Additive Increase Early Adaptive Decrease Mechanism for TCP Congestion Control

S. Mascolo[1], L. A. Grieco[2]

(1) Dipartimento di Elettrotecnica ed Elettronica, Politecnico di Bari, Italy.

(2) Dipartimento d'Ingegneria dell' Innovazione, Università di Lecce, Italy.

mascolo@poliba.it, alfredo.grieco@unile.it

Abstract

Due to the fundamental end-to-end design principle, today TCP/IP congestion control algorithm implements an *additive increase multiplicative decrease* (AIMD) probing algorithm. We propose a new mechanism that we call *additive increase early adaptive decrease* (AIEAD). The AIEAD algorithm estimates both the available bandwidth and the queue backlog in a end-to-end fashion: the backlog estimate aims at bounding queue lengths, detecting congestion before overflow and discriminating congestion losses from losses that are due to unreliable links; the bandwidth estimate aims at adaptively setting control windows after congestion detection by taking into account the used bandwidth at the time of congestion (*early adaptive decrease* phase). An implementation of the AIEAD paradigm, which we call New Westwood (NW), is tested and compared with Reno, and Vegas TCP using the *ns-2* simulator. Simulations show that NW significantly improves fairness and goodput with respect to both Reno and Vegas, whereas in a last-hop wireless lossy link scenario NW significantly improves the goodput.

1. Introduction

The stability of the Internet requires that flows use some form of end-to-end congestion control to adapt the input rate to the available bandwidth [1,3]. Due to the fundamental end-to-end design principle of the TCP/IP for which the network is a black box that cannot supply any explicit feedback [4,5], the TCP congestion control algorithm was designed by following the additive-increase multiplicative-decrease probing paradigm (AIMD) [1,6]. The *additive increase* phase aims at increasing the flow input rate until the network available capacity is hit and a congestion episode happens. The sender becomes aware of congestion via the reception of duplicate acknowledgments (DUPACKs) or the expiration of a timeout. Then it reacts to light congestion (i.e. 3 DUPACKs) by halving the congestion window (*fast recovery*) and sending again the missing packet (*fast retransmit*), and to heavy congestion (i.e. timeout) by reducing the congestion window to one. This is the multiplicative decrease phase that aims at quickly shrinking the input rate after congestion.

While today end-to-end TCP congestion control can insure that network capacity is not exceeded, it cannot insure fair sharing of that capacity [1]. Furthermore, today TCP is not well suited for wireless links since losses due to radio channel problems are misinterpreted as a symptom of congestion by current TCP schemes and lead to an undue transmission rate reduction. Thus, TCP requires supplementary link layer protocols such as reliable link-layer or split-connections approach to efficiently operate over wireless links [2,8,15,20].

In this paper we propose a new control mechanism that we call *additive increase early adaptive decrease* (AIEAD) that provides significant increase of (a) utilization of wireless links and (b) fairness in bandwidth usage of FIFO networks.

In order to obtain a friendly behavior towards Reno TCP, the standard TCP increasing phases, which are the *slow-start* and *congestion avoidance* phases, are left unchanged. The new *early adaptive decrease* phase has two parts: (1) congestion is detected before queue overflow using an end-to-end estimate of the queue backlog (early detection); (2) the congestion window is adaptively reduced by taking into account an accurate measure of the bandwidth used when congestion is detected.

The AIEAD mechanisms has several major advantages: (1) it early reacts to congestion, thus implementing a sort of end-to-end early congestion detection (ECD) (see [9] for an implementation of this concept at network layer); (2) it is able to control queuing build up so that queuing delays can be kept low, which is generally regarded as a good practice (see [10]); (3) it increases stability and transport efficiency because of its adaptive setting of control windows that can be more than multiplicative when in the presence of heavy congestion, and less than multiplicative in the case of light congestion; (4) it increases the fairness in bandwidth utilization since it tends to target the same backlog for each flow; (5) it significantly improves the goodput in wireless networks because it is able to discriminate congestion losses from losses that are due to unreliable links: in fact, after 3 DUPACKs, standard Reno TCP halves the *cwnd* whereas New Westwood estimates the queue backlog and only if this backlog is greater than a threshold then the *cwnd* is reduced to match the available bandwidth.

TCP Vegas is the first significant example of congestion control algorithm that partially departs from the AIMD paradigm [11]. Vegas estimates the expected connection rate as $cwnd/RTT_m$ and the actual connection rate as $cwnd/RTT$, where RTT is the round trip time and RTT_m is the minimum measured RTT. When the *difference* between the expected and the actual rate is less than a threshold $\alpha>0$, the *cwnd* is additively increased. When the *difference*

0-7803-7661-7/03/$17.00©2003 IEEE.

is greater than a threshold $\beta>\alpha$ then the *cwnd* is additively decreased. When the difference is between α and β, *cwnd* is maintained constant. It is worth noticing that New Westwood differs from Vegas TCP for the following features: (1) NW estimates the bandwidth used by a TCP flow by averaging the rate of the returning ACKs whereas Vegas uses the actual and expected sending rates to early detect congestion. The sending rate and the used bandwidth can be different under dynamic condition; (2) Vegas is not able to grab its bandwidth share when coexisting with Reno connections and therefore its deployment in the real Internet cannot be recommended [26]. On the other hand, NW safely coexists with Reno connections since its probing phase is left as in Reno; (3) NW sets the control windows by taking into account the used bandwidth whereas Vegas follows an additive-increase additive-decrease paradigm.

Another example of control algorithm that departs from of the AIMD paradigm is Westwood TCP [13], which uses an end-to-end bandwidth estimation to set control windows after congestion. TCP Westwood improves utilization of wireless links and ensures network stability. However, in the presence of ACK compression the bandwidth estimation proposed in [13] causes an overestimate of the bandwidth that is potentially disruptive of fairness and can lead to starvation of coexisting connections [12,14]. In [12], a new bandwidth estimation algorithm that overcomes this problem has been proposed. We name this algorithm Westwood+. The friendliness of Westwood+ towards Reno has been also analytically shown in [12]. As we will show in section 3.3, the new bandwidth estimate does not improve the utilization of wireless link as the original one does [13].

In this paper we propose a sender side only implementation of the AIEAD paradigm we call New Westwood TCP (NW TCP). The NW TCP sender infers congestion when the estimated queue backlog exceeds a threshold. When congestion is detected, the congestion window and the slow start threshold are set equal to the estimated available bandwidth times the minimum measured round trip time and the congestion avoidance phase is entered. It is worth noticing that New Westwood is particularly able to ensure high utilization of wireless links because it is able to discriminate congestion losses from wireless losses by estimating the queue along the path.

Extensive simulations are reported to evaluate the goodput, the fairness, and the friendless of New Westwood in comparison with Reno, Vegas and Westwood+ using the ns-2 network simulator [29]. We have chosen a comparison with Reno TCP because it is the leading Internet congestion control protocol that implements the AIMD paradigm. We did not consider other variants such as Reno SACK [16] or New Reno [17] because they implement features that are "orthogonal" to the AIMD/AIEAD paradigm, that is, they do not touch these paradigms and could be implemented also in New Westwood to provide same benefits. For the same reason we do not consider the larger initial window option [3,18]. On the other hand, we consider always enabled the timestamps and the delayed acknowledgement option even though the latter degrades *RTT* measurements and bandwidth estimation [19]. Vegas TCP has been considered for comparison because it also proposes a sender side mechanism for throttling the congestion window that is based on backlog estimate.

An exhaustive overview of all congestion control algorithms proposed in the literature is beyond the scope of this work, which focuses on the ones that we consider the most relevant to our study.

The work is organized as follows: Section 2 describes NW TCP, Section 3 reports simulation results for several wired and wireless scenarios. Finally, Section 4 draws the conclusions.

2. New Westwood TCP

2.1 Overview of the algorithm

New Westwood TCP is a sender side only implementation of the AIEAD mechanism that follows the fundamental end-to-end Internet design principle [4,5]. The key idea is to exploit the flow of returning ACKs to estimate both available bandwidth and queue backlog.

NW TCP preserves the standard TCP slow start and congestion avoidance phases in order to probe the network capacity until congestion is experienced. While in standard TCP (i.e. Tahoe/Reno) congestion is signaled by timeouts or duplicate acknowledgments, in NW TCP the source becomes aware of congestion by estimating the queue backlog. This allows the sender to (1) promptly detect congestion before queue overflow, (2) increase the fairness since each sender sets the same backlog threshold for each flow, (3) discriminate congestion from losses due to unreliable links.

In order to estimate the queue backlog, NW TCP needs to estimate the available bandwidth and the queuing time. The queuing time is measured by time-stamping packets and by subtracting the minimum measured round trip time (RTT_m) from the current *RTT*.

The end-to-end estimate of the available bandwidth is obtained by low-pass filtering the rate of returning ACKs. The estimate is then multiplied by the queuing time to obtain an estimate of the queue backlog. When the queue backlog is greater than a threshold (*qthresh*) then the TCP sender sets the congestion window (*cwnd*) and the slow start threshold (*ssthresh*) equal to the available bandwidth times the minimum round trip time. The rationale of this strategy is simple: in contrast with TCP Reno, which implements a "blind" multiplicative decrease algorithm after congestion, NW TCP adaptively sets a slow start threshold and a congestion window, which are consistent with the bandwidth used at the time congestion is detected.

2.2 The end-to-end bandwidth estimation algorithm

Estimating the available bandwidth at the beginning of a connection over a FIFO-queuing network as done in [7,21-24,30] is a very different and much more difficult task than measuring the actual rate a connection is achieving during the data transfer as it is done by Westwood TCP in [13]. In

[13] the idea is to estimate the available bandwidth by properly filtering the flow of returning ACKs. A sample of available bandwidth $b_k = d_k/(t_k - t_{k-1})$ 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 considers delayed ACKs, duplicate ACKs and selective ACKs [13]. Bandwidth samples b_k are low-pass filtered by employing a discrete-time low-pass filter to obtain the bandwidth estimate $\hat{b}_k$. Low-pass filtering is necessary since congestion is due to low frequency components of available bandwidth, and because of delayed ACK option [25]. However, this estimation scheme is affected by ACK compression, which happens in the presence of reverse traffic [14]. In fact, a consequence of ACK compression is that bandwidth samples contain high frequency components that cannot be filtered out by using a low-pass discrete time filter. As a consequence, ACK compression causes a systematic bandwidth overestimate. This overestimate may disrupt the fairness between TCP connections and even may lead to starvation of some connections. In order to avoid the effects of ACK compression, we employ the filtering algorithm proposed in [12], which computes a sample of bandwidth b_k every *RTT* instead of every time an ACK is received by the sender. More precisely, the estimation scheme proposed in [12] counts the amount of data $D_k = \sum d_j$ acknowledged during the last *RTT* to compute the bandwidth sample $b_k = D_k/\Delta_k$, where Δ_k is now the last *RTT*. This operation corresponds to evenly spread the bandwidth samples $d_j/(t_j - t_{j-1})$ over the *RTT* interval, that is, it corresponds to low-pass filtering high frequency components due to ACK compression.

2.3 The end-to-end estimation of the queue backlog

Once low frequency component of the available bandwidth is obtained as outlined in the previous subsection, it is possible to estimate the queue backlog by measuring the *RTT*. In fact, by employing the timestamp option, it is possible to measure *RTT* and keep track of the minimum *RTT*, so that the queuing time $T_k^q = RTT - RTT_m$ can be measured with precision every time an ACK is received. In all simulations herein reported, we have assumed a 10ms clock granularity. By using the bandwidth estimate $\hat{b}_k$ and the queuing time T_k^q, the NW TCP sender can estimate the connection queue backlog $\hat{q}_k$ at time t_k as: $\hat{q}_k = \hat{b}_k T_k^q$.

2.4 The New Westwood algorithm

NW TCP implements the standard slow-start, congestion avoidance and fast retransmit mechanisms [3]. It introduces 2 novelties: (1) an end-to-end estimation of the path backlog to increase fairness and discriminate losses due to congestion from losses due to unreliable links thus improving utilization of wireless links; (2) *cwnd* and *ssthresh* are adaptively set after congestion detection or timeout to match the available bandwidth;

NW TCP becomes aware of congestion when the queue backlog estimate exceeds the *qthresh*. Thus, on ACK reception, the *cwnd* is increased as in TCP Reno only if the estimated number of buffered segments is not greater than the *qthresh*. When the estimated backlog exceeds the *qthresh*, then *ssthresh* and *cwnd* are set equal to the available bandwidth times the RTT_m. This mechanism clearly improves network stability since a TCP source can detect and react to congestion before queue overflow. After the reception of 3 DUPACKs the congestion window and the slow-start threshold are set as in the previous case only if the queue backlog exceeds the *qthresh*. This feature allows the TCP sender to distinguish losses due to unreliable links from congestion thus improving utilization of wireless links. After timeout expiration, the congestion window and the slow-start threshold are always set equal to the available bandwidth times the RTT_m.

To conclude, it is worth to make a few comments on the way NW TCP sets the *cwnd* and the *ssthresh* after congestion detection. We have seen that when congestion is detected at time t_k, a measure of the used bandwidth is given by the estimate $\hat{b}_k$. The setting $cwnd = \hat{b}_k \cdot RTT$ would produce an input rate that is exactly equal to the used bandwidth. However this setting would lead to an unstable system because an input rate equal to the experienced output rate would not drain the queue level. Then, an increasing *RTT* due to an increasing backlog would increase the input rate causing a positive feedback that would lead to instability. On the other hand, the setting $cwnd = \hat{b}_k \cdot RTT_m$ would sustain an input rate less than the output rate so determining queue depletion. Guaranteeing queue depletion has the important consequence of improving statistical multiplexing of flows traversing a FIFO queue, that is, improves fairness. From another point of view it can be said that the setting $cwnd = \hat{b}_k \cdot RTT_m$ keeps the pipe full while reducing the backlog to zero.

Finally, it is worth noticing that the test on the queue backlog is crucial to improve fairness. In fact, this test requires that, before reducing the control windows, a number of packets equal to the *qthresh* must be queued in the routers along the path. Since taking space in a FIFO queue is equivalent to take bandwidth, this feature guarantees that any flow gets its share. Loss-driven algorithms such as Reno do not guarantee this, because flows experiencing congestion losses are reduced without considering the quota of buffer space (i.e. bandwidth) they are holding.

3. Simulation Results

New Westwood is evaluated and compared with Westwood+, Reno and Vegas TCP using ns-2 simulator [29]. In order to obtain a fair comparison, we assume that all TCP flavors benefit by the same information, that is, they all implement the timestamp and delayed ACK options. Several single and multi-hop scenarios with wired

as well as wireless links are considered. Sender and receiver socket sizes are set large enough so that connections are limited always by network capacity. Vegas parameters are set as the default values used in the Vegas code available at the ns-2 site [29]. Segments are 1500 bytes long, the *qthresh* has been set equal to 20 segments and router buffer capacities have been set equal to the bandwidth delay product as recommended in [31].

3.1 A single TCP connection scenario

In order to clearly compare the behavior of different TCP algorithms, we start by considering a single persistent TCP connection that transmits data for 500s over a 2Mbps link (see Figure 1). The *RTT* of the TCP_1 connection is equal to 252ms. Figs. 2 and 3 show the *cwnd*, *ssthresh* and bottleneck queue length when different TCP flavors are employed.

The comparison shows that the *cwnd* of TCP Vegas is constant whereas the *cwnd* of NW, Westwood+ and Reno exhibits a cyclic behavior in order to continuously probe network capacity. The main consequence of the probing behavior is that, when NW and Reno coexist, they are friendly to each other whereas when Vegas coexist with Reno it gives up bandwidth to Reno. This behavior was also reported in [26]. Moreover, by comparing Figs. 2 (a) and (c) it can be noted that the first setting of the *ssthresh* operated by Westwood+ is larger than the one of Reno, that is, Westwood+ is faster than Reno to "learn" the available capacity.

Figure 2(d) shows that NW TCP reduces by half the oscillation range of *cwnd* with respect to both Reno and Westwood+ due to early congestion detection. Figure 3(d) shows that NW TCP nicely estimates the queue backlog, which is also bounded by the *qthresh*. In other words, in contrast with Reno or Westwood+ that systematically hit the queue capacity and lose packets, NW TCP is able to set an *upper bound for the queue length*. This bound is set by the sender and can be *any small value*, i.e., the queuing delay can be kept low, which is generally regarded as a good practice [10].

Finally, in this subsection, we investigate the impact of reverse traffic. The reverse traffic is contributed by 10 Reno connections that go from the right to the left side of Figure 1 sharing the path with the ACK packets of the considered TCP_1 connection. Figure 4 shows the *cwnd* and the *ssthresh* behaviors of Reno, Vegas, Westwood+ and New Westwood. It can be viewed that Vegas TCP exhibits a very small *cwnd* thus providing a small goodput equal to 0.4Mbps. On the other hand, Westwood+ and NW exhibit remarkably larger *cwnd* and *ssthresh* windows w.r.t. to Reno TCP. This turns out in goodputs of 1.64Mbps, 1.54Mbps and 1.36Mbps for Westwood+, NW and Reno, respectively. Queue dynamics are not reported since they are similar to the ones reported in the Figure 3. This sub-section has highlighted that Vegas performances degrade when in the presence of congestion along the ACK path, this behavior was also reported in [27].

3.2 A single hop scenario

In order to evaluate how a set of N TCP flows of the same flavor with different *RTTs* share the capacity of a common bottleneck, we consider N=10,40,70,100,140,170,200 infinite greedy connections sharing a FIFO bottleneck with *RTTs* ranging uniformly from (252/N)ms to 252ms. Simulations last 100s. To provide a single numerical measure reflecting the fair share distribution across the various connections we use the Jain fairness index defined as $F.I.=\frac{(\sum_{i=1}^{N} b_i)^2}{N\sum_{i=1}^{N} b_i^2}$, where b_i is the goodput and N is the number of connections [11,28]. The goodput is defined as: *(total_data_sent − retransmitted_data)/transfer_time*.

Simulation results have shown that New Westwood, Westwood+, Reno and Vegas all achieve high utilization of the bottleneck bandwidth.

Figure 5 shows the Jain fairness index as a function of the number of connections N when the bottleneck capacity is 10Mbps and 100 Mbps. NW TCP provides an excellent fairness index in all cases. In particular, it is worth underlining that in the case of the 10Mbps link the buffer capacity is equal to 210 segments. Such a capacity is not enough wide to accommodate *qthresh* segments for each source when N>10. Nevertheless, NW provides a fairness index, which is comparable or slightly better than the Westwood+ one and very larger than the Vegas one, especially for N>40. On the other hand, in the case of the 100Mbps bottleneck, the large buffer capacity of 2100 segments greatly improves NW and Vegas fairness indexes.

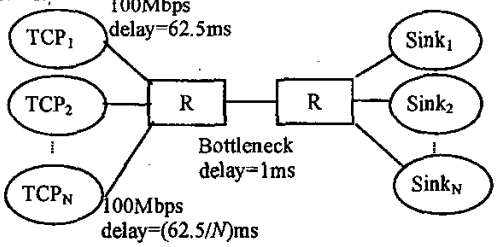

Figure 1: Single bottleneck scenario.

To give a further insight into both bandwidth sharing and estimation provided by New Westwood, Figure 6(a) reports the end-to-end bandwidth estimate versus time in the case of 40 NW connections sharing a 100Mbps bottleneck. The figure shows that, after an initial transient, the bandwidth estimate of each sender impressively converges to the same small interval of values, that is, bandwidth is correctly estimated and fairly allocated.

To test the bandwidth estimate algorithm under strenuous conditions, N New Westwood reverse connections going along the path of the ACK of the forward connections are added to provoke ACK compression. Figure 6(b) clearly shows that New Westwood is able to nicely estimate a fair bandwidth share when in the presence of ACK compression. In fact results are similar to the ones obtained without reverse traffic and reported in Figure 6(a).

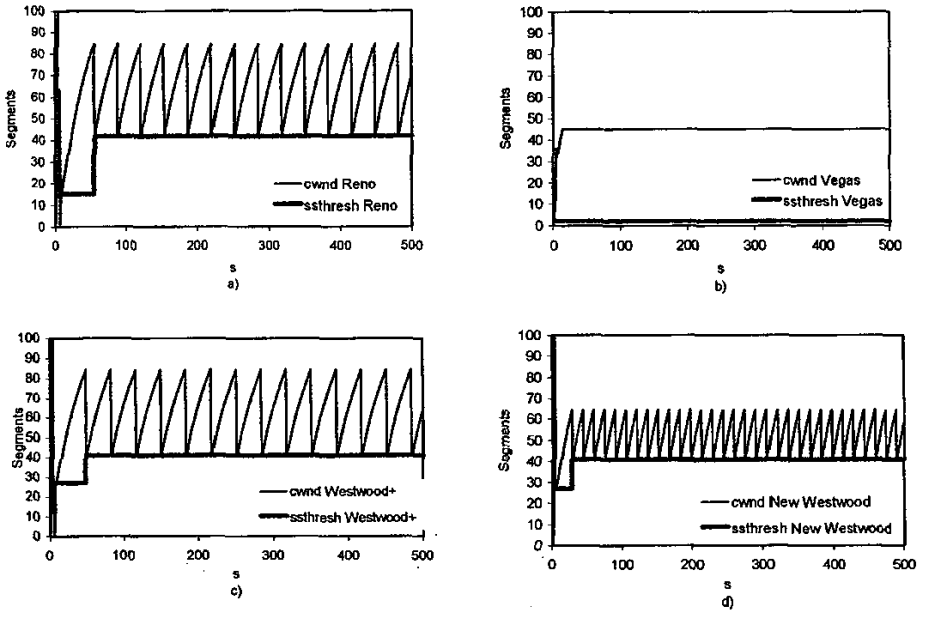

Figure 2: *cwnd* and *ssthresh* dynamics using different TCP algorithms in the case of a single connection.

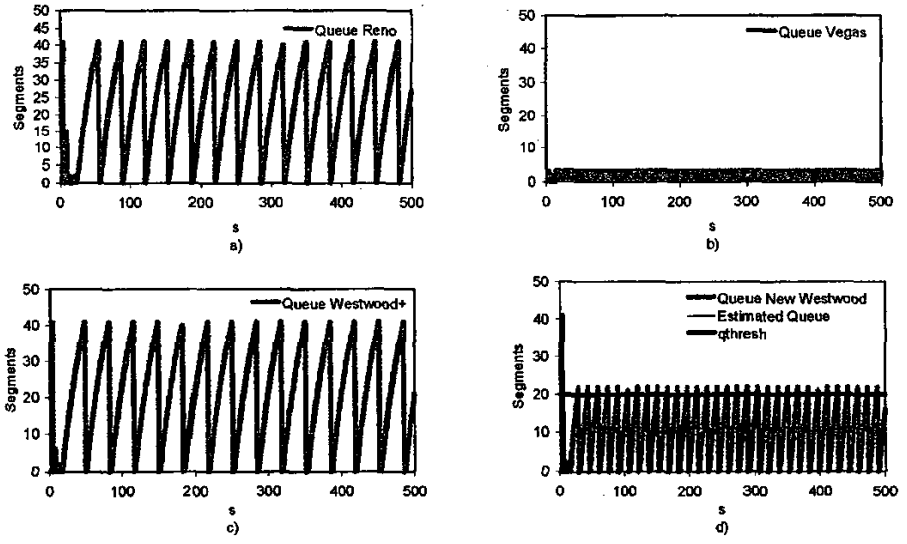

Figure 3: Bottleneck queue behaviors using different TCP algorithms in the case of a single connection.

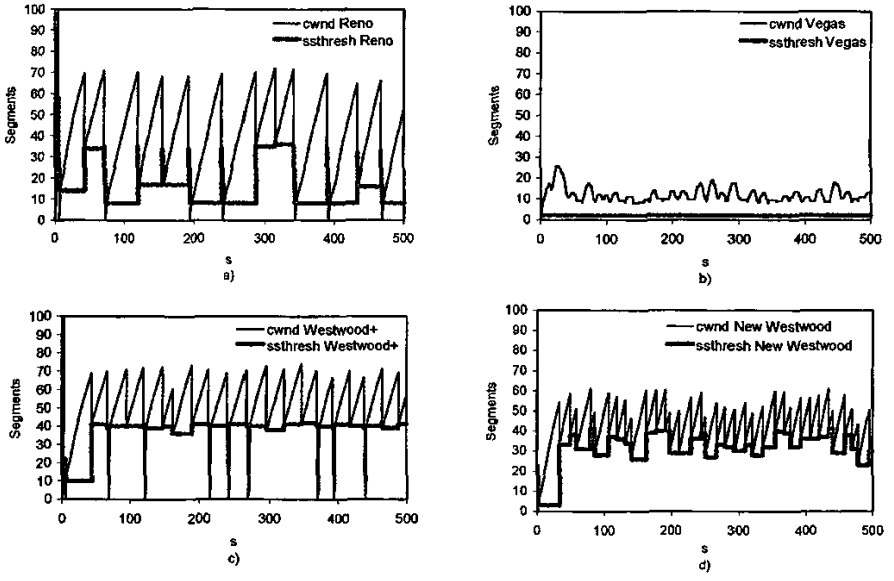

Figure 4: *cwnd* and *ssthresh* using different TCP in the case of a single connection in the presence of Reno reverse traffic.

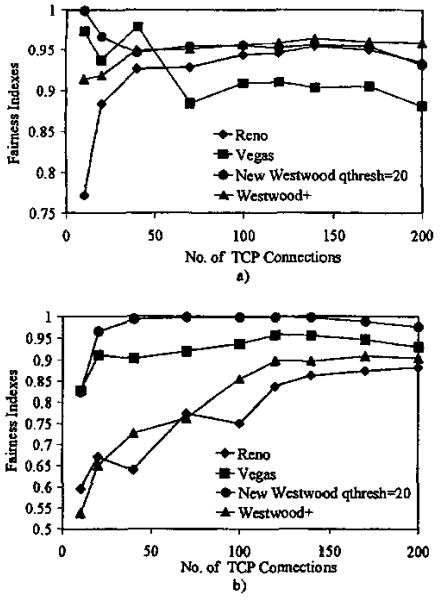

Figure 5: Fairness Indexes over a single bottleneck topology with capacity: a) 10Mbps; b) 100Mbps.

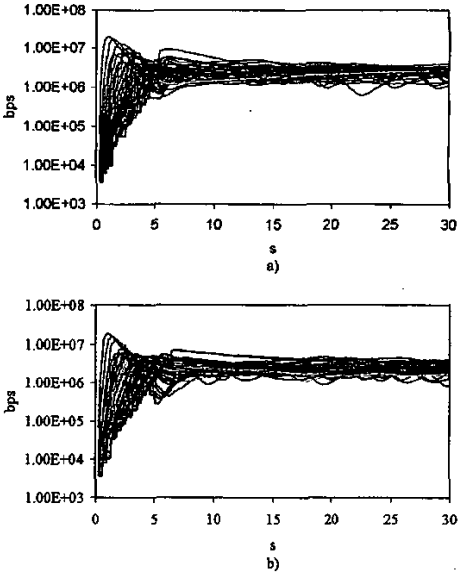

Figure 6: Bandwidth estimates in the case of 40 New Westwood connections sharing a 100Mbps bottleneck: a) without reverse traffic; b) with reverse traffic fed by 40 New Westwood connections.

3.3 A multihop scenario with lossy links

In this section we compare New Westwood with Westwood+, Reno, and Vegas in the presence of unreliable links such as radio links. We consider the goodput of a TCP_1 connections going through 10 congested gateways and finally reaching a last hop lossy link. On each gateway one TCP cross traffic source shares the outgoing link with the TCP_1 connection. The *RTT* of the TCP_1 connection is 250ms whereas the *RTT* of the cross traffic sources is 50ms. The wireless link capacity is 2Mbps. The capacity of the links between routers is 5Mbps (see Figure 7). The duration of the simulation is 110s. The cross-traffic starts at the time zero whereas the source TCP_1 starts at the time 10s. To investigate the impact of losses due to unreliable links, we assume an independent uniform packet loss probability ranging from 0.1% to 10% affecting the last hop link in both directions.

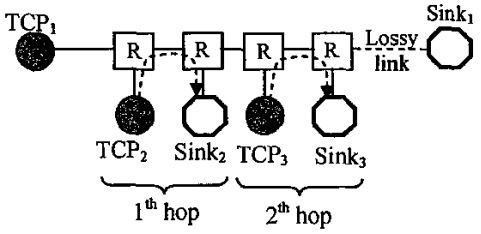

Figure 7. Multiple congested gateways with a last hop lossy link.

Figure 8(a) shows the goodput of the TCP_1 connection as a function of the packet loss rate. Four curves are shown that refer to the different TCP_1 flavors. All the cross-traffic connections are Reno. Figure 8(a) shows that New Westwood improves the goodput when the packet loss rate is greater than 1%. Results reported in Figure 8(b) have been obtained by changing the cross-traffic from Reno to New Westwood. Comparison of Figs. 8(a) and (b) show that all TCP_1 flavors perform better when going through New Westwood cross-traffic. In particular, it is worth noticing that the goodput of TCP_1 Reno is greater in the presence of New Westwood cross-traffic than in the presence of Reno cross-traffic, that is, New Westwood is more than friendly to Reno. The goodput achieved by NW TCP_1 in the presence of NW cross-traffic improves from 730% to 1600% with respect to Reno TCP_1 coexisting with Reno cross-traffic when the packet loss rate ranges from 0.1% to 10%.

4. Conclusions

The proposed AIEAD paradigm has several advantages that have been tested using the *ns-2* simulator. In particular it has been shown that the end-to-end queue estimation, along with the bandwidth estimation, provide two major advantages: (1) queuing build up can be bounded by the threshold *qthresh:* as a consequence, each flow sharing a FIFO bottleneck queue gets the same maximum space that turns out in fair bandwidth sharing; (2) the estimate of the queue backlog allows congestion losses be distinguished from losses due to unreliable links thus remarkably increasing throughput in the presence of wireless links. Simulations have shown that NW significantly improves fairness and improves goodput with respect to both Reno and Vegas, whereas in a last-hop wireless scenario NW improves goodput from 730% to 1600% when the packet loss probability ranges from 0.1% to 10%.

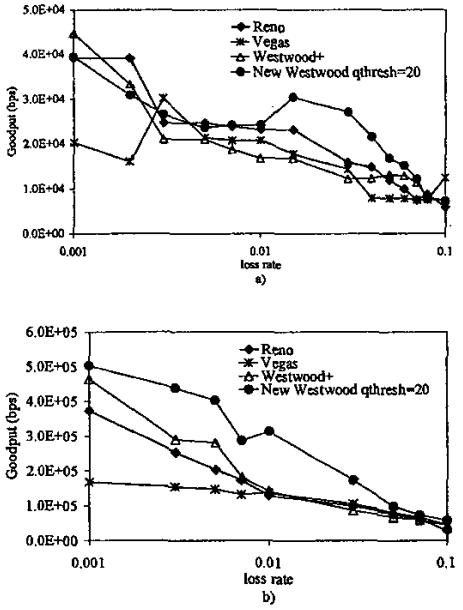

Figure 8: Goodput *vs* loss rate over a 10 hops scenario in the presence of cross-traffic: a) Reno cross-traffic; b) New Westwood cross-traffic with *qthresh*=20.

References

[1] V. Jacobson, "Congestion Avoidance and Control," in Proceedings of *ACM SIGCOMM '88*, Stanford CA, August 1988, pp. 314-329.

[2] R. Krishnan, M. Allman, C. Partridge, and J. P. G. Sterbenz, "Explicit Transport Error Notification (ETEN) for Error Prone Wireless and Satellite Networks," *BBN Technical Report* No. 8333, March 22, 2002.

[3] M. Allman, V. Paxson, W. R. Stevens, "TCP congestion control," RFC 2581, April 1999.

[4] D. Clark, "The design philosophy of the DARPA Internet protocols," in Proceedings of *ACM Sigcomm'88*, Stanford CA, August 1988, pp. 106-114.

[5] S. Floyd, K. Fall, "Promoting the use of end-to-end congestion control in the Internet," *IEEE/ACM Transactions on Networking*, 7(4), (1999), pp. 458-472.

[6] Dah-Ming Chiu, R. Jain, "Analysis of the increase

and decrease algorithms for congestion avoidance in computer networks," *Computer Networks and ISDN Systems*, June 1989, vol.17, (no.1), pp. 1-14.

[7] B. Melander, M. Bjorkman, and P. Gunningberg, "A New End-to-End Probing and Analysis Method for Estimating Bandwidth Bottlenecks," in Proceedings of *Global Internet Symposium*, 2000.

[8] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, R. H. Katz, "A comparison of mechanisms for improving TCP performance over wireless links," *IEEE/ACM Transactions on Networking*, Dec. 1997, vol.5, no.6, pp. 756-769.

[9] S. Floyd, "TCP and explicit congestion notification," *Computer Communications Review*, 24 (5), pp. 10-23, October 1994.

[10] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, L. Zhang, "Recommendations on queue management and congestion avoidance in the Internet," RFC 2309, April 1998.

[11] L. L. Brakmo, L. L. Peterson, "TCP Vegas: End-to-end congestion avoidance on a global Internet," *IEEE Journal on Selected Areas in Communications* (JSAC), 1995, vol. 13, no.8, pp. 1465-1480.

[12] L.A. Grieco and S. Mascolo, "Westwood TCP and easy RED to improve Fairness in High Speed Networks," in Proceedings of *IFIP/IEEE Seventh International Workshop on Protocols For High-Speed Networks*, PfHSN 2002, April 22-24, 2002 Berlin, Germany.

[13] S. Mascolo, C. Casetti, M. Gerla, M. Sanadidi, R. Wang, "TCP Westwood: End-to-End Bandwidth Estimation for Efficient Transport over Wired and Wireless Networks," in Proceedings of *ACM Mobicom 2001*, July, Rome, Italy.

[14] J. C. Mogul, "Observing TCP dynamics in real networks," in Proceedings of *ACM Sigcomm 1992*, Baltimore, MD, USA, pp. 305-317.

[15] C. Barakat, E. Altman, "Bandwidth tradeoff between TCP and link-level FEC," *Computer Networks*, no. 39(2002), pp. 133-150.

[16] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgement Options," RFC 2018, April 1996.

[17] S. Floyd, T. Henderson, "NewReno Modification to TCP's Fast Recovery," RFC 2582, April 1999.

[18] M. Allman, S. Floyd, C. Partridge, "Increasing initial TCP's initial window," RFC 2414, Sept. 1998.

[19] V. Jacobson, R. Braden, D. Borman, " TCP Extensions for High Performance", RFC 1323, May 1992.

[20] H. M. Chaskar, T. V. Lakshman, and U. Madhow, "TCP Over Wireless with Link Level Error Control: Analysis and Design Methodology," *IEEE/ACM Transactions on Networking*, Oct. 1999, vol.7, (no.5): pp. 605-615.

[21] S. Keshav, "A Control-theoretic Approach to Flow Control," in Proceedings of *ACM Sigcomm 1991*, Zurich, Switzerland, September 1991, pp. 3-6.

[22] J. C. Hoe, "Improving the Start-up Behavior of a Congestion Control Scheme for TCP," in Proceedings of *ACM Sigcomm '96*, Palo Alto, CA, August 1996, pp. 270-280

[23] M. Allman and V. Paxson, "On Estimating End-to-End Network Path Properties", in Proceedings of *ACM Sigcomm 1999*, Cambridge, Massachusetts, August 1999, pp. 263-276.

[24] K. Lai and M. Baker, "Measuring Link Bandiwdths Using a Deterministic Model of Packet Delay," in Proceedings of *ACM Sigcomm 2000*, Stockholm, Sweden, August 2000, pp. 283-294.

[25] S. Q. Li and C. Hwang, "Link Capacity Allocation and Network Control by Filtered Input Rate in High speed Networks," *IEEE/ACM Transactions on Networking*, vol. 3, no. 1, Feb. 1995, pp. 10-25.

[26] J. Mo, R. J. La, V. Anantharam, J. Walrand, "Analysis and comparison of TCP Reno and Vegas," in Proceedings of *IEEE Infocom 1999*, New York NY, March 1999, pp. 1556-1563.

[27] C. Parsa and J. J. Garcia-Luna-Aceves. "Improving TCP congestion control over internets with heterogeneous transmission media," in Proceedings of *IEEE ICNP 99*.

[28] R. Jain, "The art of computer systems performance analysis," John Wiley and Sons, 1991.

[29] Ns-2 network simulator (ver 2). LBL, URL: http://www-mash.cs.berkeley.edu/ns.

[30] M. Jain, C. Dovrolis, "End to End Available Bandwidth: Measurement Methodology, Dynamics, and Relation with TCP Throughput," in Proceedings of *ACM Sigcomm 2002*, Pittsburgh, PA, USA

[31] C. Villamizar and C. Song "High Performance TCP in ANSNET," *ACM Computer Communication Review*, vol. 24, no. 5, pp. 45-60.