TCP Westwood and Easy RED to Improve Fairness in High-Speed Networks

Luigi Alfredo Grieco, Saverio Mascolo

Dipartimento di Elettrotecnica ed Elettronica, Politecnico di Bari, Italy

To appear at *Seventh International Workshop on Protocols For High-Speed Networks* (PfHSN'2002), April 22 - 24, 2002 Berlin, Germany.
Sponsored by IFIP TC6 WG6.2, IEEE Comsoc TC on Gigabit Networking

Abstract. TCP Westwood (TCPW) is a sender-side only modification of TCP Reno congestion control, which exploits end-to-end bandwidth estimation to properly set the values of *slow-start threshold* and *congestion window* after a congestion episode. This paper aims at showing via both mathematical modeling and extensive simulations that TCPW significantly improves fair sharing of high-speed networks capacity and that TCPW is friendly to TCP Reno. Moreover, we propose EASY RED, which is a simple Active Queue management (AQM) scheme that improves fair sharing of network capacity especially over high-speed networks. Simulation results show that TCP Westwood provides a remarkable Jain's fairness index increment up to 200% with respect to TCP Reno and confirm that TCPW is friendly to TCP Reno. Finally, simulations show that Easy RED improves fairness of Reno connections more than RED, whereas the improvement in the case of Westwood connections is much smaller since Westwood already exhibits a fairer behavior by itself.

1. Introduction

Packet switching networks require sophisticated mechanism of flow and congestion control in order to share resources and avoid congestion phenomena. Congestion control functions were introduced into the TCP in 1988 and have been of crucial importance in preventing congestion collapse [1],[2],[9]. However, while end-to-end TCP congestion control [4],[5] can ensure that network capacity is not exceeded, it cannot insure fair sharing of that capacity [1]. In this paper we investigate via both mathematical analysis and computer simulations the issue of fairness in high-speed networks when Westwo od TCP is implemented at the sender side. Moreover we propose a simpler version of RED, called EASY RED and we investigate how it interacts with Reno and Westwood TCP.
TCP Westwood (TCPW) performs an end-to-end estimate of the bandwidth available along a TCP connection to adaptively set the control windows after congestion [3]. The rationale of TCPW is simple: in contrast with TCP Reno, which implements a multiplicative decrease algorithm after congestion, TCPW sets a slow start threshold and a congestion window which are consistent with the effective bandwidth used at the time congestion is experienced.

In this paper, TCPW employs a bandwidth estimation algorithm that is slightly different from the one used in [3] in order to avoid bandwidth overestimates due ACK compression [6],[11].
EASY RED is a simpler variant of RED that does not average the queue length but relates the drop probability to the instantaneous queue level. In fact, the purpose of early discard is to signal congestion to the sender as soon as possible. In contrast averaging the queue introduces delay, which is harmful for congestion control purposes. EASY RED has only two parameters to be set: (1) the minimum threshold (min_th) and (2) the constant drop probability $pdrop$ when the instantaneous queue length is greater or equal to min_th.
A main contribution of this paper is a mathematical model that proves stability, fairness and friendliness of TCP Westwood with respect to Reno. In particular, the model shows that the mean throughput of TCP Westwood is function of the available bandwidth and is less sensitive to round trip time than Reno throughput, that is, Westwood improves fair sharing of network capacity among flows with different RTTs. Moreover, the model highlights that the throughput of TCPW depends on the inverse of the square root of the drop probability just like the throughput of Reno [18],[25], that is, TCPW is friendly to TCP Reno.
Simulation results using Westwood show a remarkable increment of the Jain fairness index up to 200% with respect to Reno over a 100Mbps wired network. Also they confirm the theoretical model by showing that TCPW is completely friendly to Reno. Performance improvements are also shown when AQM mechanisms are used. Simulations show that EASY RED improves fairness of Reno connections more than RED, whereas the improvement in the case of Westwood connections is much smaller since Westwood already exhibits a fairer behavior by itself.
The paper is organized as follows: in Section 2 a mathematical model of TCP Westwood is developed; in Section 3, Active Queue Management algorithms are described and Easy RED is proposed; in Section 4, simulation results with many Reno or Westwood TCP connections having different RTTs and sharing a FIFO bottleneck queue implementing RED, Gentle RED, EASY RED or no AQM policy are reported. Finally, Section 5 draws the conclusions.

2.TCP Westwood

A detailed description of TCP Westwood (TCPW) is reported in [3]. In this section, we briefly resume TCPW and we introduce a new mechanism to estimate the available bandwidth. Later we develop a mathematical model of Westwood and analyze fairness and friendliness of Westwood in comparison with Reno by using their respective throughput equation models.

2.1 A Description of TCP Westwood

A TCP connection is characterized by the following variables:

- *cwnd*: Congestion Window

- *ssthresh*: Slow Start Threshold
- *RTT*: Round Trip Time of the connection
- RTT_{min}: Minimum Round Trip Time measured by the sender
- *seg_size*: Size of the delivered segments

The main idea of TCP Westwood is to perform an end-to-end estimate of the bandwidth B available along a TCP connection by measuring and low-pass filtering the rate of returning ACKs. For available bandwidth we mean the measurement of the actual rate a connection is achieving during the data transfer. This is a much more easy task than estimating the bandwidth that is available at the beginning of a TCP connection [12],[14],[15],[16]. The bandwidth estimate is then used to properly set the congestion window and the slow-start threshold after a congestion episode as described below:

```
a)  When 3 DUPACKs are received by the sender:
    ssthresh = $(B*RTT_{min})$/seg_size;
    cwnd = ssthresh;

b)  When coarse timeout expires:
    ssthresh = $(B*RTT_{min})$/seg_size;
    cwnd = 1;

c)  When ACKs are successfully received:
    cwnd is increased following the Reno algorithm.
```

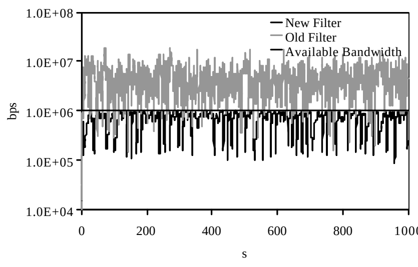
As it has been pointed out in [1],[2],[26], the stability of the Internet does not require that flows reduce their sending rate by half in response to a single congestion indication. In particular, the prevention of congestion collapse simply requires that flows use some form of end-to-end congestion control to avoid a high sending rate in the presence of high packet drop rate. In the case of TCPW the sending rate is reduced by taking into account a measurement of the available bandwidth at the time congestion is experienced. Therefore, when in the presence of heavy congestion, this reduction can be even more drastic than a by half reduction and it can be less drastic with light congestion. This feature can clearly improves network stability and utilization in comparison with the *blind* by a half window reduction performed by Reno.

2.2 Robustness of bandwidth estimate to ACK Compression

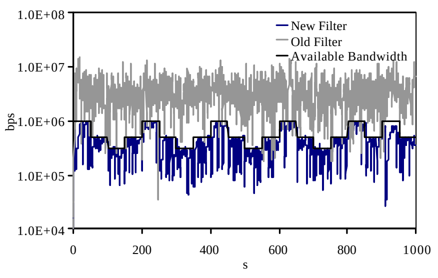
In order to fully exploit the advantages of the AIAD paradigm, it is of crucial importance to obtain a *good estimate* of the bandwidth that is available when congestion is experienced. Due to delays and ACKs compression, the flow of returning ACKs must be low-pass filtered in a proper way [11],[17]. In [3], a sample of available bandwidth $b_k = d_k / (t_k - t_{k-1}) = d_k / \Delta_k$ is computed every time t_k the sender receives an ACK, where the amount d_k of data acknowledged by an ACK is determined by a proper counting procedure that takes into account delayed ACKs, duplicate and cumulative ACKs. Samples b_k are low-pass filtered using the time-varying filter

$\hat{b}_k = \frac{2t_f - \Delta_k}{2t_f + \Delta_k}\hat{b}_{k-1} + \Delta_k \frac{b_k + b_{k-1}}{2t_f + \Delta_k}$, where t_f is the filter time constant (a typical value is $t_f = 0.5$ s). In this paper, we propose a slightly modified version of the filter used in [3] since that filter overestimates the available bandwidth when in the presence of ACK compression [6]. To overcome this problem, we compute bandwidth samples every *RTT*. More precisely, we count all data d_k acknowledged during the last *RTT* and compute the bandwidth sample $b_k = d_k / \Delta_k$, where Δ_k is the last *RTT*. Moreover, in order to comply with the Nyquist sampling theorem when $\Delta_k > t_f / 4$, we interpolate and re-sample using $N = int(4 \cdot RTT / t_f)$ [1] virtual samples b_k arriving with the interarrival time $\Delta_k = t_f / 4$.

In order to test the robustness of the new filter with respect to ACK compression, we simulate a single bottleneck scenario shared by one TCP and one UDP connection via FIFO queuing. The bottleneck link capacity is 1Mbps. In order to provoke ACK compression, 10 TCP Reno connections sending data along the reverse path are considered. Segment size is 1500 Bytes long, queue size is 20 segments and the simulation lasts 1000s. Fig. 1(a) shows the bandwidth estimate obtained using the old and the new filter when the UDP source is turned off. The tick lines marks the available bandwidth that is 1Mbps. Fig. 1(a) shows that the old filter overestimates the bandwidth ten times, whereas the new one nicely tracks the available bandwidth. Fig. 1(b) shows the bandwidth estimate obtained when the UDP sources is active. The tick line marks the bandwidth that is left available by the UDP traffic. Also in this case the new filter tracks the available bandwidth whereas the old one overestimates up to 10 times the available bandwidth.



(a)



(b)

Fig. 1. Bandwidth estimates: (a) without UDP traffic; (b) with coexisting UDP traffic

2.3 A Mathematical Model of TCP Westwood

In this section a mathematical model of the *Additive Increase Adaptive Decrease* mechanism introduced by Westwood is developed. To derive the model, we follow arguments similar to the ones developed in the excellent paper by Kelly [18]. For the sake of simplicity, we do not model the behavior after a timeout.

[1] $int(\cdot)$ stands for the integer part of $(\cdot)$

Theorem 1. Consider a TCP flow that is controlled by the Westwood algorithm. Suppose that the drop probability of a segment is *p*, the bandwidth available for the flow is *B*, the mean round trip time is *RTT* and the minimum round trip time is RTT_{min}. By letting r^W be the steady state mean throughput of the flow, it holds:

$$r^W = \frac{B \cdot RTT_{\min}}{2 \cdot RTT} + \sqrt{\left(\frac{B \cdot RTT_{\min}}{2 \cdot RTT}\right)^2 + \frac{1-p}{p \cdot (RTT)^2}} . \qquad (1)$$

Proof. The congestion window is updated upon ACK reception. Each time an ACK is received back by the sender the *cwnd* is increased by 1/*cwnd*, whereas after a segment loss the congestion window is set equal to $B{\cdot}RTT_{min}$ so that the change in *cwnd* is $B \cdot RTT_{\min} - cwnd$. Since the segment drop probability is *p*, it follows that the expected increment of the congestion window *cwnd* per update step is:

$$E[\Delta cwnd] = \frac{1-p}{cwnd} + (B \cdot RTT_{\min} - cwnd) \cdot p . \qquad (2)$$

Since the time between update steps is about $\frac{RTT}{cwnd}$, by recalling Eq. (2), the expected change in the rate *r* per unit time is approximately:

$$\frac{\partial r(t)}{\partial t} = \frac{1-p}{RTT^2} + p \cdot r(t) \cdot \frac{B \cdot RTT_{\min}}{RTT} - r^2(t) \cdot p . \qquad (3)$$

Eq. (3) is a separable variable differential equation. By separating variables, Eq. (3) can be written as:

$$\frac{\partial r(t)}{r^2(t) - r(t) \cdot \frac{B \cdot RTT_{\min}}{RTT} - \frac{1-p}{p \cdot RTT^2}} = -p \cdot \partial t . \qquad (4)$$

and by integrating each member the following solution can be obtained

$$r(t) = \frac{r_1 - r_2 \cdot C \cdot e^{-p \cdot t \cdot (r_1 - r_2)}}{1 - C \cdot e^{-p \cdot t \cdot (r_1 - r_2)}} .$$

Where $r_{1,2} = \frac{B \cdot RTT_{\min}}{2 \cdot RTT} \pm \sqrt{\left(\frac{B \cdot RTT_{\min}}{2 \cdot RTT}\right)^2 + \frac{1-p}{p \cdot (RTT)^2}}$ are the roots of the equation $r^2 - r \cdot \frac{B \cdot RTT_{\min}}{RTT} - \frac{1-p}{p \cdot RTT^2} = 0$ and *C* depends on the initial conditions.

The steady state throughput of the Westwood algorithm is then

$$r^W = \lim_{t \to \infty} r(t) = \frac{B \cdot RTT_{\min}}{2 \cdot RTT} + \sqrt{\left(\frac{B \cdot RTT_{\min}}{2 \cdot RTT}\right)^2 + \frac{1-p}{p \cdot (RTT)^2}}$$

It is easy to show the following corollary.

Corollary 1. The Westwood control algorithm is stable, that is

$$r^W \leq B \,. \tag{5}$$

Proof. From Eq. (1) we can argue that r^W can never be greater than *B*. In fact, by contradiction, let us assume that $r^W > B$. This assumption would lead to congestion collapse so that the drop probability p would increase up to 1. Thus, from Eq. (1) it would result $r^W = B\frac{RTT_{\min}}{RTT}$. Since under congestion $RTT_{\min} < RTT$, it would result $r^W < B$, which would contradict the assumption. Therefore, we can conclude that $r^W \leq B$.

Now, by noting that the end-to-end bandwidth estimation algorithm described above provides a value that is well approximated by *cwnd/RTT*, it is possible to mathematically derive the throughput of Westwood when the bandwidth estimation algorithm described in this paper is employed.

Theorem 2. The steady state throughput of Westwood using the bandwidth estimate *B=cwnd/RTT* is

$$r^{West} = \frac{1}{\sqrt{RTT \cdot T_q}} \cdot \sqrt{\frac{(1-p)}{p}} \,. \tag{6}$$

Where $T_q = RTT—RTT_{min}$ is the mean queuing time experienced by the segments of the connection.

Proof. By assuming the following estimate of the available bandwidth

$$B = cwnd / RTT = r(t) \tag{7}$$

and by substituting Eq. (7) into Eq. (3), the following differential equation is obtained:

$$\frac{\partial r(t)}{\partial t} = \frac{1-p}{RTT^2} + p \cdot r^2(t) \cdot \frac{RTT_{\min}}{RTT} - r^2(t) \cdot p \,. \tag{8}$$

By separating variables, Eq. (8) can be written as:

$$\frac{\partial r(t)}{r^2(t) - \frac{1-p}{p \cdot T_q \cdot RTT}} = -p \cdot \frac{T_q}{RTT} \partial t \,. \tag{9}$$

and integrated as:

$$r(t)=\sqrt{\frac{1-p}{p\cdot RTT\cdot T_q}}\cdot\frac{1+C\cdot e^{-2\cdot p\cdot t\cdot\frac{T_q}{RTT}\cdot\sqrt{\frac{1-p}{p\cdot RTT\cdot T_q}}}}{1-C\cdot e^{-2\cdot p\cdot t\cdot\frac{T_q}{RTT}\cdot\sqrt{\frac{1-p}{p\cdot RTT\cdot T_q}}}}\,. \qquad \textbf{(10)}$$

Where C depends on the initial conditions. The steady state throughput (6) is then obtained for $t\to\infty$.

2.3 Fairness and Friendliness Evaluation

Kelly derives the following steady state mean throughput of Reno TCP [18]:

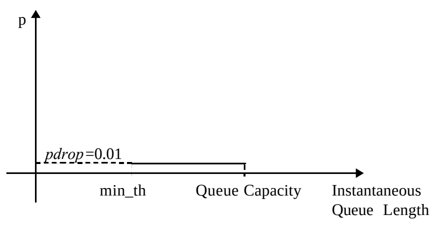
$$r_R=\frac{1}{RTT}\cdot\sqrt{\frac{2\cdot(1-p)}{p}}\,. \qquad \textbf{(11)}$$

With reference to friendliness, by comparing (6) and (11) it can be noted that both throughputs of Westwood and Reno depend on $1/\sqrt{p}$, that is Westwood and Reno are friendly to each other. Moreover, Eq. (6) shows that flows with different *RTTs* and going through the same bottleneck, experience the same mean queuing time $T_{q.}$ Therefore, the throughput of Westwood depends on round trip time as $1/\sqrt{RTT}$ whereas throughput of Reno as $1/RTT$, that is, Westwood increases fair sharing of network capacity between flows with different RTTs.

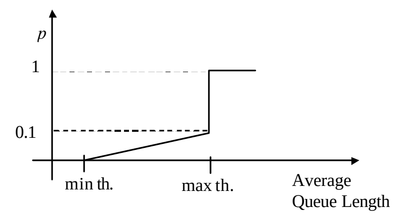
3. AQM Policies and Easy RED

The idea behind Active Queue Management (AQM) is to discard a packet before queue overflow in according to a drop probability function. The rationale is that, by discarding a packet before queue overflow, a TCP sender can detect congestion earlier and react earlier.
The most know example of AQM mechanism is RED, which uses a drop probability function that increases linearly with the average queue length [19]. RED needs the tuning of four parameters that are: (1) the minimum queue threshold (*min_th*); (2) the maximum queue threshold (*max_th*); (3) the drop probability max_p when the average queue reaches the *max_th* and (4) the constant value used by the exponential filter to average the queue length. A delicate issue with RED is that it requires fine-tuning of many parameters in order to work properly. Consequently, there is considerable nervousness in the community regarding the deployment of RED [10],[20],[21],[22].
Several complex variants of RED have been proposed in order to obtain algorithms less sensitive to parameter tuning. In [29], stabilized RED (SRED) is proposed, which aims at stabilizing buffer occupation by estimating the number of active connections in

order to set the drop probability as a function of the number of the active flows and of the instantaneous queue length. In [28], Flow RED (FRED) is proposed which uses per-active-flow accounting to impose on each flow a loss rate that depends on the flow's buffer use. FRED employs the same drop probability function of RED; furthermore, it maintains minimum and maximum limits on the packets that a flow may have in the queue and uses a more aggressive drop against the flows that violates the maximum bound. In [27] and [32] schemes to auto tune RED parameters are proposed. These schemes essentially increase the max_p parameter when the average queue length exceeds a fixed target and decrease max_p when the average queue length falls below the target level. The Balanced RED algorithm, which tries to regulate the bandwidth assigned to each flow by doing per flow accounting, is proposed in [30]. BRED stores the per flow buffer level and for each incoming packet it computes the drop probability as a function of the buffer level of the flow to which the packet belongs. Finally Dynamic RED [31] proposes to discard packets with a load dependent probability. In particular DRED continuously update the drop probability by employing an integral controller with a gain in cascade. The input of the controller is the difference between the average queue length and the target buffer level whereas the output is the drop probability.
In this section, we introduce a simpler variant of RED that we call EASY RED. We show that EASY RED improves fairness and that it is not sensitive to parameters tuning. EASY RED does not average the queue length but it relates the drop probability to the instantaneous queue level. In fact, the purpose of early discard is to signal congestion to the sender as soon as possible. In contrast to this, the queue average of RED introduces delay, which is harmful for congestion control purposes. In control terms, averaging means the introduction of an extra pole in the control loop [20],[22].
EASY RED has only two parameters to be tuned: (1) the *min_th* and (2) the constant drop probability when the instantaneous queue length is greater or equal to *min_th.* Fig. 2 shows the dropping profile of EASY RED and RED. EASY RED has a flat dropping probability that is function of the *instantaneous queue length*, whereas RED has a linearly increasing drop probability that jumps to one when the *average queue length* reaches the *max_th* [23]. The gentle variant of RED eliminates the jump to one using another linear piece of curve [24].



(a)



(b)

Fig. 2. Drop Probability. (a) Easy RED; (b) RED

4. Performance Evaluation

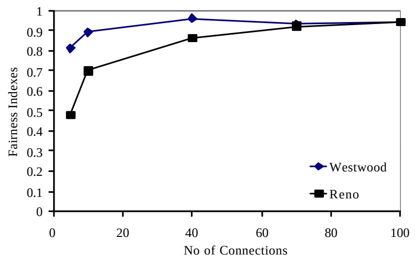
In this section, we test TCPW using the ns-2 simulator [7] and we validate results obtained in Sec. 2, which are: (1) TCP Westwood improves fairness; (2) TCP Westwood is friendly to Reno. Moreover we test the behavior of EASY RED.
In order to evaluate the performance of TCP Westwood we consider N greedy connections sharing a FIFO bottleneck, with N=5,10,40,70,100 and RTT ranging uniformly from (250/N)ms to 250ms. The sources transmit data during a period of 30 seconds. The segment size is 1500 Bytes long. The bottleneck link bandwidth is set equal to 10Mbps or 100Mbps and the bottleneck queue capacity is set equal to the link capacity times the maximum round trip propagation time, that is, the bottleneck queue size is set equal to 200 and 2000 segments, respectively. Note that these settings allow a number of segments proportional to the number of flows be accommodated in the bottleneck queue so avoiding the *many flows* effect [13]. To provide a single numerical measure reflecting the fair share distribution across the various connections we use the Jain Fairness Index defined as:

$$FairnessIndex = \frac{(\sum_{i=1}^{N} b_i)^2}{N \sum_{i=1}^{N} b_i^2}$$

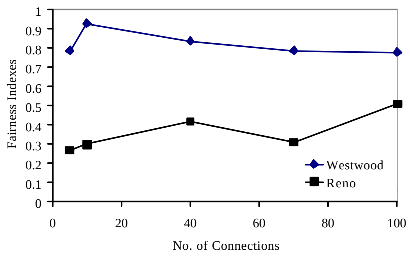
where b_i is the throughput and N is the number of connections [8].

4.1 Fairness of TCP Westwood

In this section, we compare the fairness of TCPW versus the fairness of TCP Reno without using AQM policies. Fig. 3 (a) shows the Jain fairness index as a function of the number of connections when the bottleneck capacity is 10Mbps and Fig. 3 (b) when the bottleneck capacity is 100Mbps. Fig. 3 shows that TCPW improves fairness up to 200% when bottleneck capacity is 100Mbps and up to 60% when bottleneck capacity is 10Mbps.



(a)



(b)

Fig. 3. Jain's fairness Indexes: (a) 10Mbps bottleneck link; (b) 100Mbps bottleneck link

Figs. 4(a) and (b) show the corresponding mean throughputs computed as the sum of all the throughputs of the N TCP sources sharing the bottleneck divided by N. To give a further insight, Figs 5(a) and (b) show the curves of Bytes_sent vs. time in the case of 40 connection using Reno and in the case of 40 connections using Westwood, respectively. The bottleneck is 100Mbps. Figures clearly show that Reno curves are much more spread than Westwood curves, i.e. TCPW increases fairness.

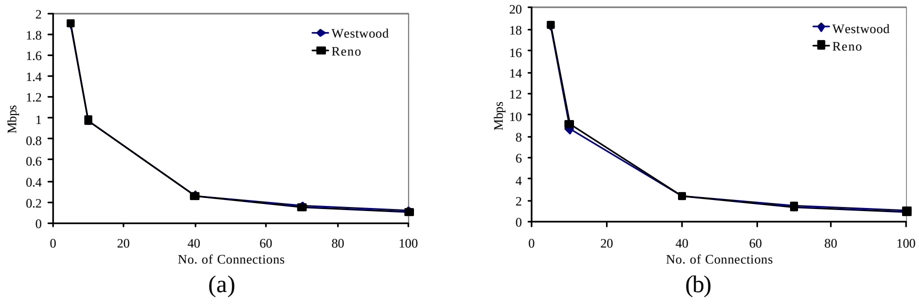


Fig. 4. Mean Throughputs: (a) 10Mbps bottleneck link; (b) 100Mbps bottleneck link

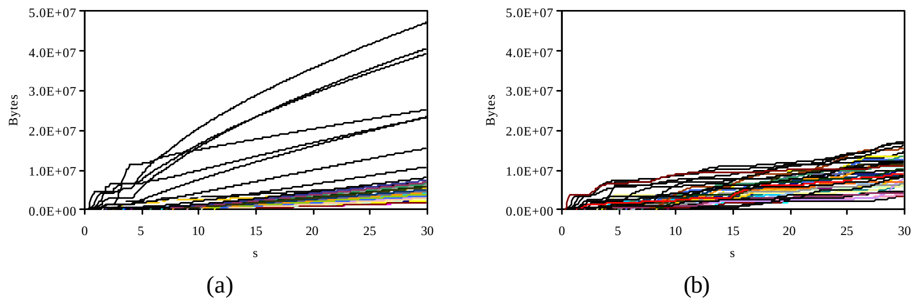


Fig. 5. Bytes sent vs. time. (a) 40 Reno connections. (b) 40 Westwood connections

4.2 Interaction with AQM Policies

In this section, we study the effect of AQM policies on the performance of TCP Reno and Westwood. Moreover, we evaluate the performance improvement when Easy RED is employed.
Fig. 6(a) shows the Jain fairness index as a function of the number of Westwood connections when the bottleneck capacity is 10Mbps and Fig. 6(b) when the bottleneck capacity is 100Mbps. Four curves are shown that refer to RED, Gentle RED, EASY RED and no AQM, i. e. drop from tail, policy. Fig. 6(a) shows that EASY RED does not change the fairness whereas RED and gentle RED reduces the fairness with respect to simple drop of tail. Fig. 6(b) shows that EASY RED improves fairness up to 12% with respect to no AQM policies whereas RED and gentle RED reduces fairness with respect no AQM. Figs. 7(a) and (b) show corresponding mean throughputs: RED and gentle RED reduces the throughput of Westwood with respect to EASY RED and drop tail.
Figs.8(a) and (b) show the Jain fairness index as a function of the number of Reno connections when the bottleneck capacity is 10Mbps and 100Mbps, respectively. Fig. 8 (a) shows that for N<40 EASY RED improves fairness up to 40% whereas RED and gentle RED reduces fairness also respect to no AQM policies for N>10. In the case of Fig. 8(b), EASY RED improves fairness up to 65% with respect to RED policy and up

to 165% with respect to drop tail. Figs. 9(a) and (b) show the corresponding mean throughputs. Note that, in the case of 100Mbps bottleneck, RED and gentle RED reduces the Reno throughput with respect to EASY RED and drop tail.
RED parameters have been set as suggested by [23]: filter constant *q_weight*=0.002, *min_th*=5, *max_th*=15, maximum drop probability = 0.1. Gentle RED parameters have been set following recommendations in [24]. EASY RED parameters have been set as follows: *pdrop*=0.01 and *min_th*=queue_capacity/3.

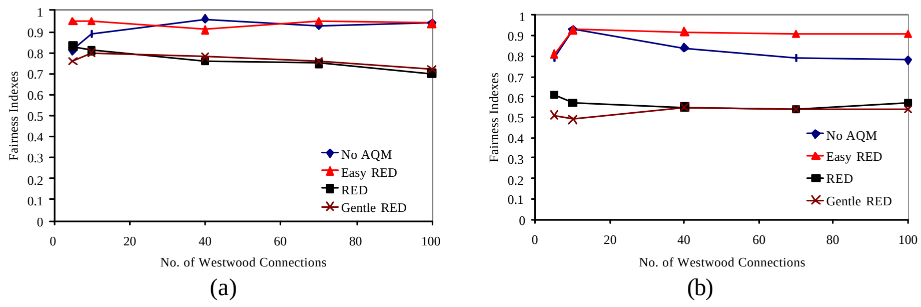


Fig. 6. Jain's fairness Indexes: (a) 10Mbps bottleneck link; (b) 100Mbps bottleneck link

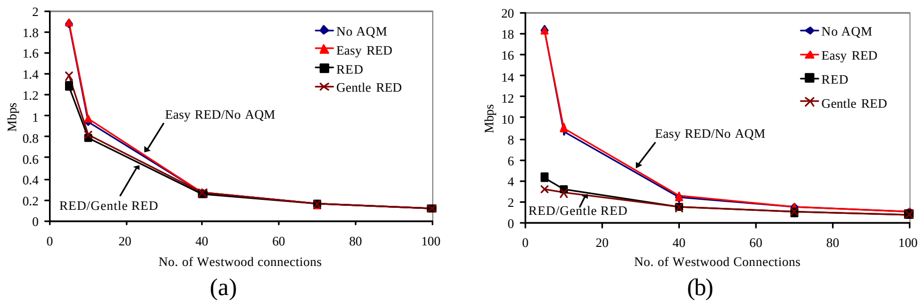


Fig. 7. Mean Throughputs: 10Mbps bottleneck link; (b) 100Mbps bottleneck link

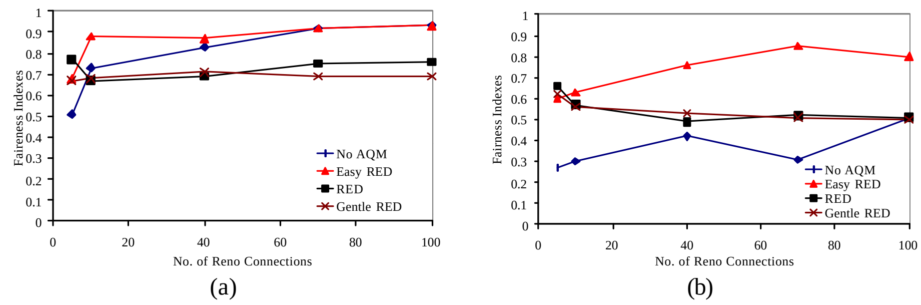
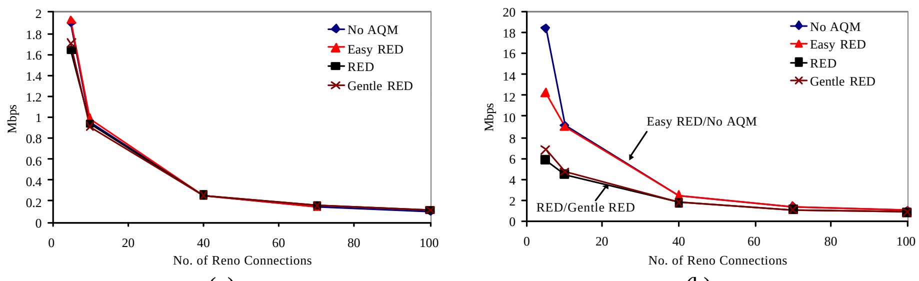


Fig. 8. Jain's fairness Indexes: 10Mbps bottleneck link; (b) 100Mbps bottleneck link



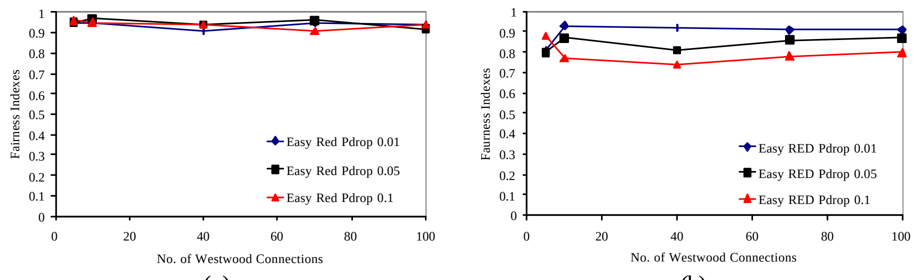
(a) (b)

Fig. 9. Mean Throughputs: (a) 10Mbps bottleneck link; (b) 100Mbps bottleneck link

4.3 Sensitivity of Easy RED to Parameters setting

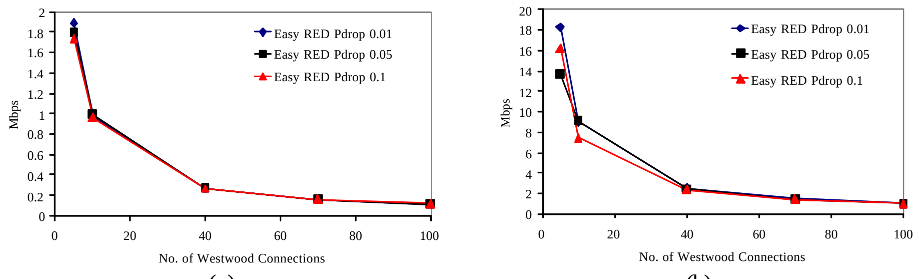
To investigate the effects that the *pdrop* parameter used in Easy RED has on the fairness and throughput, we vary *prodp* from 0.01 to 0.1 by keeping *min_th=queue_capacity*/3. Figs. 10-13 show that fairness indexes and mean throughputs of Reno and Westwood are not sensitive to *pdrop* over links with capacity of 10Mbps and 100Mbps.

To investigate the effect of the *min_th* parameter we set *min_th=queue_capacity/a* and we vary *a* from 2 to 4 by keeping *pdrop*=0.01. Figs. 14-17 show that fairness indexes and mean throughputs of Reno and Westwood are not sensitive to *min_th*.



(a) (b)

Fig. 10. Sensitivity of the Jain fairness indexes to *pdrop*: (a) 10Mbps link; (b) 100Mbps link



(a) (b)

Fig. 11. Sensitivity of mean throughputs to *pdrop*: (a) 10Mbps link; (b) 100Mbps link

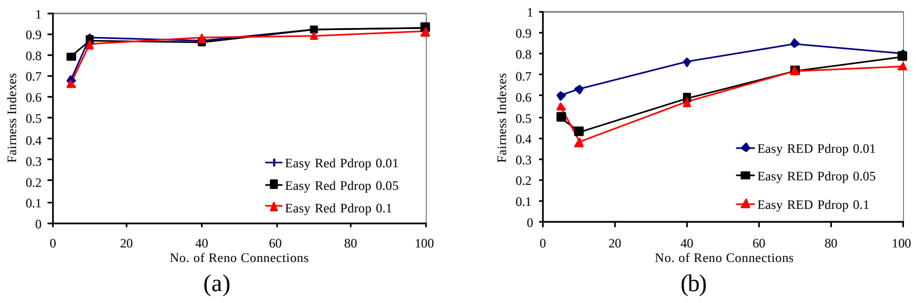


Fig. 12. Sensitivity of the Jain fairness indexes to *pdrop*: (a) 10Mbps link; (b) 100Mbps link

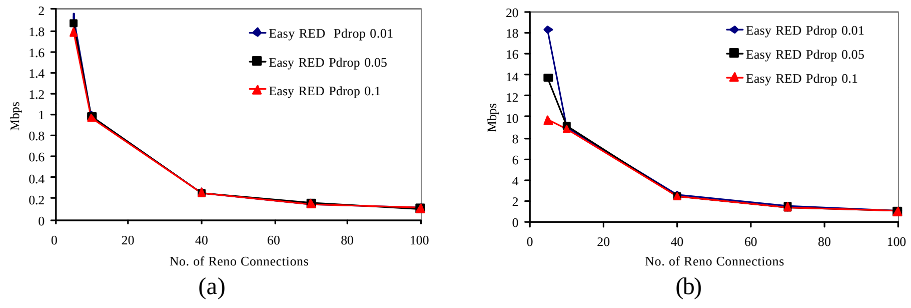


Fig. 13. Sensitivity of mean throughputs to *pdrop*: (a) 10Mbps link; (b) 100Mbps link

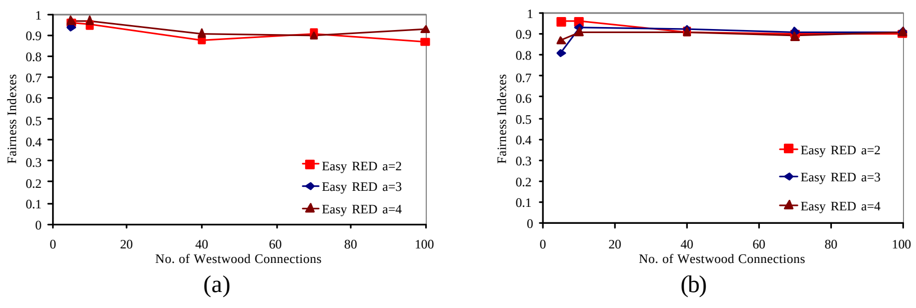


Fig. 14. Jain's fairness Indexes sensitivity to *min_th*: (a) 10Mbps link; (b) 100Mbps link

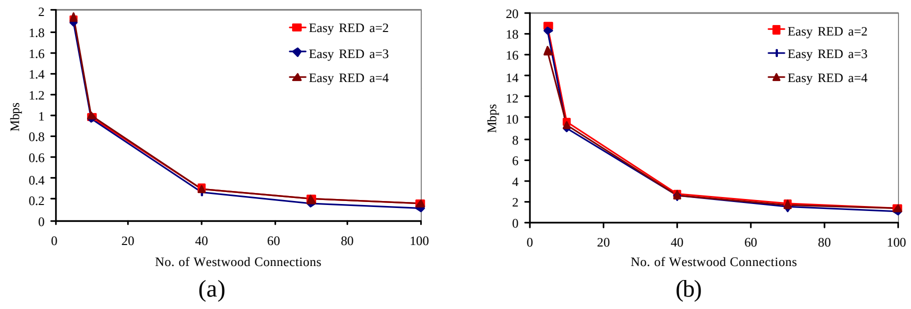


Fig. 15. Mean Throughputs sensitivity to *min_th*. (a) 10Mbps link; (b) 100Mbps link

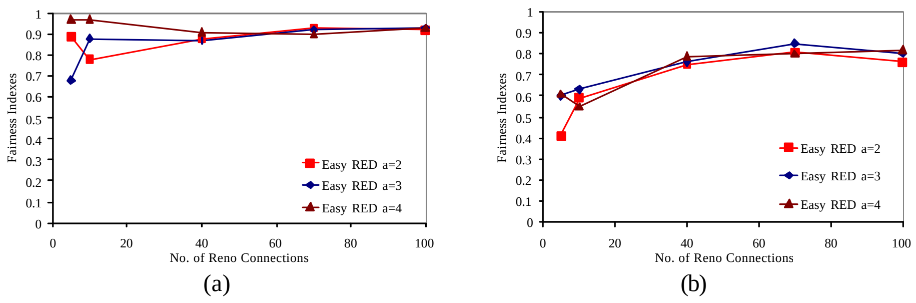


Fig. 16. Jain's fairness Indexes sensitivity to *min_th*: (a) 10Mbps link; (b) 100Mbps link

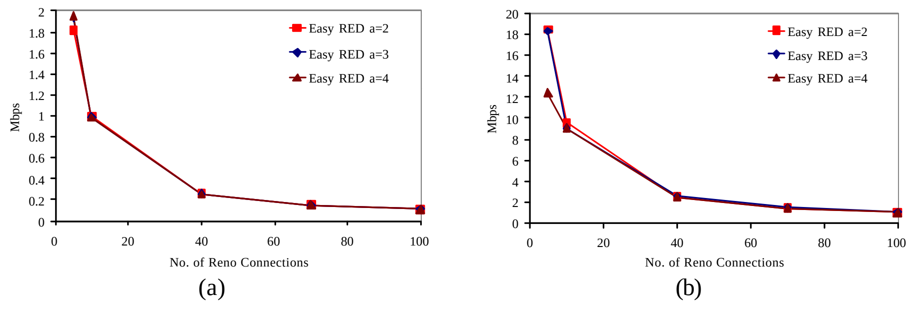


Fig. 17. Mean Throughputs sensitivity to *min_th*: (a) 10Mbps link; (b) 100Mbps link

4.4 Friendliness of TCP Westwood

Friendliness relates to how connections running different TCP flavors affect the performance of one another. In particular, we demand that a newly proposed TCP be *friendly* to the TCP versions already in use in the Internet. That is, the new TCP connections must be able to coexist with connections using existing TCP protocols while providing opportunities for all connections to progress satisfactorily. At a minimum, the new connections should not cause the starvation of the connections using the existing version of TCP.
We simulate the same scenario described in the previous sections with N=10,40,70,100 connections. The Westwood connections are mixed with the Reno connections. In particular, N/2 Reno and N/2 Westwood connections are mixed. Round trip times are spread as in the scenario described in the previous section.
Table I and Table II show fairness indexes and mean throughputs when the bottleneck link capacity is 10Mbps and 100Mbps, respectively. Results show that indexes obtained in the mixed environments are better than ones obtained with only Reno connections, especially over the 100Mbps high speed link, that is, TCPW is more than friendly to Reno.

Table 1. 10Mbps bottleneck link

Connections	Fairness Index	Mean Throughput (Mbps)
100 West	0.94	0.117
50 West 50 Reno	0.9	0.113
100 Reno	0.94	0.107
70 West	0.93	0.16
35 West 35 Reno	0.9	0.155
70 Reno	0.92	0.148
40 West	0.96	0.266
20 West 20 Reno	0.87	0.261
40 Reno	0.86	0.256
10 West	0.89	0.949
5 West 5 Reno	0.8	0.952
10 Reno	0.7	0.949

Table 2. 100Mbps bottleneck link

Connections	Fairness Index	Mean Throughput (Mbps)
100 West	0.78	1.04
50 West 50 Reno	0.64	1.02
100 Reno	0.51	0.997
70 West	0.79	1.46
35 West 35 Reno	0.66	1.43
70 Reno	0.31	1.43
40 West	0.84	2.39
20 West 20 Reno	0.58	2.38
40 Reno	0.42	2.42
10 West	0.93	8.67
5 West 5 Reno	0.65	8.74
10 Reno	0.3	9.17

5. Conclusions

We have shown, via both mathematical modeling and extensive simulations, that TCP Westwood provides a sensible fairness increment with respect to TCP Reno over high-speed networks. Moreover, we have shown that it is friendly to Reno. We have also introduced a simpler variant of RED, called EASY RED, which improves fairness of Reno connections more than RED, whereas the improvement in the case of Westwood connections is much smaller since Westwood already exhibits a fairer behavior by itself.

References

1. Jacobson, V.: Congestion Avoidance and Control. ACM Computer Communications Review, Vol. 18(4) (1988) 314 - 329
2. Allman, M., Paxson, V., Stevens, W. R.: TCP congestion control. RFC 2581, April 1999
3. Mascolo, S.,Casetti, C., Gerla, M., Sanadidi, M., Wang, R.: TCP Westwood: End-to-End Bandwidth Estimation for Efficient Transport over Wired and Wireless Networks. Proceedings of ACM Mobicom, Rome Italy (2001). To appear in ACM Journal on Wireless Networks (WINET), Special Issue on Wireless Networks with selected papers from MOBICOM2001
4. Clark, D.: The design philosophy of the DARPA Internet protocols. Proceedings of Sigcomm in ACM Computer Communication Review, Vol. 18(4) (1988) 106-114
5. Floyd, S., Fall, K.: Promoting the use of end-to-end congestion control in the Internet. IEEE/ACM Transactions on Networking, Vol. 7(4) (1999) 458-72
6. Mogul, J.C.: Observing TCP dynamics in real networks. Proceedings of Sigcomm in ACM Computer Communication Review, Vol. 22(4) (1992) 305-317
7. ns-2 network simulator (ver 2). LBL, URL: http://www-mash.cs.berkeley.edu/ns
8. Jain, R.: The art of computer systems performance analysis. John Wiley and Sons, (1991)
9. Stevens, W.: TCP/IP illustrated, Addison Wesley, Reading, MA, (1994)
10. Iannaccone, g., May, M, and Diot, C.: Aggregate Traffic Performance with Active Queue Management and Drop from Tail, Computer Communication Review, Vol. 31(3) (2001) 4-13
11. Capone, A., Martignon, F.: Bandwidth Estimates in the TCP Congestion Control Scheme. Tyrrhenian IWDC 2001, Taormina Italy (2001)
12. Hoe, J., C., Improving the Start-up Behavior of a Congestion Control Scheme for TCP. Proceedings of ACM Sigcomm in ACM Computer Communication Review, Vol 26(4) (1996) 270-280
13. Morris, R.: TCP behavior with Many Flows. IEEE International Conference on Network Protocols, Atlanta Georgia (1997) 205-211
14. Keshav, S.: A control-theoretic approach to flow control. Proceedings of Sigcomm in ACM Computer Communication Review, Vol. 21(4) (1991) 3-15
15. Allman M., and Paxson, V.: On Estimating End-to-End Network Path Properties. Proceedings of Sigcomm in ACM Computer Communication Review, (1999) 263-274
16. Lai, K. and Baker, M.: Measuring Link Bandwidths Using a Deterministic Model of Packet Delay. Proceedings of Sigcomm in ACM Computer Communication Review, (2000) 283-294
17. Li, S.Q., and Hwang, C.: Link Capacity Allocation and Network Control by Filtered Input Rate in High speed Networks. IEEE/ACM Transactions on Networking, Vol. 3(1) (1995) 10-25
18. Kelly, F.: Mathematical modeling of the Internet. Proceedings of the Fourth International Congress on Industrial and Applied Mathematics, (1999) 105-116
19. Floyd S., and Jacobson, V.: Random Early Detection gateways for congestion avoidance. IEEE/ACM Transactions on Networking, Vol 1(4) (1997)
20. Hollot, C.V., Misra, V., Towsley, D., and Gong, W,: A control Theoretic Analysis of RED. Proceedings of Infocom (2001)
21. May, M., Bolot, J., Diot, C., Lyles, B.: Reasons not to deploy RED. Seventh International Workshop on Quality of Service IWQoS (1999)
22. Hollot, C.V., Misra, V., Towsley, D., and Gong, W.: On Designing Improved Controllers for AQM Routers Supporting TCP Flows. Proceedings of Infocom (2001)
23. Floyd, S.: RED: Discussions of Setting Parameters, (1997). At http://www.aciri.org/floyd/
24. Floyd, S.: Recommendation on using the "gentle" variant of RED, (2000). At http://www.aciri.org/floyd/

25. Padhye, J., Firoiu, V., Towsley, D., Kurose, J.: Modeling TCP Throughput: A Simple Model and its Empirical Validation. Proceedings of Sigcomm in ACM Computer Communication Review, Vol 28(4) (1998) 303-314
26. Floyd, S., Handley, M., Padhye, J., and Widmer, J.: Equation-Based Congestion Control for Unicast Applications. Proceedings of Sigcomm in ACM Computer Communication Review, Vol. 18 (2000) 43-56
27. Feng, W., Kandlur, D., Saha, D., Shin, K.G.: A Self-Configuring RED Gateway. Proceedings of Infocom (1999)
28. Lin, D., and Morris, R.: Dynamics of Random Early Detection. Proceedings of Sigcomm in ACM Computer Communication Review, Vol. 27(4) (1997) 127-137
29. Ott, T.J., Lakshman, T.V., Wong, L.: SRED: Stabilized RED. Proceedings of Infocom (1999)
30. Anjum, F.M., and Tassiulas, L.: Balanced RED: an algorithm to achieve fairness in the Internet. Proceedings of Infocom (1999)
31. Aweya, J., Ouellette, M., Montuno, D.Y.: A control theoretic approach to active queue management. Computer Networks, Vol. 36 (2001) 203-235
32. Floyd, S., Gummadi, R., Shenker, S.: Adaptive RED, An algorithm for Increasing the Robustness of RED's Active Queue Management. Submitted for publication. Available at http://www.aciri.org/floyd/