WARD: A deterministic fluid model

C.-Y. Ho, Y.-C. Chan and Y.-C. Chen

Abstract: Queue management, bandwidth share and congestion control are very important to both the robustness and fairness of the Internet. A new TCP-friendly router-based active queue management scheme, called WARD, approximates the fair queueing policy. WARD is a simple packet dropping algorithm with a random mechanism and discriminates against the flows which submit more packets per second than is allowed by their fair share. By doing this, it not only protects transmission control protocol (TCP) connections from user datagram protocol (UDP) flows, but also solves the problem of competing bandwidth among different TCP versions, such as TCP Vegas and TCP Reno. Furthermore, it is stateless and easy to implement, so WARD controls unresponsive or misbehaving flows with a minimum overhead. In this article, we present a deterministic fluid model of TCP/WARD system, and explain the UDP throughput behaviour with WARD. Also, we prove that, provided the number of TCP flows is large, the UDP bandwidth share peaks at $(2e)^{-1} = 0.184$ when UDP input rate is slightly larger than link capacity and drops to zero as UDP input rate tends to infinity.

1 Introduction

Transmission control protocol (TCP) is believed to be largely responsible for preventing congestion collapse while the Internet has undergone dramatic growth in the last decade. By using additive increase/multiplicative decrease (AIMD) strategy, TCP helps a traffic source to determine how much bandwidth is available in the network and adjust its transmission rate accordingly. Indeed, numerous measurements have consistently shown that more than 90% of the traffic on the current Internet is still TCP packets, which, fortunately, are congestion controlled.

However, a lot of TCP implementations do not include the congestion avoidance mechanism either deliberately or by accident [1]. Moreover, different types of applications, especially multimedia and audio/video streaming applications, are being increasingly deployed over the Internet. Such applications utilise the user datagram protocol (UDP) instead, which does not employ end-to-end flow and congestion control. Rather, the sending rate is set by the application and normally no 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.

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, 5] since their slow start, congestion avoidance and fast retransmit mechanisms are different. For example, TCP Vegas uses the difference between the expected and the actual throughput, while TCP Reno detects the packet loss as an indicator, to estimate the available bandwidth in the network, control the throughput and avoid congestion. TCP Vegas achieves much higher throughput and has a fairer and stabler bandwidth share than TCP Reno does [2]. 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 [6].

Accordingly, it is necessary to have router mechanisms to shield responsive flows from unresponsive or aggressive flows, to effectively detect congestion in the network, to achieve fair share among flows, and to provide a satisfactory quality of service to all users. This has motivated several active queue management schemes, e.g. [1, 7–14], that aim at penalising aggressive flows and ensuring fairness. The scheme, WARD, of [14] is particularly interesting in that it does not require any state information and yet can provide a minimum throughput to TCP flows. In this article, we provide an analytical model of WARD that explains both the throughput behaviour and the feedback equilibrium of TCP/WARD system. Here, we focus on a deterministic fluid model of TCP/WARD system with UDP flows. The discussion of TCP not-friendly will be addressed in the future work. Moreover, some simulation results of different TCP mechanisms can be found in [14].

The rest of this paper is organised as follows. Section 2 describes the WARD algorithm in detail. We present in Section 3 a deterministic fluid model that explicitly models the feedback equilibrium of TCP/WARD system. In Section 4, by analysing the TCP/WARD model, we prove that, provided the number of TCP flows is large, the UDP bandwidth share peaks at $(2e)^{-1} = 0.184$, and drops to zero as UDP input rate tends to infinity. The simulation results are presented in Section 5. Finally, a summary of this work is provided in Section 6.

© The Institution of Engineering and Technology 2007

doi:10.1049/iet-com:20060380

Paper first received 4th April 2006 and in revised form 4th February 2007

C.-Y. Ho and Y.-C. Chen are with the Department of Computer Science, National Chiao Tung University, No. 1001, Ta Hsueh Road, Hsinchu City 30050, Taiwan, Republic of China

Y.-C. Chan is with the Department of Computer Science and Information Engineering, National Changhua University of Education, No. 1, Jin-De Road, Changhua City 50007, Taiwan, Republic of China

E-mail: cyho@csie.nctu.edu.tw

2 WARD mechanism description

Suppose that a router maintains a single first in first out (FIFO) buffer for queueing the packets of all the flows that share an outgoing link. We describe an algorithm, WARD, that differentially penalises unresponsive and unfriendly flows. In addition, even though the idea of WARD is similar to CHOose and Keep for responsive flow (CHOKe) [1], 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 1st to 9th position are 0.0, the 10th to 19th are 0.1, ..., the 90th to 99th are 0.9, and the last position is 1.0. This is because 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.

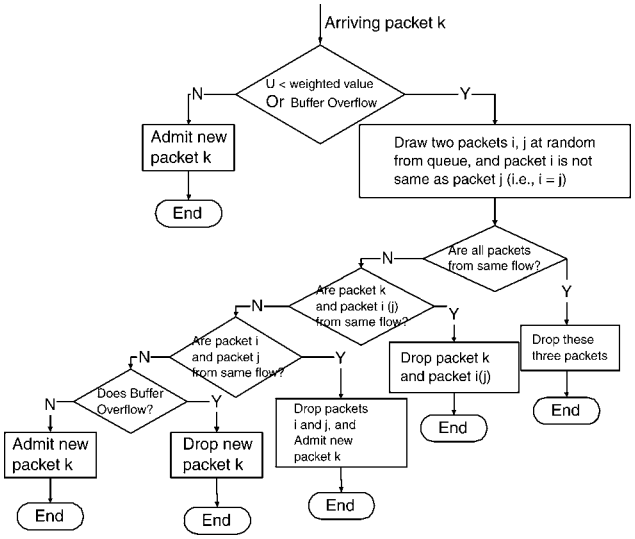

Fig. 1 *WARD algorithm*

There are two reasons for choosing two packets randomly from the FIFO buffer. (1) The FIFO buffer is more likely to have packets belonging to a misbehaving flow and hence these packets are more likely to be chosen for comparison. (2) If we randomly choose more than two packets from the buffer, the comparison of those packets will be complicate. In addition, although the dropping rate of the unresponsive flows for choosing more packets is higher (just little) than that for choosing two packets, the responsive flows' performance and total throughput decrease according to simulation results.

In short, there are two major differences between WARD and CHOKe. One is that WARD is embedded in drop tail and CHOKe is embedded in random early detection (RED) [7]. Therefore the behaviour of two schemes to deal with an incoming packet is not the same. The other is that in order to protect congestion-sensitive flows from congestion-insensitive or congestion-causing flows effectively, WARD selects more packets queued in the buffer to compare with an incoming packet than CHOKe does.

In the following sections, we now present a deterministic fluid model of TCP/WARD system, explain the UDP throughput behaviour with WARD, and demonstrate that the proposed scheme improves the fairness of bandwidth allocation based on the numerical results.

3 Model

We focus on the single bottleneck FIFO buffer where packets are queued and drained at a rate of c packets per second. The buffer is shared by N identical TCP flows and a single UDP flow. All TCP flows have a common round trip propagation delay of d seconds. Similar to the work in [15], we assume the system is stable and model its equilibrium behaviour.

3.1 Notations

Quantities (rate, backlog, dropping probability etc.) associated with the UDP flow are indexed by 0. Those associated with TCP flows are indexed by $i = 1, \ldots, N$. These are equilibrium quantities which we assume exist. We collect here the definitions of all the variables and some of their obvious properties.

B Physical buffer sise of a router.

b_i Packet backlog from flow i, $i = 0, 1, \ldots, N$.

b Total backlog: $b = \sum_{i=0}^{N} b_i$.

P_i The position that an incoming packet of flow i, $i = 0, 1, \ldots, N$, is queued into. Moreover, $P_i = b + 1$.

W_i The probability that an incoming packet of flow i, $i = 0, 1, \ldots, N$, is compared with two packets which are randomly selected from the FIFO buffer by WARD

$$W_i = \frac{\left\lfloor \frac{10P_i}{B} \right\rfloor}{10}$$

Moreover, if $P_i >$ B (buffer overflow), $W_i = 1$.

h_i^3 The probability that three packets of flow i (including an incoming packet), $i = 0, 1, \ldots, N$, are dropped by WARD

$$\frac{b_i(b_i - 1)}{b(b-1)} W_i$$

h_i^2 The probability that two packets of flow i (including an incoming packet), $i = 0, 1, \ldots, N$, are dropped by WARD

$$\frac{2b_i(b-b_i)}{b(b-1)}W_i$$

d_i^2 The probability that two packets of flow j (not including an incoming packet), for some $j \neq i$, are dropped by WARD

$$\frac{(b-b_i)(b_j-1)}{b(b-1)}W_i$$

h_i^1 The probability that an incoming packet of flow i, $i = 0, 1, \ldots, N$, is dropped by WARD while $W_i = 1$, or is queued into a buffer when $W_i \neq 1$

$$\begin{cases} \dfrac{(b-b_i)(b-b_i-b_j)}{b(b-1)}W_i = 1 - h_i^3 - h_i^2 - d_i^2, \\ \quad \text{if } W_i = 1 \\ \dfrac{(b-b_i)(b-b_i-b_j)}{b(b-1)}W_i = W_i - h_i^3 - h_i^2 - d_i^2, \\ \quad \text{otherwise} \end{cases}$$

h_i^0 The probability that an incoming packet of flow i, $i = 0, 1, \ldots, N$, is queued into a buffer

$$\begin{cases} (1-W_i) + d_i^2 = d_i^2, & \text{if } W_i = 1 \\ (1-W_i) + h_i^1 + d_i^2 = 1 - h_i^3 - h_i^2, & \text{otherwise} \end{cases}$$

p_i Overall probability that a packet of flow i, $i = 0, 1, \ldots, N$, is dropped by WARD before it gets through

$$\begin{cases} 3h_i^3 + 2h_i^2 + h_i^1, & \text{if } W_i = 1 \\ 3h_i^3 + 2h_i^2, & \text{otherwise} \end{cases} \quad (1)$$

The explanation of (1) is provided later

x_i Source rate of flow i, $i = 0, 1, \ldots, N$. The spatial properties of WARD are insensitive to the specific TCP algorithm. In general, $x_i = f(p_i, \tau)$, $i = 1, \ldots, N$, for some function f as a function of overall loss probability p_i and queueing delay τ at equilibrium.

τ Common queueing delay. Round-trip time is $d + \tau$.

It is important to keep in mind that x_0 is the only independent variable; all other variables listed above are functions of x_0, though this is not made explicit in the notations. In the light of the work [15], we also use μ_i as a shorthand for flow i's normalised bandwidth share, $\mu_i := x_i(1-p_i)/c$, $i = 0, 1, \ldots, N$.

3.2 TCP/WARD model

A packet may be dropped, either on arrival due to WARD, or after it has been admitted into the queue when a future arrival, which may be from the same flow or not, triggers a comparison. Let p_i be the probability that a packet from flow i is eventually dropped. To see why p_i is related to WARD dropping probability according to (1), note that every arrival from flow i can trigger either 0 packet loss from the buffer, or 1, 2, or 3 packet losses due to WARD. In addition, the respective probabilities of these events are described above. Hence, each arrival to the buffer is accompanied by an average packet loss of

$$\begin{cases} 3h_i^3 + 2h_i^2 + 1h_i^1 + 2d_i^2, & \text{if } W_i = 1 \\ 3h_i^3 + 2h_i^2 + 0h_i^0 + 2d_i^2, & \text{otherwise} \end{cases}$$

We take the overall loss probability p_i to be

$$\begin{cases} 3h_i^3 + 2h_i^2 + h_i^1, & \text{if } W_i = 1 \\ 3h_i^3 + 2h_i^2, & \text{otherwise} \end{cases}$$

because d_i^2 is the probability that two packets of flow j, for some $j \neq i$, are dropped by WARD. We now justify this probability from another perspective.

Consider a packet of flow i that eventually goes through the queue without being dropped. The probability that it is not dropped on arrival is h_i^0. Once it enters the queue, it takes τ time to go through it. In this time period, there are on average $\tau\sum_{m=0}^{N} x_m$ packets from these $(N+1)$ flows that arrive at the queue. We assume that the probability (S_i) that this packet is not dropped by WARD is

$$S_i = \left(1 - \frac{1}{b}\right)^{\tau x_i} \left(1 + 2\frac{1-b_i}{b(b-1)}\right)^{\tau\left(\sum_{m=0}^{N} x_m - x_i\right)} \quad (2)$$

Hence, the overall probability that a packet of flow i survives the queue is

$$1 - p_i = h_i^0 S_i \quad (3)$$

According to the work [15], when the link is fully utilised, the flow throughputs sum to link capacity

$$\sum_{i=0}^{N} x_i(1-p_i) = c \quad (4)$$

This completes the description of the model. Let z denote the all dependent variables, which are defined above. Similar to the work in Ref. [15], (1)–(4) and dependent variables can be expressed as

$$F(z, x_0) = 0 \quad (5)$$

This can be regarded as implicitly defining z in terms of x_0. The nonlinear equation (5) that models the TCP/WARD system can be solved numerically by minimising the quadratic cost. A direct search method [16] for multidimensional unconstrained nonlinear minimisation implemented in Matlab is used for this optimisation problem. The solution is accurately validated with *ns*2 simulations; see Section 5.2.

4 Throughput analysis

In this section, we make two approximations to above model. They allow us to readily derive the maximum achievable UDP throughput and a proof that UDP throughput approaches zero as $x_0 \to \infty$.

4.1 Approximations

4.1.1 First approximation: The first approximation is that N is so large that a comparison of TCP packets triggered by a packet arrival never yields a match. This means that, once in the queue, a TCP packet will never be dropped. Furthermore, since the TCP sources are identical, those quantities all have the same value, and hence we will refer to flow 1 as the generic TCP flow.

More importantly, this provides a simple relation between queueing delay, throughput and backlog. We assume that $x_1(1-p_1) \leq x_0(1-p_0) \leq Nx_1(1-p_1)$. The queueing delay is τ. The number of TCP packets in

the buffer is $b_0 + Nb_1 = b$. Then Little's Theorem implies

$$\tau = \frac{b_0 + Nb_1}{x_0(1-p_0) + Nx_1(1-p_1)} \geq \frac{b}{(N+1)x_0(1-p_0)}$$
$$\geq \frac{b}{(N+1)x_0} \quad (\because 1 - p_0 \leq 1) \tag{6}$$

$$\frac{x_0 h_0^0}{c} = \frac{x_0 h_0^0}{x_0(1-p_0) + Nx_1(1-p_1)} \leq \frac{x_0 h_0^0}{2x_0(1-p_0)}$$
$$\simeq \frac{x_0 h_0^0}{2x_0 h_0^0} = \frac{1}{2} \quad (\text{assuming } h_0^0 \simeq h_0^0 S_0) \tag{7}$$

These are the key equations for throughput analysis.

4.1.2 Second approximation: The second approximation is that the total backlog b is large enough so that

$$1 - \frac{1}{b} \to 1, \qquad \left(1 - \frac{1}{b}\right)^b \simeq e^{-1} \tag{8}$$

and

$$1 + 2\frac{1 - b_0}{b(b-1)} = \frac{b^2 - b + 2 - 2b_0}{b^2 - b} \to 1$$

Using (6) and (8) to rewrite S_0, which is from (2), as

$$S_0 = \left(1 - \frac{1}{b}\right)^{\tau x_0} \left(1 + 2\frac{1 - b_0}{b(b-1)}\right)^{\tau N x_1}$$
$$\leq \left(1 - \frac{1}{b}\right)^{\tau x_0} \left(1 + 2\frac{1 - b_0}{b(b-1)}\right)^{\tau N x_0}$$
$$\leq \left(1 - \frac{1}{b}\right)^{(N+1)\tau x_0} \leq \left(1 - \frac{1}{b}\right)^b = e^{-1} \tag{9}$$

4.2 Maximum and asymptotic throughput

Recall that $\mu = x_0(1 - p_0)/c$ denotes the UDP throughput share, and let $\mu_0^* = \max\mu_0$ denote the maximum achievable UDP share. We now estimate μ_0^* and prove that μ_0 approaches 0 asymptotically as $x_0 \to \infty$.

Theorem 1: The maximum UDP bandwidth share is $\mu_0^* = (2e)^{-1} = 0.184$.

Proof: The UDP bandwidth share is

$$\mu_0 = \frac{x_0(1-p_0)}{c} = \frac{x_0 h_0^0 S_0}{c} \quad (\text{from (2)})$$
$$\leq \frac{x_0 h_0^0}{c} e^{-1} \quad (\text{from (9)})$$
$$\leq \frac{1}{2} e^{-1} \quad (\text{from (7)})$$
$$= \frac{1}{2e} =: \mu_0^* \tag{10}$$ □

The solution is accurately validated with *ns*2 simulations; see Section 5.2.

The next result says that, as UDP rate x_0 grows without bound, its bandwidth share μ_0 drops to zero.

Theorem 2: As $x_0 \to \infty$, $\mu_0 \to 0$.

Proof: $b_0 = \mu_0 b = x_0(1 - p_0)b/c$ and $b_0 = b - Nb_1$, so we could obtain $1 - p_0 = c(b - Nb_1)/(x_0 b)$. Since b, c, Nb_1 are constants, we will obtain $1 - p_0 = c(b - Nb_1)/(x_0 b) \to 0$ as $x_0 \to \infty$. Therefore we will have $p_0 \to 1$. This means that, the UDP packets are almost dropped by WARD before they get through. Hence, $(b - Nb_1) \to 0$. Now, due to $(b - Nb_1) \to 0$, we have $\mu_0 = x_0(1 - p_0)/c = (b - Nb_1)/b \to 0$. □

5 Simulation results

This section presents simulation results of WARD's performance in penalising misbehaving flows and thus approximating fair bandwidth allocation. We use the RED, CHOKe and Drop Tail schemes for comparison. The mechanisms that require full perflow state information are not included here because of the practical limitations of scalability, especially in high-speed routers that usually handle thousands of flows. The simulations range over a spectrum of network configurations and traffic mixes. Since the space is limited, only some simulations validating the analysis solution are shown in this section. Other simulation results and details, such as TCP sources with different versions and TCP sources with different round-trip times, can be found in [14].

5.1 Simulation setup

We use the network simulator *ns2* [17] 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 routers R1 and R2. The link, with capacity of 1 Mbps, is shared by m TCP (with one version) and n UDP 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, so that the delay experienced by a packet is mainly caused by the buffer delay rather than the transmission delay [1]. The maximum window size of TCP is set to 500 segment such that it does not 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 constant bit rate (CBR) of γ Kbps, where γ is variable. The size of all packets are set to 1 KB.

5.2 Single unresponsive flow

To study how much bandwidth a single nonadaptive UDP source can obtain when the routers use different queue

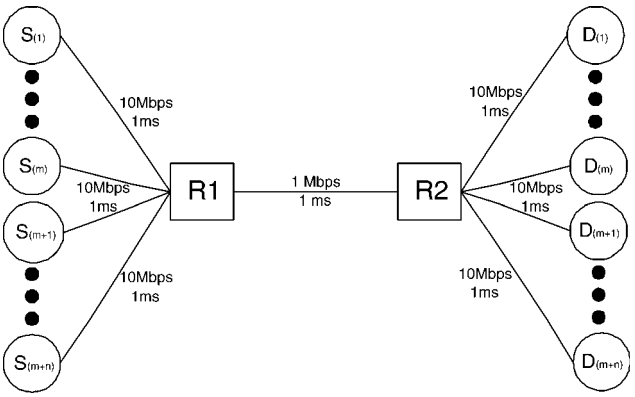

Fig. 2 *Simulation topology*

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 the numerical solution of TCP/WARD model, which is described in Section 3.2, are plotted in Fig. 3. In addition, the ideal fair share throughput is 30.3 Kbps. 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 [7], we set the maximum threshold to be twice the minimum threshold. In addition, with no partiality, the physical buffer size of each queue management is fixed at 300 packets. From Fig. 4, we could clearly see that the Drop Tail and RED gateways do not discriminate against unresponsive flows.

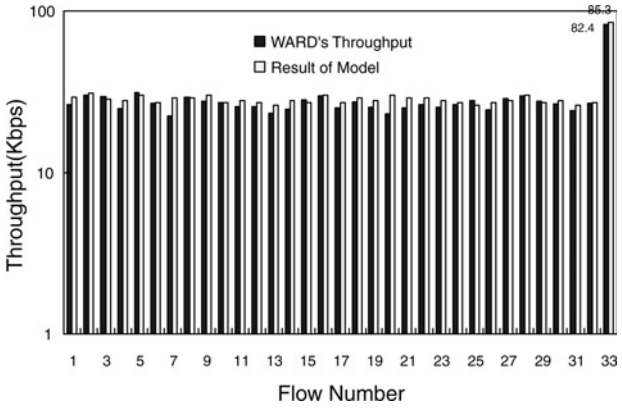

Fig. 3 *Throughput per flow*

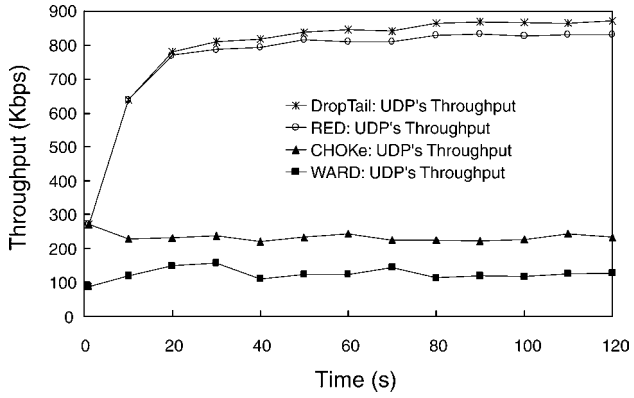

Fig. 4 *UDP throughput comparison*

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, 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 investigate WARD's performance under different traffic load conditions. The simulation results are summarised in Figs. 5 and 6, where the UDP's and average TCP's throughput versus the UDP flow arrival rate are plotted. The drop percentage of the UDP flow is also shown in Fig. 5. From Fig. 5, when $x_0 = 600$ Kbps, we could see that the maximum UDP bandwidth share is 0.1845 ($=184.5$ Kbps$/1$ Mbps), which closes to the numerical value ($(2e)^{-1}$) of (11). Moreover, from computing $x_0(1 - p_0) = (2e)^{-1}c$ with $x_0 = 600$ Kbps and $c = 1$ Mbps, we could obtain $p_0 \simeq 69\%$. This value is near to the simulation result with same parameters 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

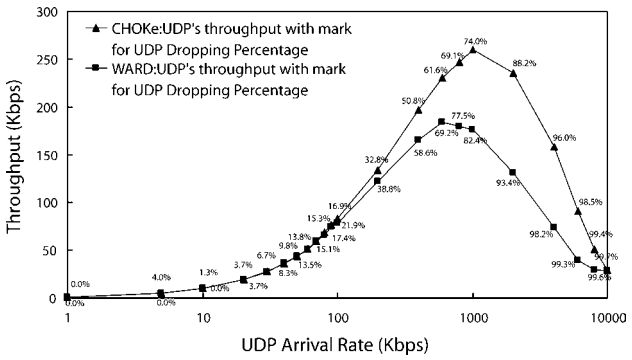

Fig. 5 *Performance under different traffic load*

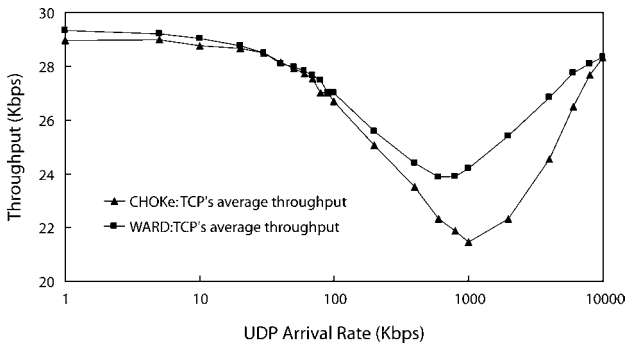

Fig. 6 *Average TCP throughput under different traffic load*

Table 1: Comparisons of WARD with different traffic load

Traffic load	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	–
TCP_{avg}	29.30	27.35	26.51	25.63	24.71	24.50	24.55	23.90	24.03	24.3	24.10	–
UDP_{thr}	–	64.60	95.26	129.06	155.26	160.93	163.33	182.00	179.93	170.00	179.26	–
P_{drop} (%)	–	32.7	41.0	51.4	63.1	68.2	72.3	71.9	78.1	80.0	81.3	–

Table 2: Performance of WARD with different buffer sizes

Buffer size	66	132	200	264	330	–
TCP_{avg}	28.87	28.61	27.02	25.15	24.2	–
UDP_{thr}	65.26	82.46	131.33	193.8	225.53	–
P_{drop} (%)	96.7	95.9	93.4	90.3	88.7	–

Table 3: 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

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 do not compare WARD with Drop Tail and RED here because the comparisons of CHOKe, RED and Drop Tail is already shown in [1].

Now, UDP source uses variable bit rate with Pareto model and shape being 1.5 to generate the packets. In addition, UDP source is an ON–OFF source. During ON periods, UDP sends data at 2 Mbps. The average throughput of UDP flow is from 0 Kbps to 1 Mbps, which is the link capacity. The simulation results are shown in Table 1, where the traffic load is defined as the average throughput of UDP flow divided by 1 Mbps, TCP_{avg} is average TCP throughput (Kbps), UDP_{thr} is UDP throughput (Kbps), and P_{drop} is drop percentage of the UDP flow, respectively. No matter what traffic load is, WARD protects TCP flows well as the best it can.

With the UDP sending packets at CBR of 2 Mbps, we vary the buffer sise from 66 (2 times 33 flows) to 330 (10 times 33 flows) to study WARD's performance. The details of different conditions are shown in Table 2, where TCP_{avg} is average TCP throughput (Kbps), UDP_{thr} is UDP throughput (Kbps), and P_{drop} is drop percentage of the UDP flow, respectively. Although the average TCP flow's throughput is still close to the fair share throughput, however, the UDP throughput increases a lot relatively. The reason is that, when the buffer size becomes larger, more UDP packets will be queued in the buffer even the dropping probability of UDP packets increases.

5.3 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 do not 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 3, 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 3, 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.

5.4 Web-mixed experiments

Fig. 7 shows the dropping rate of web packets of Drop Tail, RED, CHOKe and WARD for the web-mixed experiments with different web traffic load and FTP flows. In addition, the dropping rate of web packets is counted from the number of dropping web packets divided by the number of dropping packets. From this figure, we could see that when the web traffic load is lower than 93%, WARD drops less web packets than other three mechanisms do. In other words, short-lived TCP connections may have better protections from long-lived TCP

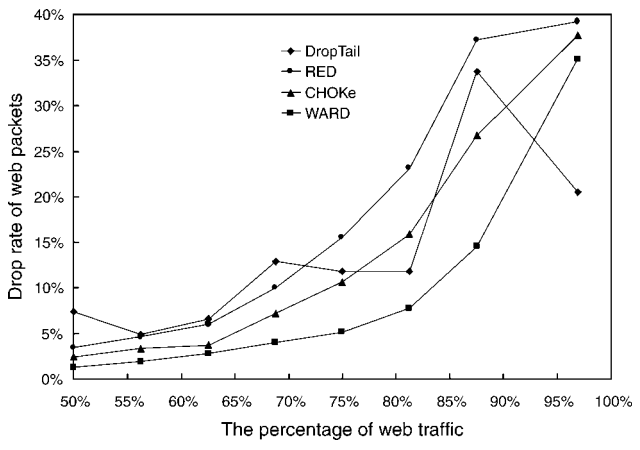

Fig. 7 *Dropping rate of web packets under different web traffic load*

flows with WARD while the web traffic load is lower than 93%. On the other hand, when the web traffic load is higher than 93%, only the performance of Drop Tail is better than that of WARD. Maybe this is because the probability of choosing packets from the same web source is growing. In order to further improve the performance of WARD with high web traffic load, the scheme will need to be fine-tuned in the future.

6 Conclusions

In this article, we introduce a packet dropping scheme, called WARD. It aims to approximate fair queueing at a minimal implementation cost. We also develop the feedback equilibrium of TCP/WARD model. We prove that as UDP input rate increases, its bandwidth peaks at $(2e)^{-1} = 0.184$ when UDP input rate is slightly larger than link capacity, and drops to zero as UDP input rate tends to infinity. Simulations demonstrate that it works well in protecting congestion-sensitive flows from congestion-insensitive or congestion-causing flows. Our model applies only to the equilibrium behaviour of TCP/WARD which presumes an asymptomatically stable system. It is restricted to the simple case of homogeneous TCP flows, a single UDP flow, a single drop candidate, at a single bottleneck link. It would be interesting to extend the analysis to a more general setting. Therefore 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. Also, more analysis and simulation of WARD with short-lived TCP flows such as Web traffic will be discussed in our future work.

7 References

1 Pan, R., Prabhakar, B., and Psounis, K.: 'CHOKe: A stateless active queue management scheme for approximating fair bandwidth allocation'. IEEE INFOCOM'2000, March 2000, vol. 2, pp. 942–951

2 Brakmo, L.S., and Peterson, L.L.: 'TCP Vegas: end to end congestion avoidance on a global internet', *IEEE J. Select. Areas Commun.*, 1995, **13**, pp. 1465–1480

3 Jacobson, V.: 'Modified TCP congestion avoidance algorithm'. Mailing list, end-to-end-interest, April 1990

4 Ait-Hellal, O., and Altman, E.: 'Analysis of TCP Vegas and TCP Reno'. IEEE ICC'97, June 1997, vol. 1, pp. 495–499

5 Mo, J., La, R.J., Anantharam, V., and Walrand, J.: 'Analysis and comparison of TCP Reno and Vegas'. IEEE INFOCOM'99, March 1999, vol. 3, pp. 1556–1563

6 Lai, Y.C., and Yao, C.L.: 'Performance comparison between TCP Reno and TCP Vegas'. IEEE ICPADS'2000, July 2000, pp. 61–66

7 Floyd, S., and Jacobson, V.: 'Random early detection gateways for congestion avoidance', *IEEE/ACM Trans. Networking*, 1993, **1**, (4), pp. 397–413

8 Lin, D., and Morris, R.: 'Dynamics of random early detection'. ACM SIGCOMM'97, September 1997, pp. 127–137

9 Anjum, F.M., and Tassiulas, L.: 'Fair bandwidth sharing among adaptive and non-adaptive flows in the internet'. IEEE INFOCOM'99, March 1999, pp. 1412–1420

10 Ott, T.J., Lakshman, T.V., and Wong, L.H.: 'SRED: Stabilised RED'. IEEE INFOCOM'99, 1999, vol. 3, pp. 1346–1355

11 Feng, W., Kandlur, D.D., Saha, D., and Shin, K.G.: 'Stochastic fair blue: A queue management algorithm for enforcing fairness'. IEEE INFOCOM'2001, 2001, pp. 1520–1529

12 Albuquerque, C., Vickers, B.J., and Suda, T.: 'Network border patrol: Preventing congestion collapse and promoting fairness in the internet', *IEEE/ACM Trans. Networking*, 2004, **12**, (1), pp. 173–186

13 Mahajan, R., Floyd, S., and Wetherall, D.: 'Controlling high-bandwidth flows at the congested router'. IEEE ICNP'2001, November 2001, pp. 192–201

14 Ho, C.-Y., Chan, Y.-C., and Chen, Y.-C.: 'WARD: A TCP-friendly stateless AQM scheme'. *IET Commun*. (accepted for publication)

15 Tang, A., Wang, J., and Low, S.H.: 'Understanding CHOKe: Throughput and spatial characteristics', *IEEE/ACM Trans. Networking*, 2004, **12**, (4), pp. 694–707

16 Nelder, J.A., and Mead, R.: 'A simplex method for function minimisation', *Comput. J.*, 1965, **7**, pp. 308–313

17 http://www.isi.edu/nsnam/ns/