlikely to be matched, so they will be dropped mainly because of buffer overflow.

The throughput of the UDP flow under different queue management algorithms: Drop Tail, RED, CHOKe, and WARD, is plotted in Fig. 4. The minimum threshold in the RED and CHOKe is set to 100, allowing on average around 3 packets per flow in the buffer before a router starts dropping packets. Following [5], we set the maximum threshold

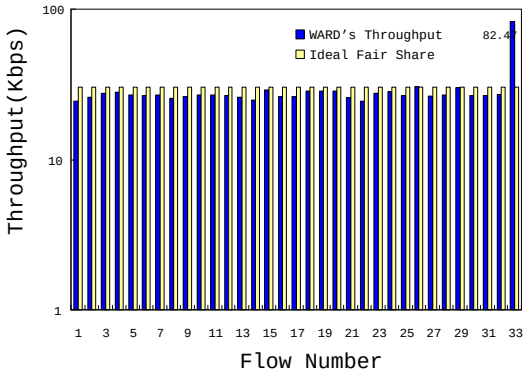

Figure 3. Throughput Per Flow.

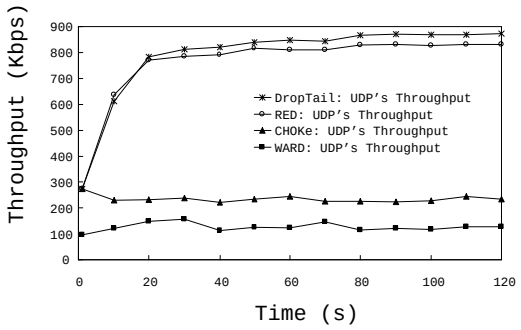

Figure 4. UDP Throughput Comparison.

to be twice the minimum threshold. In addition, with no partiality, the buffer size of Drop Tail and WARD is fixed at 200 packets due to the maximum threshold mentioned above. From Fig. 4, we could clearly see that the Drop Tail and RED gateways do not discriminate against unresponsive flows. The UDP flow takes away more than 85% of the bottleneck link capacity and the TCP connections only obtain the remaining 150 Kbps. Although CHOKe improves the throughput of the TCP flows dramatically by limiting the UDP throughput to 250 Kbps, however, the UDP throughput is still much higher than each of TCP throughput. WARD boosts the total TCP flows' throughput from 150 Kbps (in Drop Tail gateway) to at least 850 Kbps and limits UDP throughput to at most 150 Kbps, which is only around 15% of the link capacity.

With the fixed buffer size (200), we vary the UDP arrival rate γ to ivestigate WARD's performance under different traffic load conditions. The simulation results are summarized in Fig. 5, where the UDP's throughput versus the UDP flow arrival rate is plotted. The drop percentage of the UDP flow is also shown in the Fig. 5. From the plots, we can observe some characteristics of WARD comparing with

CHOKe. (1) When UDP arrival rate is lower than fair share bandwidth, WARD protects the throughput of UDP flow as best it can. For example, there is no packets dropped from UDP flow while its arrival rate is 10 Kbps. As the UDP arrival rate increases, the drop percentage goes up as well. For instance, WARD drops 21.9% of the UDP packets when its rate achieves 100 Kbps. Moreover, WARD drops almost all packets (99.7%) while the arrival rate reaches 10 Mbps. Since the probability of obtaining a matched UDP packet increases with the increasing arrival rate of UDP flow. In other words, the packets of UDP flow have higher probability to be matched. (2) The average throughput of TCP flows with WARD algorithm is higher than that with CHOKe. Since the drop percentage of UDP flow by using WARD is bigger than CHOKe when its rate is higher than the fair share bandwidth. In addition, we don't compare WARD with Drop Tail and RED here because the comparisons of CHOKe, RED, and Drop Tail is already shown in [1].

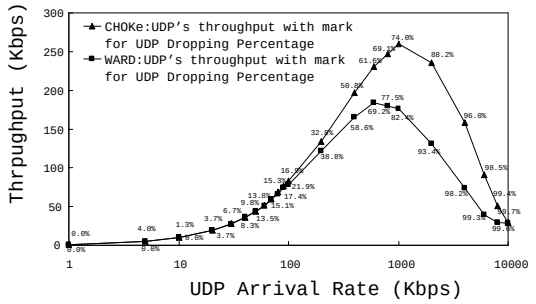

Figure 5. Performance versus UDP Rates.

4.3 Multiple Unresponsive Flows

We follow the traffic model mentioned above (i.e., Fig. 2). Recall that the first model includes 32 TCP flows (*Flow*1 to *Flow*32) and 1 UDP flow (*Flow*33), we don't change any variables here except the number of sources with TCP or UDP flows. The second traffic model includes 31 TCP flows (*Flow*1 to *Flow*31) and 2 UDP flows (one (*Flow*32) sending rate is 2 Mbps and the other (*Flow*33) is 1 Mbps), and there are 29 TCP flows (*Flow*1 to *Flow*29) and 4 UDP flows (*Flow*30 to *Flow*33) in the third traffic model. The rates of the UDP flows are 2 Mbps, 1 Mbps, 100 Kbps, and 30 Kbps, which is smaller than the ideal fair share throughput, respectively. The results of these two traffic models are shown in Table 1, where Thr is throughput (Kbps) of a flow, SR is sending rate (Kbps) of a UDP flow, and P_{drop} is dropping probability, respectively. In addition, the ideal fair share throughput is 30.3 Kbps. From Table 1, we could see that the UDP flow with a low rate is also treated fairly. In other words, the dropping probability is bigger when the sending rate becomes higher. When the number of TCP and UDP flows change, the WARD algorithm tries to achieve fair queueing. If we concern the input rates of UDP flows, the performance is really satisfactory.

Table 1. Average Throughput of TCP and Throughput of each UDP.

2UDP	TCP	UDP1	UDP2	UDP3	UDP4
Thr	26.3	65.1	41.2	—	—
SR	—	2000	1000	—	—
P_{drop}	—	96.75%	95.88%	—	—
4UDP	**TCP**	**UDP1**	**UDP2**	**UDP3**	**UDP4**
Thr	24.8	64.4	62.7	57.6	25.5
SR	—	2000	1000	100	30
P_{drop}	—	96.78%	93.73%	42.40%	15.11%

4.4 TCP Sources With Different Versions

When a TCP Vegas user competes with other TCP Reno users, it does not receive a fair share of bandwidth due to the conservative congestion avoidance mechanism used by TCP Vegas [4]. To avoid this situation, how to drop some packets sent by Reno before buffer overflow is an issue.

Table 2. Fairness Index of Different TCP Versions and Buffer Sizes.

Fairness Index					
B	Vegas – Reno flows				
	20 - 0	15 - 5	10 - 10	5 - 15	0 - 20
60	99.7%	99.6%	99.4%	99.5%	99.5%
70	99.8%	99.6%	99.7%	99.4%	99.4%
80	99.7%	99.7%	99.5%	99.6%	99.6%
90	99.7%	99.8%	99.6%	99.5%	99.3%
100	99.1%	99.6%	99.7%	99.6%	99.5%
110	99.9%	98.7%	99.2%	99.5%	99.8%
120	99.5%	99.5%	99.2%	99.6%	99.6%

In this subsection, we use 20 TCP sources in the dumbbell topology, as shown in Fig. 2. Moreover, the ideal fair share throughput of each flow is 50 Kbps. Among these 20 TCP sources, the source traffic from TCP Vegas are decreasing from 20 to 0. We show the simulation results of using WARD algorithm with different buffer sizes in Table 2, where B is buffer size. In Table 2, we could see the fairness index of each buffer size over 99% regardless of the number of sources using either TCP Vegas or TCP Reno in

the network. Thus, we think that TCP Vegas could compete the network resources with TCP Reno when using WARD algorithm in a router.

We compare the fairness index of WARD with other algorithms, such as Drop Tail, RED, CHOKe, and FRED. In addition, the physical buffer size is 120 for each algorithm, and minimum threshold is set to 60 and maximum threshold is set to 120 for RED, CHOKe, and FRED algorithms. The result is shown in Table 3. We could see that the performance of WARD is much better than others, even the algorithm requires full per-flow state information.

Table 3. Fairness Index of Different Queue Management Algorithms.

Fairness Index					
Algorithm	Vegas – Reno flows				
	20–0	15–5	10–10	5–15	0–20
Drop Tail	96.3%	60.7%	76.9%	95.8%	97.7%
RED	96.3%	74.2%	83.4%	91.3%	98.9%
CHOKe	96.3%	84.3%	92.9%	97.1%	99.3%
FRED	96.3%	62.2%	78.4%	91.5%	99.7%
WARD	99.4%	99.5%	99.2%	99.6%	99.6%

4.5 TCP Sources With Different RTTs

When TCP Reno connections feature different RTTs, the sources with shorter RTT will get more network resource. Different from TCP Reno, TCP Vegas is not biased against the connections with longer RTT. Thus, we show the results of TCP Reno sources with different RTTs in Fig. 6. There are three cases in our test. Case 1: There are 20 sources competing 1 Mbps bottleneck bandwidth. The RTT of ten sources are 20 ms, and 40 ms for the others. From simulation result, the average throughput of shorter RTT is 47.31 Kpbs, and that of longer RTT is 48.04 Kbps. We add another 10 sources with 60 ms RTT to the network topology in Case 2. The average throughput of sources with 20 ms, 40 ms, and 60 ms RTTs are 32.05, 31.65, 31.42 Kbps respectively. Similarly, 10 sources with 80 ms RTT are added in Case 3, the result is demonstrated in Fig. 6. Although these 40 sources come with different RTTs, the average throughput for each RTT is about 24 Kbps.

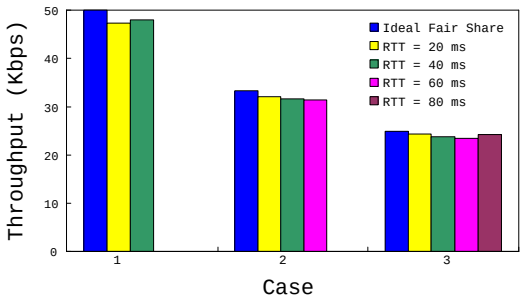

Figure 6. TCP Sources with Different RTTs.

5 Conclusions

In this article, we introduce router-based approaches and propose a packet dropping scheme, called WARD. It aims to approximate fair queueing at a minimal implementation cost. Simulations demonstrate that it works well in protecting congestion-sensitive flows from congestion-insensitive or congestion-causing flows. Furthermore, it solves the problem of competing bandwidth among different TCP versions, such as TCP Vegas and TCP Reno. Further work involves studying the performance and spatial characteristics analysis of this algorithm under a wider range of parameters, network topologies and real traffic traces, obtaining more accurate theoretical models and insights, and considering hardware implementation issues.

References

[1] R. Pan, B. Prabhakar, and K. Psounis, 'CHOKe: A Stateless Active Queue Management Scheme for Approximating Fair Bandwidth Allocation', *IEEE INFOCOM'2000, Vol. 2, pp. 942-951, Mar. 2000.*

[2] L. S. Brakmo and L. L. Peterson, 'TCP Vegas: End to End Congestion Avoidance on a Global Internet', *IEEE J. Select. Areas Commun., Vol. 13, pp. 1465-1480, Oct. 1995.*

[3] V. Jacobson, 'Modified TCP Congestion Avoidance Algorithm', *end-to-end-interest, Apr. 1990.*

[4] J. Mo, R.J. La, V. Anantharam, and J. Walrand, 'Analysis and Comparison of TCP Reno and Vegas', *IEEE INFOCOM'99, Vol. 3, pp. 1556-1563, Mar. 1999.*

[5] S. Floyd and V. Jacobson, 'Random Early Detection Gateways for Congestion Avoidance', *IEEE/ACM Transaction on Networking, Vol. 1, Issue 4, pp. 397-413, Aug. 1993*

[6] D. Lin and R. Morris, 'Dynamics of Random Early Detection', *ACM SIGCOMM, pp. 127-137, Sep. 1997.*

[7] C. Y. Ho, Y. C. Chan, and Y. C. Chen, 'WARD: A TCP-Friendly Stateless AQM Scheme', *Submitted to IEEE/ACM Transactions on Networking,* http://mmlabwww.csie.nctu.edu.tw/cyho/WARD.pdf

[8] http://www.isi.edu/nsnam/ns/