

# TCP Westwood with Adaptive Bandwidth Estimation to Improve Efficiency/Friendliness Tradeoffs

Mario Gerla, Bryan Kwok Fai Ng, M.Y. Sanadidi, Massimo Valla, Ren Wang\*  
UCLA Computer Science Department  
Los Angeles, CA 90095, USA  
Tel: +1-310-206-8589. Fax: +1-310-825-7578  
E-mail: {Gerla,bryannng,medy,mvalla,renwang}@cs.ucla.edu

## Abstract

In this paper, we propose an extension of TCP Westwood allowing the management of the Efficiency/Friendliness-to-NewReno tradeoffs. We show that the extended TCP Westwood is able to achieve higher total link utilization, yet at the same time maintain friendliness. TCP Westwood (for short, TCPW) implements a novel window congestion control algorithm based on eligible rate estimation. The performance of TCPW has been promising, exceeding that of TCP NewReno in “large leaky pipes”; i.e. network paths with high bandwidth-delay product and non-negligible random error rate. Consider the situation where TCPW and TCP NewReno connections coexist and share common bottlenecks. Friendliness in this shared environment is paramount. Under certain conditions TCP NewReno may experience some performance degradation since TCPW “learns” more about connection performance and thus can take better advantage of available bandwidth. To manage the efficiency/friendliness tradeoffs, we propose to combine the original TCPW Bandwidth Estimation (BE) strategy with a new Rate Estimation (RE) strategy. One finds that BE provides significantly higher utilization, but may, under certain conditions, overestimate a connection fair share. RE, on the other hand, tends to be closer to the achieved rate of a connection, but it may underestimate the connection fair share. The question is: which estimate – RE or BE – yields better throughput/friendliness tradeoffs? Our studies show that RE works best when packet loss is mostly due to congestion. If, on the other hand, packet loss is mostly due to link errors, BE gives better performance. To achieve the “best of all worlds”, we introduce a method we call Combined Rate and Bandwidth estimation (CRB.) A connection first infers the predominant cause of packet loss (buffer congestion or random error) and then uses the more appropriate estimation method. Simulation shows that the adaptive CRB provides a very effective compromise between efficiency and friendliness.

## 1. Introduction and Related Work

The Transmission Control Protocol (TCP) protocol provides end-to-end, reliable, congestion controlled connections over the Internet [CK74]. The congestion control method introduced in TCP Tahoe included two phases: slow-start and congestion avoidance [Jaco88]. In TCP Reno, recovery from sporadic packet losses is enhanced by Fast Retransmission and Fast Recovery [Jaco90]. When three duplicate acknowledgments are received at a sender, the packet that is acknowledged three times or more is presumed lost. Instead of waiting for the lost packet timeout to expire, in TCP Reno, the sender immediately retransmits the lost packet (Fast Retransmit). Then, the slow start threshold is set to current window size, the congestion window is halved (Fast Recovery) and the congestion avoidance phase is entered. Recovery enhancements techniques have been developed to diminish the impact of multiple losses in a single window. SACK-based TCP [RFC2883], [FF96] provides the sender with more complete information about which packets are lost and speeds up recovery in case of multiple losses within a window. Another TCP version referred to as “NewReno” [RFC2582] [Hoe96] also speeds-up multiple loss recovery without requiring SACK information and applying only sender side modifications. Recent measurements show that the majority of TCP implementations are NewReno [PF01]. Increasingly, TCP is called upon to provide reliable and efficient data transfer over a variety of link technologies including wired (e.g. cable and fiber optic), and wireless (ground radio, and satellite links), with increasing bandwidth capacity. The new ultra high speed wired/wireless environment is exceeding the range for which TCP was initially designed, tested and tuned. As a consequence, active research is in progress to extend the domain of effective TCP operability [GMLW99][KVM99][GMPG00][BSAK95] [BK98].

In this paper, we explore the use of bottleneck bandwidth estimation in order to improve TCP performance in the above mentioned novel and challenging network environments and traffic conditions. We study and extend a TCP variant called TCP Westwood which uses bandwidth estimation. TCP Westwood has been introduced and demonstrated before

---

\* The author names are arranged in alphabetical order.

[CGMSW01]. The novel contribution of this paper is to propose an adaptive bandwidth estimation technique that considerably improves the friendliness (towards NewReno) and makes TCP Westwood a “good Internet citizen”.

The use of bandwidth and rate estimates for enhancing congestion control in TCP has been proposed before; see for example, Packet Pair [Kesh91] and TCP Vegas [BP95]. In TCP Vegas, the sender infers the bottleneck backlog, a congestion level measure, from round trip time (RTT) and current connection rate (derived from average ACK arrival rate measurements). If backlog increases beyond a specified threshold, the source will decrease its congestion window (*cwnd*), thus reducing its transmission rate, and vice versa. Some fairness issues with TCP Vegas were pointed out in [HMM98] [BB00]. It was argued there that new arriving connections to a congestion in progress may not be able to get a fair share of the bottleneck bandwidth. In the Packet Pair scheme, a sender estimates the available bottleneck bandwidth from ACK pair interval and from that computes the bottleneck backlog. It then adjusts its sending rate accordingly. Again, fairness is an issue. To this end, the Packet Pair scheme explicitly assumes Round-Robin scheduling at the routers to achieve good estimates and maintain fairness. Round Robin scheduling is a feature not available in many commercial routers. It is interesting to note that from the very beginning the techniques based on bandwidth estimation ran into fairness problems.

Packet Pair and TCP Vegas addressed steady state performance. Similar bandwidth estimation techniques were also proposed to improve congestion window convergence to steady state. For example, Hoe reported in [Hoe96] a method to improve start up behavior of TCP connections. The idea is to use a Packet Pair type bandwidth estimate in setting the first slow start threshold (*ssthresh*) value of a connection. The impact is a more rational *ssthresh* setting, which is actually important for short lived connections over large pipes. Also related to this work is the probing of “available bandwidth” on a path. Allman and Paxson, in [AP99], reported and compared techniques for probing the available bandwidth in order to properly initialize the *ssthresh* before a TCP connection is started. An excessively large *ssthresh* can lead to premature Timeout, unnecessary return to slow start and thus lower efficiency. Finally, Lai and Baker in [LB00] describe an improved measurement technique (packet tailgating) to probe available bandwidth that is less intrusive (i.e., consumes less bandwidth) than previous techniques. This bottleneck capacity estimation method and has not been used in TCP congestion control. The methods above share with TCPW the notion of measuring the ACK interarrival intervals to estimate bandwidth information. They are not intended for TCP congestion control throughout a connection lifetime.

In the attempt to make the TCP protocol fairer and more robust to the high speed and wireless medium challenges, in addition to modifying the TCP protocols itself, research has also been directed to instrumenting and/or strengthening the lower layers. In particular, several network and link-layer enhancements have been proposed to improve TCP performance under various conditions (congestion loss, random loss, handoff, out of order delivery, etc.). These schemes fall into two general categories: “Transparent” schemes, and “Active feedback” schemes. Transparent schemes do not provide feedback to TCP sources, and include Active Queue Management (AQM) schemes such as RED, SRED, and FRED [FJ93]. AQM probabilistically drops packets before buffer overflow to prevent extreme congestion and “phasing”, and to enhance fairness. Also in this category are link layer protocols sheltering the sender from random local loss; e.g., Snoop, Split Connection protocols, and the like [BPSK97]. Active Feedback schemes include ECN (Explicit Congestion Notification) which explicitly informs the source of congestion at some router [Floy94]; and ELN (Explicit loss Notification) which informs the source that a packet was lost because of reasons other than network congestion [BK98].

The TCP Westwood enhancement proposed in this paper does not invoke (as is the case also with the original TCP Westwood scheme) any changes/support at lower layers. Both the original TCP Westwood [CGLMS00][CGMSW01] design and its proposed enhancements here adhere to the end-to-end transparency guidelines set forth in [Clar88], and require only sender side modification. This does not mean that TCP Westwood cannot benefit from lower layer interventions. In fact, preliminary studies show that TCPW works well with AQM schemes such as RED, and is expected to benefit also from ECN. Lower level intervention is not required, however, to achieve a satisfactory level of efficiency and fairness.

We recall that the key innovation of the original TCPW is to use the bandwidth estimate directly to drive *cwin* and *ssthresh*. This is in contrast with TCP Vegas and Packet Pair rate control schemes which use the measured rate to estimate bottleneck backlog first; and then adjust *cwin* based on the backlog. The original estimation method in TCPW attempts to estimate the bandwidth share available to the connection. The resulting strategy, referred to in the following as TCPW BE, provides significant throughput gains, especially for large leaky pipes [CGMSW01]. Under certain load conditions, however, BE exceeds the “fair share” of the bottleneck bandwidth available to a connection resulting in

possible unfriendliness to other TCP varieties, for example TCP NewReno connections. This friendliness/efficiency tradeoff is the main focus of this work.

We present an extension of the original TCPW in which two estimators are maintained, along with a method to identify the predominant cause of packet loss. Depending on the loss cause, the appropriate estimator is "adaptively" selected. Both estimators use information obtained from ACKs received at the sender. One estimator, as in the standard TCPW, called Bandwidth Estimator (BE) considers each ACK pair separately to obtain a bandwidth sample, filters the samples into a low-pass filter and returns as a result the (short term) bandwidth share that the TCP sender is getting from the network. The other estimator, proposed in this paper for the first time and called Rate Estimator (RE), measures the amount of data acknowledged during the latest interval  $T$ . It then feeds such data sample into an appropriate low-pass filter to get the estimated rate. Thus, RE tends to estimate the (relatively longer term) rate that the connection has recently experienced. We call the TCPW refinement that adaptively selects one or the other method TCPW CRB (Combined Rate/Bandwidth-Estimation).

A brief digression on random loss versus congestion loss discrimination is in order. To identify the predominant cause of packet loss, ECN and ELN could be used. However, ECN requires all the routers along a network path to support ECN, while ELN has its share of implementation problems as reported in [BPSK97]. We have selected instead a method that does not require support from lower layers. Namely, we use the relationship between the current congestion window value and the estimated pipe size, the latter being defined as the product of RE and the minimum RTT observed. The pipe size corresponds to the ideal window required to achieve the measured rate RE. When the measured pipe size is significantly smaller than  $cwin$ , it is very likely that packet losses are due to congestion. An analysis of this relationship and a simulation study to evaluate the efficacy of this method are presented in Section 5.

The remainder of paper is organized as follows. In Section 2 we overview TCP Westwood and the Bandwidth Estimator (BE). In Section 3 we introduce the proposed estimator—Rate estimator (RE). In Section 4 we use analysis along with simulation results to determine the semantics and the accuracy of the estimates they provide, and explain the differences between the two. In Section 5 we present our CRB algorithm, and the method to identify the more likely cause of packet loss and its use in selecting an estimator for the present window adjustment. Section 6 presents a performance study under different network environments obtained by changing system parameters like: link error rates, end-to-end propagation time, router buffer space, and bottleneck link speed. A study on fairness, friendliness and link utilization is also presented. Finally, in Section 7, we provide concluding remarks and identify areas of future research.

## 2. Overview of TCP Westwood and the Bandwidth Estimation (BE)

TCP Westwood design follows the end-to-end principle [Clar88]. The network is considered a "black box", providing no support to the TCP congestion control scheme. The key innovative idea in TCP Westwood is that the TCP sender continuously computes the connection Bandwidth Estimate (BE) which is estimating the bottleneck bandwidth currently available to the connection. Thus, BE also represents the share of the bottleneck bandwidth - and therefore the rate currently used by the connection. The bandwidth estimate is based on bandwidth samples collected by the source as the ACKs are received. After a packet loss indication, which could be due to either congestion or link errors, the sender uses the estimated available bandwidth to properly set the congestion window and the slow start threshold. Packet loss symptoms used by a sender are the reception of 3 duplicates, or a coarse timeout expiration.

Another important element of the TCPW algorithm beside BE is the RTT estimation on each connection. The RTT is required to compute the window that drives the connection to the estimated rate BE. Ideally, the RTT should be measured when the bottleneck is empty. In practice, this is not possible. Thus, in our proposed scheme RTT is the overall minimum round trip delay measured on the connection so far (based on continuous monitoring of ACK RTTs).

### 2.1 Setting $cwin$ and $ssthresh$ in TCPW

Let us assume that a sender has determined the connection bandwidth estimate BE (details on this estimation are provided in Section 2.2). In this section we describe the use of BE in setting  $cwin$  and  $ssthresh$  after a packet loss indication.

First, we note that in TCPW, congestion window dynamics during slow start and congestion avoidance are unchanged, that is they increase exponentially and linearly, respectively, as in current TCP Reno and NewReno.

A packet loss is indicated by (a) the reception of 3 DUPACKs, or (b) a coarse timeout expiration. In case the loss indication is 3 DUPACKs, TCPW sets *cwin* and *ssthresh* as follows:

```

if (3 DUPACKs are received)
  ssthresh = (BE * RTTmin) / seg_size;
  if (cwin > ssthresh) /* congestion avoid. */
    cwin = ssthresh;
  endif
endif

```

In the pseudo-code, *seg\_size* identifies the length of a TCP segment in bits. Note that the reception of *n* DUPACKs is followed by the retransmission of the missing segment, as in the standard Fast Retransmit implemented by TCP Reno. Also, the window growth after the *cwin* is reset to *ssthresh* follows the rules established in the Fast Retransmit algorithm (i.e., *cwin* grows by one for each further ACK, and is reset to *ssthresh* after the first ACK acknowledging new data). During the congestion avoidance phase we are probing for extra available bandwidth. Therefore, when *n* DUPACKs are received, it means that we have hit the network capacity (or that, in the case of wireless links, one or more segments were dropped due to sporadic losses). Thus, the congestion window and slow start threshold are set equal to the window capable of producing the estimated rate BE when the bottleneck buffer is empty (namely, window =  $BE * RTTmin$ ). The congestion avoidance phase is entered to gently probe for new available bandwidth. The value *RTTmin* is set as the smallest RTT estimated by TCP, using its own RTT estimation algorithm. Note that after *ssthresh* has been set, the congestion window is set equal to the slow start threshold only if  $cwin > ssthresh$ . This is the most common situation. It is possible however that the current *cwin* may be below threshold. This occurs after time-out for example, when the window is dropped to 1 and a random loss occurs during slow start. In this case we do not reset the window but allow it to continue its exponential increase.

In case a packet loss is indicated by a timeout expiration, *cwin* and *ssthresh* are set as follows:

```

if (coarse timeout expires)
  cwin = 1;
  ssthresh = (BE * RTTmin) / seg_size;
  if (ssthresh < 2)
    ssthresh = 2;
  endif;
endif

```

The rationale of the algorithm above is that after a timeout, *cwin* and the *ssthresh* are set equal to 1 and BE, respectively. Thus, the basic Reno behavior is still captured, while a speedy recovery is ensured by setting *ssthresh* to the value of BE.

## 2.2 Bandwidth Estimation (BE)

The TCPW sender uses ACKs to estimate BE. More precisely, the sender uses the following information: (1) the ACK reception rate and, (2) the information an ACK conveys regarding the amount of data recently delivered to the destination.

We discuss the use of the information in (2) above in the next sections. For now, let assume that an ACK is received at the source at time  $t_k$ , notifying that  $d_k$  bytes have been received at the TCP receiver. We can measure the *sample* bandwidth available to that connection as  $b_k = d_k / (t_k - t_{k-1})$ , where  $t_{k-1}$  is the time the previous ACK was received.

Letting  $\Delta t_k = t_k - t_{k-1}$ , then  $b_k = d_k / \Delta t_k$ .

One can easily verify that the sample bandwidth computation is consistent with existing bandwidth estimation techniques (eg, Packet Pair flow control [Kesh91]). In particular, it provides a measure of the fraction of bottleneck bandwidth available to the connection. If the connection is the sole user of the bottleneck and the two ACKs in question belong to the same window cycle, the sample bandwidth is an accurate estimate of total bottleneck bandwidth. Note that the sample bandwidth is generally different from the actual rate achieved by the connection (as discussed in the next section).

Since congestion occurs whenever the low-frequency input traffic rate exceeds the available bottleneck link capacity, we employ a low-pass filter to average sampled measurements and to obtain the low-frequency components of the available bandwidth. Notice that this averaging is also useful in filtering out the noise due to delayed acknowledgments.

In our early design and experimentation, we used a filter similar to the one used for RTT estimation in TCP. We determined that such an exponential filter with constant coefficients is not capable of efficiently filtering out high-frequency components of the bandwidth measurements. We propose the following filter which is a discrete version of a continuous first order low-pass filter using the Tustin approximation [Witt97].

Let  $b_k$  be the bandwidth sample, and  $\hat{b}_k$  the filtered estimate of the bandwidth at time  $t_k$ . Let  $\alpha_k$  be the time-varying exponential filter coefficient at  $t_k$ . The TCPW BE filter is then given by  $\hat{b}_k = \alpha_k \hat{b}_{k-1} + (1 - \alpha_k) \left( \frac{b_k + b_{k-1}}{2} \right)$ , where

$$\alpha_k = \frac{2\tau - \Delta t_k}{2\tau + \Delta t_k}, \text{ and } 1/\tau \text{ is the filter cut-off frequency.}$$

Notice the coefficients  $\alpha_k$  are made to depend on  $\Delta t_k$  to counteract the effect of non-deterministic interarrival times. In fact, when the interarrival time  $\Delta t_k$  increases, the last value  $\hat{b}_{k-1}$  should have less significance. On the other hand, a recent  $\hat{b}_{k-1}$  should be given higher significance. The coefficient  $\alpha_k$  decreases when the interarrival time increases, and thus the previous value  $b_{k-1}$  has less significance with respect to the last two recent samples which are weighted by  $(1 - \alpha_k)/2$ .

As an example, let  $t_k - t_{k-1} = \Delta_k = \tau/10$ . Then,

$$\hat{b}_k = \frac{19}{21} \hat{b}_{k-1} + \frac{2}{21} \left( \frac{b_k + b_{k-1}}{2} \right)$$

The new value  $\hat{b}_k$  is thus made up of approximately 90 % of the previous value  $\hat{b}_{k-1}$  plus approximately 10 % of the arithmetic average of the last two samples  $b_k$  and  $b_{k-1}$ .

Since the TCPW filter has a cut-off frequency equal to  $1/\tau$ , all frequency components above  $1/\tau$  are filtered out. According to the Nyquist sampling theorem, in order to sample a signal with bandwidth  $1/\tau$  a sampling interval less than or equal to  $\tau/2$  is necessary. But, since the ACK stream is asynchronous, the sampling frequency constraint cannot be guaranteed. Thus, to guarantee the Nyquist constraint, we establish that if a time  $\tau/m$  ( $m \geq 2$ ) has elapsed since the last received ACK without receiving any new ACK, then the filter assumes the reception of a *virtual* sample  $b_k = 0$ . The situation is shown in Figure 1, where  $t_{k-1}$  is the time an ACK is received,  $\hat{t}_{k+j}$  are the arrival times of the virtual samples, with  $\hat{t}_{k+j+1} - \hat{t}_{k+j} = \tau/m$  for  $j = 0, n-1$ ; and  $b_{k+j} = 0$  for  $j = 0, n-1$  are the virtual samples.

Then,  $b_{k+n} = \frac{d_{k+n}}{\Delta t_{k+n}}$  is the bandwidth sample at  $t_{k+n}$ .

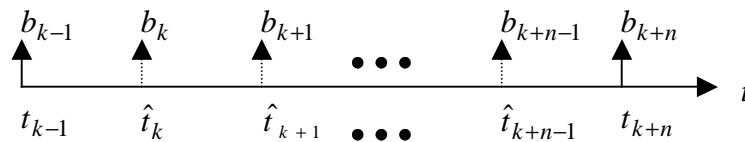


Figure 1. Bound on the maximum sampling interval obtained by inserting virtual samples.

It is desirable that after a long time without ACKs (i.e., because no new data were sent), the filter acts in a conservative fashion, progressively decreasing the bandwidth estimation as time elapses without new samples. Consider thus the operation of the TCPW filter when there is prolonged absence of ACKs after a time  $t = t_k$ . As can be seen from the following sequence of bandwidth estimates, the estimated bandwidth exponentially goes to zero:

$$\hat{b}_k = \frac{2m-1}{2m+1} \hat{b}_{k-1} + \frac{0+b_{k-1}}{2m+1}$$

$$\hat{b}_{k+1} = \frac{2m-1}{2m+1} \hat{b}_k$$

.....

$$\hat{b}_{k+h} = \left(\frac{2m-1}{2m+1}\right)^h \hat{b}_k$$

### 2.3. ACK Sequence Number Processing

A number of considerations must be taken into account while interpreting the information that a returning ACK carries regarding delivery of segments to the destination. The following summarizes these considerations:

1. When an ACK is received by the source, it implies that a transmitted packet was delivered to the destination. A DUPACK also implies that a transmitted packet was delivered, triggering the transmission by the receiver of the DUPACK. Thus, a DUPACK should not be ignored when estimating bandwidth. A DUPACKs impacts the bandwidth estimate, and a new estimate should be computed right after its reception.
2. Since TCP ACKs can be “cumulative”, i.e., an ACK indicates the reception of a complete sequence of bytes regardless of which segments they were delivered in, care must be taken so that the correct amount of data bytes delivered is accounted for, and not accounted for more than once.
3. TCP ACKS can be “delayed,” i.e., receivers wait for a second packet before sending an ACK, unless 200 ms elapse in which case an ACK is sent without waiting. Delayed ACKs must be carefully processed as well.
4. A source is in no position to tell for sure which segment triggered the DUPACK transmission, and it is thus unable to update the data count by the size of that segment. An average of the segment sizes sent thus far in the ongoing connection should therefore be used.

The above considerations have been taken into account in designing the algorithm *AckedCount*. *AckedCount* is intended to produce an accurate estimate of the data delivered at a destination. When it is difficult to be exact, we take a conservative approach ensuring that the estimated bandwidth is no greater than the actual connection bandwidth.

### 3. Rate Estimation (RE)

Significant efficiency improvements have been obtained by TCPW using the bandwidth estimate (BE) produced by the sampling and filtering methods above. This is particularly true in environments with large leaky pipes. Further, note that when routers employ a round robin policy in scheduling transmissions, BE is accurate in estimating a connection fair share. However, in drop-tail routers TCP traffic has been observed to be “bursty”, i.e. sending out a full window of packets and then waiting for the acknowledgements. In this situation, competing connections may “take turns” in injecting their windows into the network, with the result that each basically sees the bottleneck as dedicated to itself and thus tends to over-estimate its fair share. The problem clearly stems from the bandwidth sample definition introduced in the previous section.

Consider an alternative available bandwidth sample, defined as the amount of data reported to be delivered by all ACKs that arrived in the last T time units, divided by T. As explained later in more detail, this measurement corresponds to the rate actually achieved recently by the connection over time T. Thus, we call this method Rate Estimation (RE). This alternative is identical to the earlier TCPW sample definition if the ACKs are *uniformly spaced* in time. Simulation and measurements, however, show that ACKs tend to cluster in bursts. Thus, the BE sampling method “overestimates” the connection fair share, while providing (in the bursty case) a reasonably good estimate of the *instantly available bandwidth* at the bottleneck. It turns out that BE is more effective in environments where friendliness (to other TCP connections) is not of concern, for instance, in the case of single connection operation or when random error/loss is significant and TCP NewReno is unable to take its fair share.

Let us define the Rate Estimation sample. The RE sample associated with the  $k^{\text{th}}$  received ACK is expressed by:

$RE_k = \frac{\sum_{t_j > k-T} d_j}{T}$ , where  $d_j$  is the amount of data reported by ACK  $j$ . Similarly, at the previous time instant,  $k-1$ , the sample  $k-1$  is:

$$RE_{k-1} = \frac{\sum_{t_j > k-1-T} d_j}{T}$$

Therefore,

$$RE_k - RE_{k-1} = \frac{1}{T} \left( \sum_{t_j > k-T} d_j - \sum_{t_j > k-1-T} d_j \right); \text{ which on rearrangement gives,}$$

$RE_k = RE_{k-1} + \frac{d_k - \sum_{t_j < k-T, t_j > k-1-T} d_j}{T}$ . Thus, at any instant, a “sliding window” of length  $T$  is used to obtain a bandwidth sample.

The expression above is a recursive one, because the sample is calculated using its previous value as reference. Additionally, the technique places equal emphasis on all data points in the sampling range. Thus a value in the near past will have the same influence as a more current measurement when calculating the sample. This is a desirable feature when we are dealing with bursty TCP traffic in presence of congestion. Finally, sliding window samples are exponentially averaged in order to obtain a smoothed bandwidth share estimate over time. We use here the same filter as introduced in Section 2.2 to calculate the Rate Estimate at the instant the  $k^{\text{th}}$  ACK is received as:

$$\hat{RE}_k = \alpha_k \hat{RE}_{k-1} + (1 - \alpha_k) \left( \frac{RE_k + RE_{k-1}}{2} \right), \text{ where } \alpha_k = \frac{2\tau - \Delta t_k}{2\tau + \Delta t_k},$$

Note that the choice of  $\tau$  and the ACK sequence number processing for this filter is similar to that introduced in Section 2.3.

#### 4. Comparison of BE and RE Estimates

In this section we discuss the two aforementioned sampling methods (BE and RE) and compare their accuracy under different conditions. It is worth noting that both methods are passive measurement schemes carried out only on the TCP sender side. Compared to other active network estimation methods [CC96, DJ02], TCPW estimation methods will be more useful for actual deployment over the Internet. Moreover, the goal of TCPW is not to obtain physical link bandwidth nor available bandwidth. Rather, TCPW aims to estimate an eligible sending rate for the connection that improve total link utilization while remaining friendly to coexisting NewReno connections.

To start with, we provide a definition of fair bandwidth share. We then analyze the two sampling methods, and evaluate their estimates relative to the fair share estimate. TCP design aims to utilize all available bandwidth, while maintaining “fairness” in the allocations made to different flows. In this paper, we focus on bandwidth fair sharing of TCPW and TCP NewReno, both among themselves and against each other. To focus on the essential behavior of TCPW and TCP NewReno, we use the standard model where TCP flows share a single bottleneck link with bandwidth  $C$ . We also assume infinite backlog for all TCP sources, and assume that all flows have identical propagation delays. Thus fairness is achieved by equally allocating the available bandwidth to active TCP flows, unless some of them are inherently unable to use their share regardless of the existence of competing flows. For instance, on a “leaky” large pipe, NewReno utilization is dramatically reduced [LM95]. In this case, a flow using a new proposed protocol can achieve higher bandwidth share and preserve fairness. However, this should be accomplished without reduction in the legacy connections throughput.

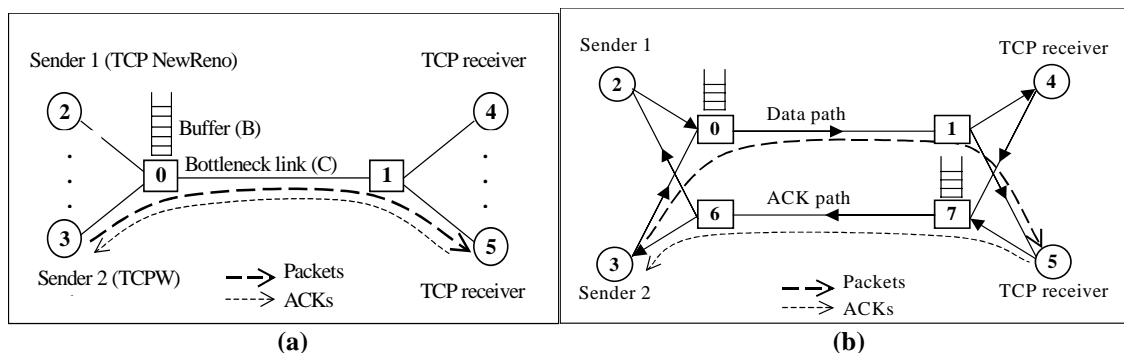
We define the fair bandwidth share of a connection for the following cases:

- a)  $N$  TCPW flows sharing a bottleneck link with capacity  $C$ . The fair share is  $C/N$ .
- b) A total of  $N$  TCPW and TCP NewReno flows. Assuming that no random errors, both protocols should be able to deliver the same throughput, and therefore the fair share for each flow is  $C/N$ .

- c) Assume that there are random errors on the paths (e.g. from a wireless link) to which all flows are subjected. It is well known that TCP NewReno flows are inherently unable to utilize the full link capacity in this case: TCPW flows should not be penalized to force the same share as New Reno, ie they should not be considered too aggressive simply because they get a larger bandwidth share than the NewReno flows. We define the fair share of a NewReno flow in a mixed traffic environment as the value achieved if all flows were TCP NewReno. For instance, suppose the fair share of the NewReno flow is  $S_r$  given that there are total  $N$  homogenous TCP NewReno flows, then when this total  $N$  flows includes some TCPW flows, the fair share of NewReno flow should remain  $S_r$ , while TCPW flows could have a fair share of higher value. Thus, a TCPW fair share can be higher than a NewReno share when the latter connection is incapable of using the full link capacity.
- d) In the presence of CBR type flows (e.g, real time streaming traffic) which take away a portion of bandwidth determined by their transmission rates, we just reduce the capacity available to TCP connections by the amount used by the non-adaptive flows and proceed to calculate fair share of a TCPW connection as in the above three cases.

In TCPW, we proposed a Bandwidth Estimation (BE) [CGMSW01] algorithm used by a TCPW sender to drive its *cwnd* and *ssthresh*. Let us call this version of TCP, TCPW BE. This protocol has been shown to achieve a high utilization when used over large leaky pipes. As we will show below, in certain cases, BE may overestimate its fair share. In this case, TCP NewReno (and other TCP-like protocols) may experience performance degradation when coexisting with TCPW. For this reason we introduce and study here TCPW RE (Rate Estimation) to address the issue of friendliness to NewReno. The objective of the remainder of this section is to present and compare the two estimations and explain the rationale behind them. Both estimations are based on the ACK arrival process received by the TCPW sender; thus, they are passive and introduce no extra link overhead.

To capture the essence of the two estimators, we use the network topologies shown in Figure 2. The bottleneck link is shared by  $N$  TCP flows including NewReno flows and TCPW flows. Flows are assumed to have identical propagation delay and infinitely backlogged sources.



**Figure 2 Network topology**

Note that in Figure 2.a, there is a single path in which ACKs and data packet flow. This is the standard configuration typically simulated for TCP congestion control studies. It is possible of course that the ACKs actually return on a different path as in Figure 2.b. We note that the sender estimation based on ACK information should take into account the entire loop, that is both the forward and reverse paths. And we also note that the bottleneck link can be on either the forward or the reverse path. The entire loop is relevant and the bottleneck on either path is relevant because a TCP sender can make progress in data transfer only if it receives ACKs. Further, the ACK packet length is irrelevant as far as calculating bandwidth share. What is significant is what an ACK *reports* as the amount of data bytes delivered to the destination.

The ACK arrival process at node 3 is affected by two different factors: first, it is related to the sending process. Second, it is influenced by the queuing delays at the bottleneck router. Such delays depend on both the scheduling policy used by the router and the amount of “background” traffic, in this case, TCP NewReno flow sent from node 2. We assume the scheduling to be first-come-first-serve, which is the policy found on most routers today.

For a TCP flow, the sender tends to receive the ACKs in bursts due to the following two reasons: first, the sender sends packets in bursts as a consequence of the slow-start phase of the TCP protocol, in which packets belonging to the same window will be sent close to one another, while packets belonging to different windows will be separated. Second, the

first-come-first-serve policy at the router can introduce further gaps (or increase the existing gaps) between groups of packets of the same flow having to wait for packets from other flows to be served by the router.

Let us assume that the TCPW sender will receive the  $k$ -th ACK at time instant  $t_k$ , confirming that  $d_k$  bytes are correctly received by the receiver. Let us define  $\Delta t_k$  the inter-ACK time elapsed between the  $(k-1)$ th and the  $K$ -th ACK (same as in Section 2.1).

First consider that if two packets are sent back to back by the TCPW sender and not separated due to background traffic, then the corresponding ACKs would come back to the sender with an inter-ACK delay  $\Delta t_k$  such that  $b_k = \frac{d_k}{\Delta t_k}$

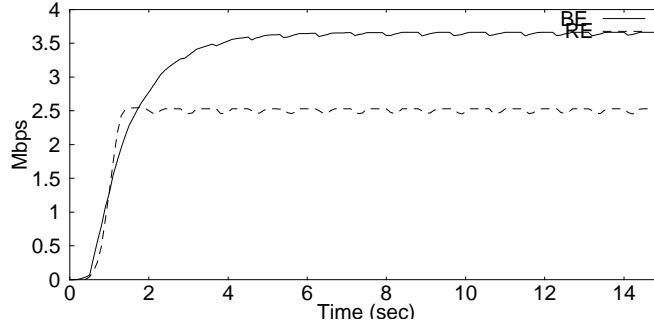
will be equal to  $C$ , the bottleneck bandwidth. Note that  $b_k = \frac{d_k}{\Delta t_k}$  is exactly the sample used by the BE estimator.

In the case of multiple connections, the ACKs would likely arrive in bursts separated by gaps during which the server was occupied by other flows. For this reason the filter will receive bursts of bandwidth samples  $b_k \cong C$  interleaved by other lower bandwidth samples, and the total effect is that the BE may overestimate its fair share on the bottleneck link.

In RE, a sample is calculated by dividing by  $T$  the number of bytes acknowledged in an interval of length  $T$ . The RE sample is thus a smoother sample, and thus it produces an estimate that is closer to the *throughput* actually achieved by the TCPW connection.

#### 4.1. Estimation With No Random Errors on the Paths

In figure 3, we first compare the BE and RE estimations when there are no random errors on the paths. The graph was obtained applying the BE and RE estimators to TCPW. The network configuration in Fig 2 is used, with one TCPW flow and one NewReno flow sharing the bottleneck. The following parameters apply: bottleneck bandwidth  $C = 5$  Mbps, packet size = 1400 bytes (fixed), round trip time  $RTT = 70$  ms, buffer size ( $B$ ) = pipe capacity which is equal to 32 packets; time interval  $T = 4$  RTT. The link is error free and no packet is dropped in initial seconds due to congestion.



**Figure 3. BE and RE with concurrent TCP NewReno connection**

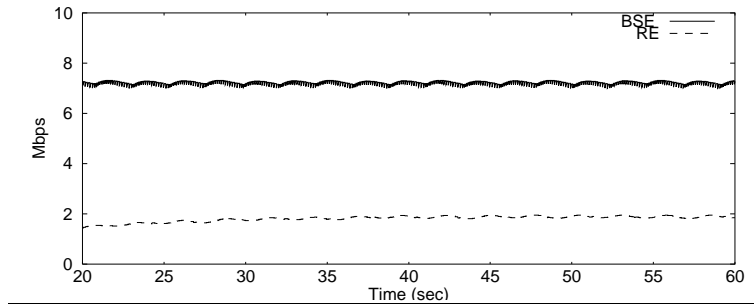
As seen in figure 3, the BE is estimating about 3.6 Mbps which is larger than its 2.5 Mbps fair share, while the RE estimator is estimating exactly the fair share value.

As we mentioned before, when the bottleneck is congested the BE and RE estimations would coincide if a round-robin scheduling policy was used by the router: in this case packets from the two flows would be served by the router alternately, the corresponding ACKs would come back to the sender equally spaced (except for the gaps due to the slow start phase) and the BE estimate would coincide with the recent throughput estimate RE.

In conclusion, the BE estimator tends to overestimate the fair share because of the effects of the samples generated by the bursts of packets served back-to-back by the router. The RE estimator gives a more accurate measure of the fair share in this case since its estimate reflects the throughput recently achieved by the flow. On the other hand, RE may underestimate the connection fair share if the path suffers from random errors and the bottleneck is not fully used., We will analyze this case by simulation in Section 4.2 and 4.3.

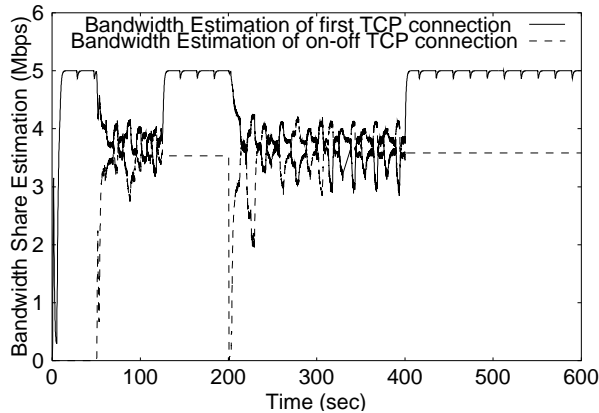
The above simulation experiments use a simple 2-flow scenario to reveal BE and RE behavior under congestion. To examine the estimation methods under more realistic conditions, we also run simulations with multiple connections sharing a bottleneck. Figure 4 shows BE and RE estimates when there are one TCPW flow and 4 NewReno flows

sharing a 10Mbps bottleneck. The same trend as in Figure 3 (BE overestimates while RE oscillates around fair share) is observed. More extensive evaluation of TCPW performance under multiple connections is carried out in Section 6.3.

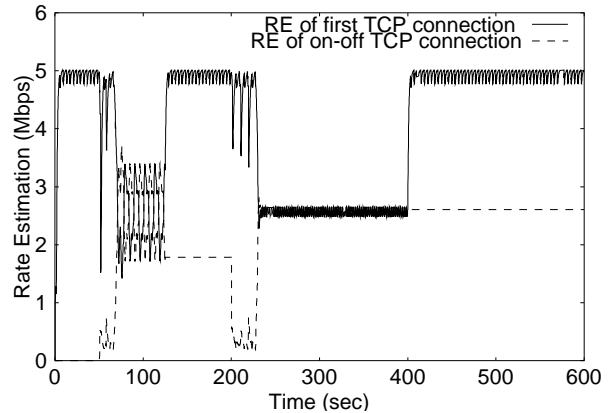


**Figure 4. BE and RE with multiple NewReno connections**

The next simulation experiments include two TCPW connections sharing an error free bottleneck link. To illustrate the dynamic accuracy of the estimates, we maintain one connection active throughout the simulation run, while the other one is turned ON and OFF twice during the simulation experiment. From Figure 5, we can see that when both connections are running, BE is 3.1Mbps, an overestimate of the fair share (2.5 Mbps.) Figure 6 shows that the RE provides an estimate equal to the fair share (2.5 Mbps). It is important to note that in either case the throughputs obtained by two TCPW connections are fair (ie, they are identical when both connections are on) since the two connections use the same estimates, either RE or BE. The estimates oscillate due to the TCP's bursty transmission. An interesting observation here is that the connection starting second readily acquires its fair share against the established connection.



**Figure 5. Bandwidth Estimation with concurrent TCP**



**Figure 6. Rate Estimation with concurrent TCP**

The results reported above in Figures 3-6, all confirm that RE is more appropriate than BE when the bottleneck is congested and packet loss is caused only by congestion.

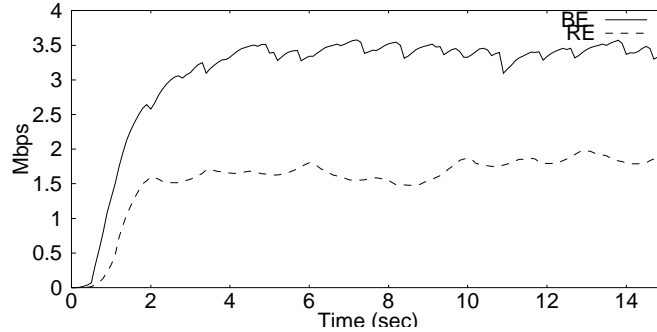
#### 4.2. Estimation in Presence of Random Errors on the Path

We present simulation results to study scenario (c) defined earlier in this Section (ie, random errors). We run one TCPW and one NewReno flow sharing a bottleneck link with random error rate of 0.5%. Network configuration is the same as reported in the experiments in Figs 3, 5 and 6. Estimates provided by BE and RE are shown in Figure 7, and Table 1 lists the average estimates and throughputs for 2 TCP flows sharing the bottleneck link.

The results in Figure 7 and Table 1 confirm that BE should be used when random error is the cause of packet losses. First, note that the fair share of TCPW is actually larger than  $C/2 = 2.5$ Mbps in this case, since NewReno is inherently unable to utilize the link capacity. Through simulation (see Table 1), when two NewReno flows share the link, the throughput obtained by each flow is 1.4Mbps. Thus, in this configuration, the TCPW flow can take as much as 3.6 Mbps out of the total 5 Mbps without harming NewReno performance. From Figure 6, RE estimate settles at about 1.8 Mbps, underestimating its fair share, while BE estimates a more accurate share at 3.4 Mbps.

The throughput results also confirm that using BE is better in this case, where random errors dominate as the cause of packet loss. Note that the throughput achieved by a TCPW connection is typically different from the share estimate

although they are closely related—estimates are used to set the *cwnd* and *ssthresh* after a packet loss event. From Table 1, TCPW BE and TCP RE obtain 2.42Mbps and 1.97Mbps, respectively, while the competing TCP NewReno sustains a 1.4Mbps which is equal to the fair share when two NewReno flows share the link. Thus, both TCP BE and RE are friendly to New Reno. In addition TCP BE is more efficient than TCP RE.



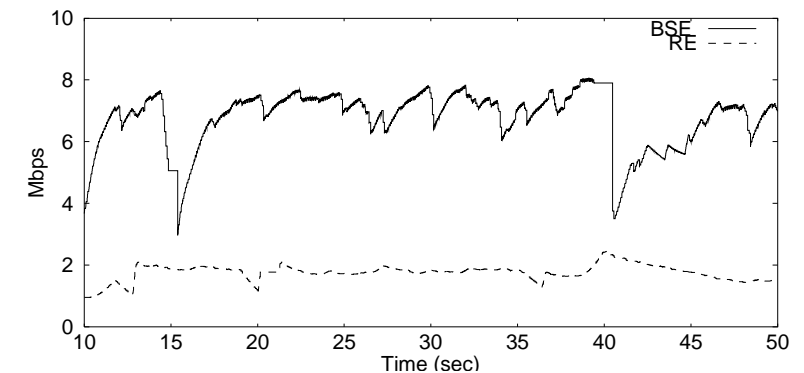
**Figure 7. BE and RE with concurrent NewReno connection (error rate 0.5%)**

Estimate(Mbps)	BE	NewReno & NewReno	TCPW BE & NewReno	TCPW RE & NewReno
	RE	-----	-----	3.4
Throughput(Mbps)	NewReno	1.4	1.4	1.4
	TCPW BE	-----	2.42	-----
	TCPW RE	-----	-----	1.76
	Total	2.8	3.82	3.16

**Table 1. Estimate and throughput (2 TCP flows sharing the bottleneck link)**

Again, we examine BE and RE estimates in the multiple-connection scenario. We run simulations with 1 TCPW and 4 NewReno flows sharing a bottleneck link of 10Mbps with random error rate of 1%. Estimates provided by BE and RE are shown in Figure 8; and Table 2 lists the average estimates and throughputs for the TCPW flow, and average throughputs for NewReno flows.

Similar to the previous experiment, the fair share of TCPW is actually larger than  $C/5 = 2\text{Mbps}$  in this case. Through simulation (see Table 2), when 5 NewReno flows share the link, the average throughput obtained by each flow is 1.1Mbps. Thus, in this configuration, the TCPW flow can take as much as 5.6Mbps out of the total 10Mbps without harming NewReno performance. Figure 8 shows that RE estimates settle way below the potential fair share, while BE achieves a higher estimate. The throughput results also confirm that using BE is better in the random error case. From Table 2, TCPW BE and TCPW RE obtain 2.42Mbps and 1.97Mbps, respectively, while the competing TCP NewReno sustains a 1.2Mbps which is equal to the fair share when five NewReno flows share the link. Thus, both TCPW BE and RE are friendly to New Reno in this random error case. In addition TCPW BE is more efficient than TCPW RE.



**Figure 8. BE and RE with multiple concurrent NewReno connections (error rate 1%)**

Estimate(Mbps)	BE	All NewReno	TCPW BE & NewReno	TCPW RE & NewReno
	RE	-----	-----	6.7
	NewReno	1.1	1.04	1.12

Throughput(Mbps)	TCPW BE	-----	3.62	-----
	TCPW RE	-----	-----	1.78
	Total	5.49	7.77	6.27

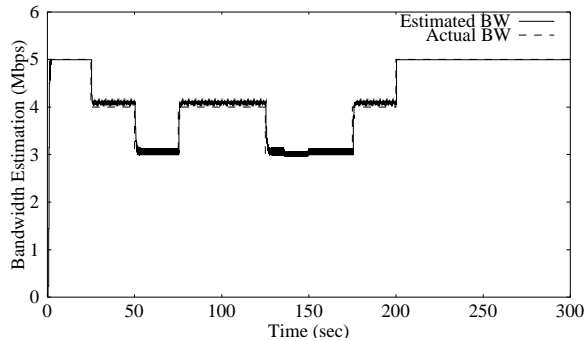
**Table 2. Estimate and throughput ( 5 TCP connections sharing the 10Mbps bottleneck link)**

### 4.3. Estimation When Available Bandwidth Fluctuates Due to Non-Adaptive Traffic

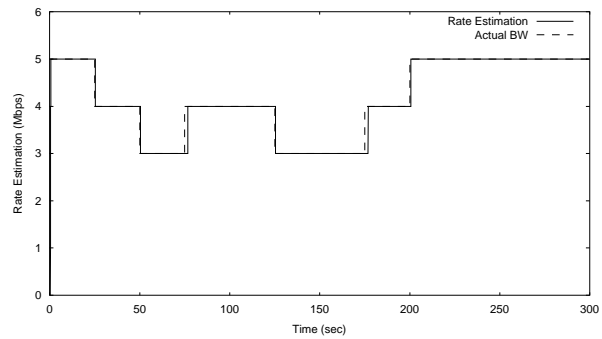
Using the same network configuration, we first examine how the two sampling methods perform when a single TCPW connection shares a bottleneck link with non-adaptive UDP traffic with time varying intensity. The UDP traffic is used here to simulate non-adaptive high priority traffic. Both BE and RE are able to track the bandwidth left unused by the UDP traffic. The actual available bandwidth and the estimated connection share are shown in Figure 9 and Figure 10. The results show that both methods track the available bandwidth accurately, with RE providing a more stable tracking. This is because TCPW is the only adaptive traffic without competition (note the UDP rate is fixed) in this case, which makes TCP behaves as a cyclic process with a cycle of RTT. The Sender sends a window full of packets every RTT then wait for the acknowledgement. The BE sample is very sensitive to whether the packets are clustered, but the RE sample is relatively less sensitive.

Figure 11 and Table 3 illustrates the results for a link shared by one NewReno, one TCPW, and one non-adaptive UDP flow with a fixed rate of 2Mbps. The shared bottleneck is the same as that in figure 7, with capacity of 5Mbps and 0.5% link error rate. Results in Table 3 show that the fair share a NewReno flow can get is 1.1 Mbps when two NewReno flows share the link with the UDP flow. Thus, the fair share a TCPW flow can get without harming NewReno equals  $5 - 2 - 1.1 = 1.9$  Mbps. RE estimates fluctuate near 1.3Mbps, under its fair share, while BE has more accurate estimates at around 2.0Mbps.

Again we look at the throughputs achieved by TCPW and NewReno flows. From Table 3, when a TCPW RE flow and a NewReno flow share the link with the UDP flow, the throughputs obtained are 1.15Mbps and 1.1Mbps respectively. Thus, in this case, TCP RE preserves perfect friendliness (recall that the throughput a NewReno flow can get without competing with TCPW is 1.1Mbps) but the total utilization of the link is only slightly enhanced. On the other hand, when TCPW BE flow shares the link with a NewReno flow and the UDP flow, it gets 1.34Mbps, increasing total utilization significantly, while the competing TCP NewReno flow get 1.07Mbps only slightly down from its 1.1Mbps. Therefore, the BE method is the better method to use in this case, and in fact, in all cases when random link errors are the predominant cause of packet losses. These results are consistent with the results in Fig 7.



**Figure 9. Bandwidth Estimation (BE) with concurrent UDP traffic**



**Figure 10. Rate Estimation (RE) with concurrent UDP traffic**

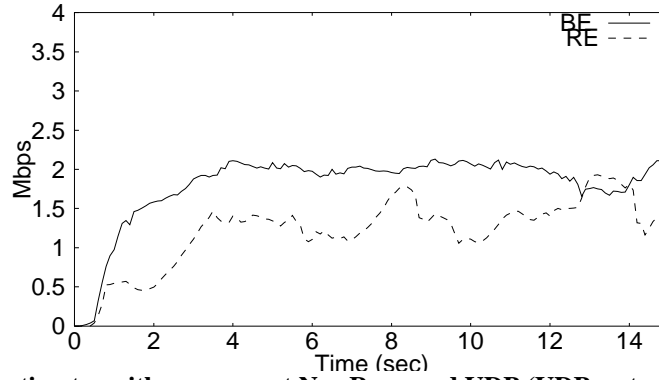


Figure 11. BE and RE estimates with concurrent NewReno and UDP (UDP rate =2Mbps; error rate 0.5%)

		NewReno & NewReno	TCPW BE & NewReno	TCPW RE & NewReno
Estimation(Mbps)	BE	-----	2.0	-----
	RE	-----	-----	1.3
Throughput(Mbps)	NewReno	1.1	1.07	1.1
	TCPW BE	-----	1.34	-----
	TCPW RE	-----	-----	1.15
	Total	2.2	2.41	2.25

Table 3. Estimate and throughput (2 TCP flows and a UDP flow of 2Mbps sharing the bottleneck link)

## 5. Combined RE/BE (CRB) Method

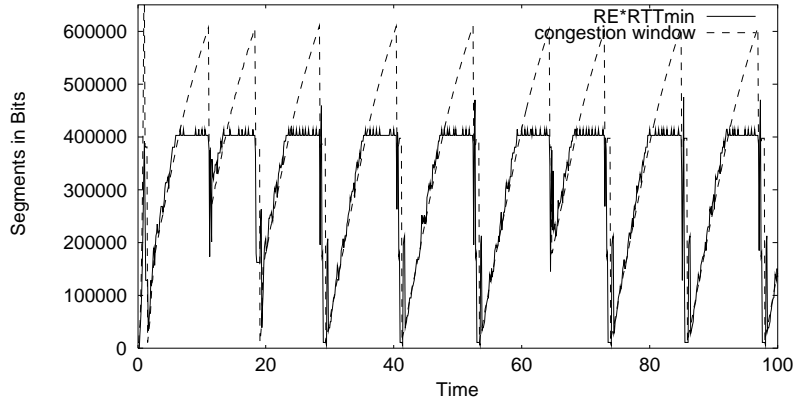
We addressed above the question of which estimate – achieved rate or available bandwidth – yields better bandwidth estimates. BE is more effective in environments with random error. On the other hand, the RE method appears to be more appropriate when packet losses are due to congestion (ie, router buffer overflow).

The above results suggest that a hybrid method that combines both sampling strategies would be of interest, provided we can determine the cause of packet loss: errors or buffer overflow. This issue of course is beyond the sampling methodology investigation itself. In fact, if a sender were able to distinguish with certainty between error and buffer overflow losses, it could easily devise an optimal transmission policy, namely: in case of error, retransmit immediately with no change of window size. In case of congestion, use the conventional TCP W algorithm.. For our immediate purposes, it suffices to be able to determine the predominant cause of loss, and then select the sampling strategy accordingly. Below we present a method to determine the predominant cause of packet loss requiring no assistance from layers below TCP.

### 5.1. Identifying Predominant Cause of Packet Loss

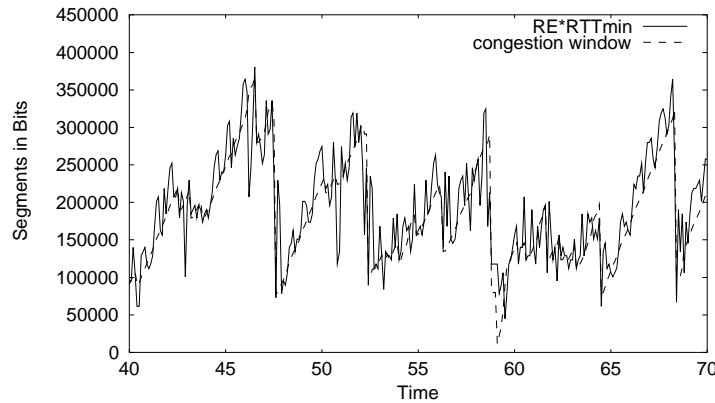
We propose to use the relationship between the current *cwin* value and the estimated pipe size, the latter indicated by the product of RE and the minimum RTT. When  $RE * RTT_{min}$  is significantly smaller than *cwin*, it is more likely that packet losses are due to congestion. This is because the connection is using a *cwin* value much higher than its share of pipe size, thus congestion is likely. Figure 12 shows the relationship between the two quantities in a simulation experiment where congestion is the only cause of packet loss (no link errors.)

The simulation includes two connections competing for a 5Mbps bottleneck link with no random error. The RTT is set to 70ms, and buffer size to 32 packets (equal to pipe size). Figure 12 shows that the  $RE * min\ RTT$  is much lower than the *cwin* at the point of loss due to congestion.



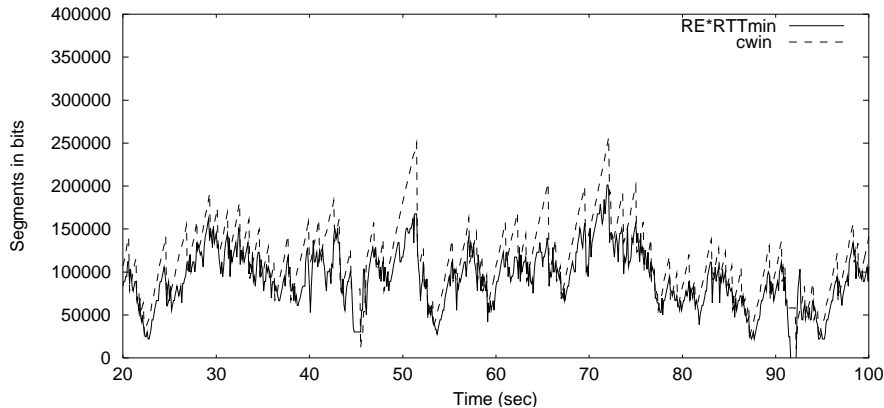
**Figure 12. Congestion window and RE\*RTTmin comparison (Congestion case)**

Next we consider an environment where a single connection is running over the same bottleneck link with 2Mbps capacity, 100ms RTT, and 1% packet loss rate due to random link error. Figure 13 shows that there is no significant difference between RE \* min RTT and *cwin* as predicted by theory. happens.



**Figure 13. Congestion window and RE\*RTTmin comparison (Link-error case)**

In Figure 14, we show the same data for a configuration with losses caused by both random error (1%) and congestion. We simulate 4 connections sharing the bottleneck in this case. At 52 seconds, *cwin* is much larger than RE\*RTTmin, indicating a loss due to congestion. At instants where *cwin* is close to RE\*RTTmin, the loss is due to random error. But at instants where *cwin* is larger than RE\*RTTmin, but not significantly so, it is not possible to identify with accuracy the cause of packet loss. However we have observed in our simulations that BE and RE tend to be not very far off from each other in such cases.



**Figure 14. Cwin and RE\*RTTmin comparison (both congestion and Link-error)**

## 5.2. CRB Algorithm

In CRB, whenever a packet loss is indicated, the sender determines the predominant cause of loss as follows: when the ratio RE \* RTTmin to *cwin* exceeds a threshold value  $\theta$ , the use of RE is prescribed. Below  $\theta$ , BE is prescribed. In our experiments, a threshold of  $\theta=1.4$  was found to give the best results. A packet loss is detected either by a reception of 3

DUPACKs or a coarse timeout. The TCPW CRB algorithm is similar to TCPW BE except for setting the *ssthresh* and *cwin* according to network congestion condition as we described above.

Below we present the TCPW CRB algorithm to set *ssthresh* and *cwin* after a packet loss indicated by three duplicate ACKs or by timeout expiration respectively. In the pseudo-code, *seg\_size* identifies the length of a TCP segment in bits. The value *RTTmin* is set as the smallest RTT estimated by TCP, using its own RTT estimation algorithm. Note that the basic Reno behavior is still captured, while setting *ssthresh* to the value of BE or RE, as appropriate, provides a more rational recovery.

In case the loss indication is 3 DUPACKs, *ssthresh* and *cwin* are set by TCP CRB as follows:

```

if (3 DUPACKs are received)
  if (cwin/((RE * RTTmin) / seg_size) > θ) /*network is congested*/
    ssthresh = (RE * RTTmin) / seg_size;
  else /* no congestion */
    ssthresh = (BE * RTTmin) / seg_size;
  endif
  if (cwin > ssthresh) /* congestion avoid. */
    cwin = ssthresh;
  endif
endif

```

In case of a packet loss indicated by a timeout expiration, *ssthresh* and *cwin* are set as follows:

```

if (coarse timeout expires)
  cwin = 1;
  ssthresh = (BE * RTTmin) / seg_size;
  if (ssthresh < 2)
    ssthresh = 2;
  endif;
endif

```

The rationale of the algorithm above is that after a timeout, *cwin* and the *ssthresh* are set equal to 1 and BE, respectively, to provide a more speedy recovery based on estimated share. Note that RE is not used in this case. During a timeout, the sender either does not send anything (lack of acknowledgements) or just send a packet per RTT (due to multiple losses). Under such conditions, RE would underestimate the fair share and it is more appropriate to set the *ssthresh* to the BE value.

## 6. Performance Evaluation

We compare the performance of RE, BE, CRB, and NewReno. All results are obtained using the ns2 simulator [ns2][TCPW]. We first evaluate the throughput of the different protocols with lossy links (wireless) in Section 4.1. Then in Section 4.2, we study the fairness, friendliness and utilization of the bottleneck link. We introduce the Efficiency/Friendliness Tradeoff graph in Section 4.2.

The summary of the results is that the combined method provides a means of adapting to loss causes, and to effectively manage the Friendliness/Efficiency tradeoff.

### 6.1. Throughput with Lossy Links

We begin with the evaluation of the efficiency provided by our schemes. In this section, a number of scenarios are studied to compare the performance of BE, RE, and CRB in the wired/wireless configuration shown below.

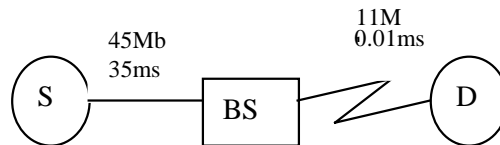
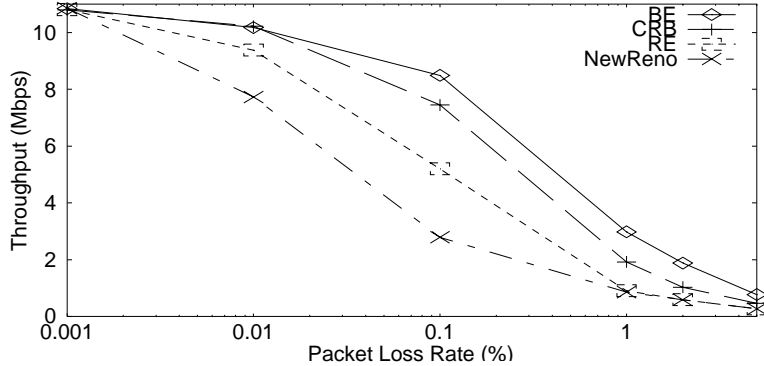


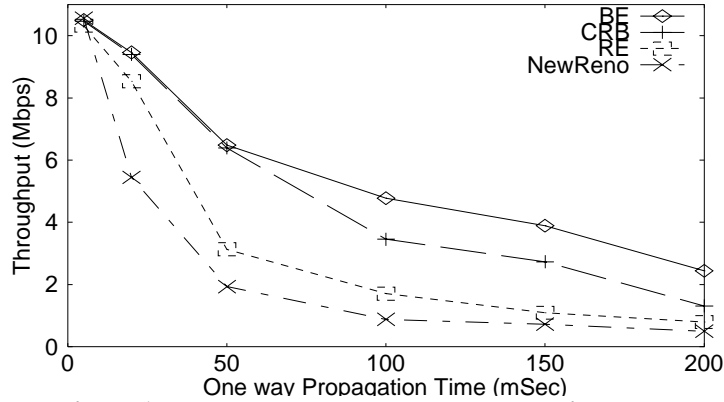
Figure 15. Simulation topology

Figure 15 shows a mixed network, where the wired portion has capacity of 45 Mbps and one-way propagation time of 35ms (roughly the delay from West to East coast.) The wireless portion of the network is a very short 11-Mbps wireless link with a propagation time of 0.01ms (e.g., WaveLAN.) The wireless link is assumed to connect a base station to a destination mobile terminal.



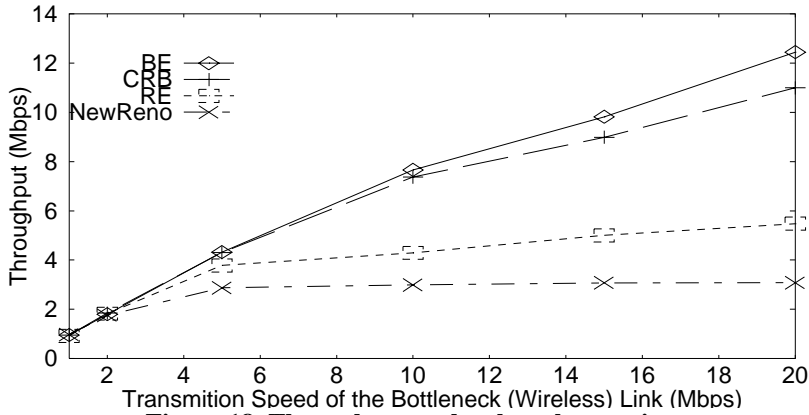
**Figure 16. Throughput vs. packet loss rate of the wireless link**

The throughput of BE, CRB, RE and NewReno are compared under packet loss rate varying from 0 to 5%. This range is similar to that used in [HPSK97]. The results in figure 16 show that BE, RE and CRB throughput is higher than that of NewReno. The largest improvement is obtained around 0.1% to 1% loss rate, where BE gains up to 320% and CRB 267% over TCP NewReno. RE provides improvement also, but only about 190% over NewReno.



**Figure 17. Throughput vs. one-way propagation delay**

To assess the relation of the throughput gains to the E2E propagation time, we ran simulations with the wired portion propagation time varying from 0 to 200ms and the wireless link loss rate set to 0.1%. The results in Figure 17 show a significant gain for BE and CRB of up to 542% and 400% respectively, at a two way propagation time of 100ms. Again RE shows moderate improvement over TCP NewReno. When the propagation time is small (say, less than 5ms), all protocols are equally effective. This is because a small window is adequate and window optimization is not an issue. The “network aware” schemes reach maximum improvement over NewReno as the propagation time increases to about 100ms. After that, the gain starts to decrease. A potential reason for this may be that the feedback information used to estimate the available bandwidth arrives too late to be of significant help. This is a scaling problem that is worth future investigation.



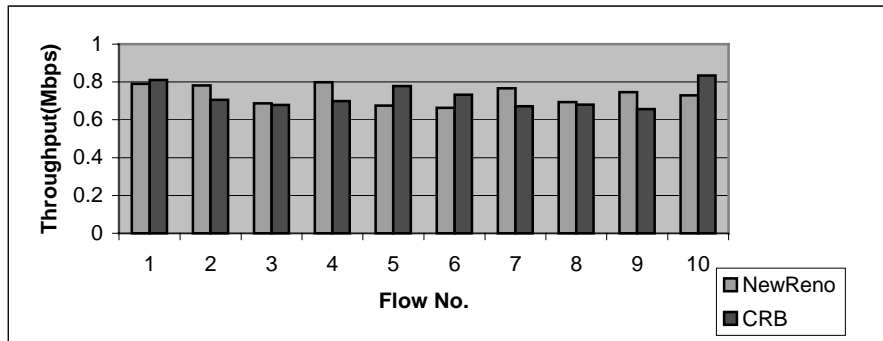
**Figure 18. Throughput vs. bottleneck capacity**

Simulation results in Figure 18 show that BE and CRB gains increase significantly as the bottleneck link transmission speed increases. Thus, BE and CRB are more effective than TCP NewReno in utilizing the Gbps bandwidth provided by new-generation, high-speed networks. Figure 18 shows that the improvements obtained via BE and CRB increase to approximately 418% and 370% respectively when the wireless link speed reaches 20 Mbps. The error model is still Bernoulli with parameter 0.1%, and the E2E propagation time here is 70ms.

### 6.2. Fairness, Friendliness and Utilization

In this subsection, we focus on fairness, friendliness and its tradeoff with utilization. We distinguish between Fairness and Friendliness as follows. Fairness relates to the relative performance of a set of connections of the same TCP variant. Friendliness relates to how sets of connections running different TCP flavors affect the performance of each other. The simulation topology consists of a single bottleneck link with a capacity of 10 Mbps, and one-way propagation delay of 35ms. The buffer size at the bottleneck router is 64 packets (equal to the pipe size), except where otherwise stated. The link error rate varies depending on the scenario.

A set of simulations with 10 simultaneous flows was run to investigate the fairness property of the CRB scheme (the throughput result is shown in Figure 19.) The Jain's fairness index of CRB reached 0.9944, and that of NewReno is 0.9952. Therefore, fairness of CRB is comparable to that of NewReno.



**Figure 19. Fairness Comparison (10 flow of CRB or NewReno respectively)**

We studied friendliness properties of TCPW BE and TCPW CRB towards NewReno. We ran simulations with two TCP connections of various protocols sharing the same bottleneck. Recall that BE may overestimate the fair share under congestion conditions. The throughput results are shown in Figure 20 and 21. As we can see from the figures, among the three tests, namely, BE/NewReno, CRB/NewReno and NewReno/NewReno, the BE scheme achieves the highest throughput and the combination of BE/NewReno achieves the highest total utilization of the bottleneck capacity.

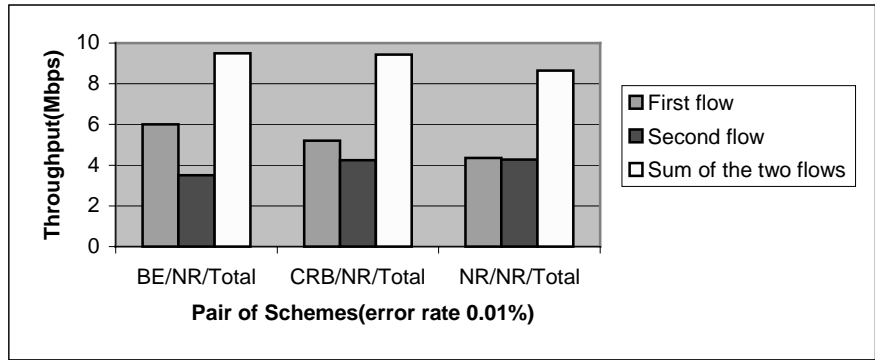


Figure 20. Friendliness and utilization comparison (Link-error: 0.1%)

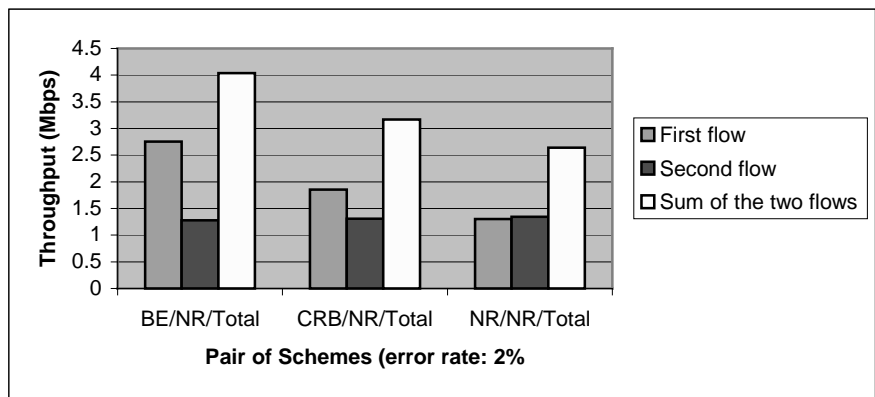


Figure 21. Friendliness and utilization comparison (Link-error: 2%)

However, BE appears to be unfriendly to NewReno. It deteriorates Reno’s performance as shown in Figure 20. Meanwhile, CRB shows almost no effect on NewReno, being quite comparable to the NewReno/NewReno combination. This indicates that CRB only takes advantage of the unused bandwidth and does not “steal” bandwidth from NewReno. Under larger error rate, as shown in figure 21, both BE and CRB hardly damage NewReno. The reason is the bottleneck is NOT congested, and TCPW (of either type) does not interfere with NewReno. Both BE and CRB are able to pick up part of the unused bandwidth without hurting a coexisting NewReno.

We also investigated the impact of buffer space on friendliness. We use a configuration with a pipe size of 64 packets and the paths are random error free. We can see from Figure 22, that when the buffer size is much smaller than the pipe size, BE is unfriendly to NewReno; and the throughput ratio reaches about 2:1. In the small buffer case, buffer overflows happen frequently, both NewReno and BE cannot grow their windows to fill the pipe before another buffer overflow. NewReno, upon packet loss indicated by three duplicate ACKs, reduces its window by one half, which would be smaller than half the pipe size, while BE keeps its window higher corresponding to the available bandwidth estimation. RE and CRB, on the other hand, show much more friendliness to NewReno in the small buffer case.

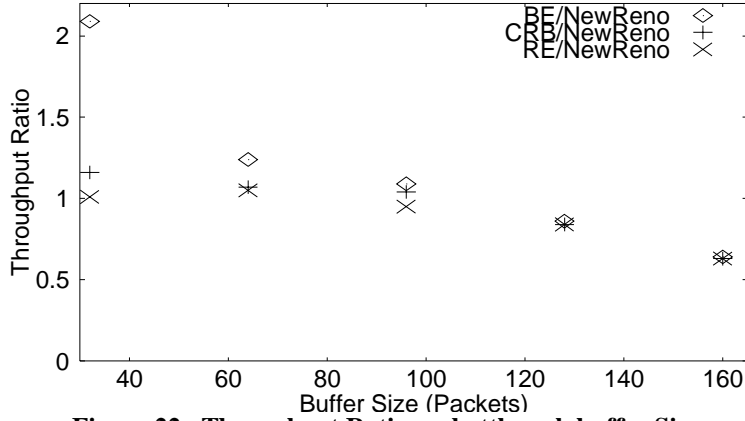


Figure 22. Throughput Ratio vs. bottleneck buffer Size

Next, we examine (again in Fig 22) the case where the buffer size is much larger than the pipe size = 64. Here, NewReno gets more than its fair share, thus it is unfriendly to TCPW. This was expected since in the large buffer case (say buffer > 120), New Reno reduces the window by half; but the resulting window is still too large to clear the buffer, and relieve congestion. On the other hand, TCPW schemes reduce the *cwin* and *ssthresh* by much more than one half, helping to clear the buffer backlog faster and provide better performance.

From the above experiments we conclude that BE provides high efficiency and RE provides more friendliness. CRB incorporates both schemes in order to achieve both high utilization and friendliness to NewReno. To better understand and visualize how BE and CRB interact with TCP NewReno, we introduce in the next section the Efficiency/Friendliness Tradeoff Graph.

### 6.3 Efficiency/Friendliness Tradeoff Graph

To better visualize the efficiency/friendliness tradeoff behavior of our sampling schemes, we have developed the Efficiency/Friendliness Tradeoff Graph. Basically we intend to study how the total link utilization and TCP NewReno's throughput are impacted by the introduction of TCPW connections sharing the same bottleneck with NewReno connections. The following experiments are carried out to produce a point on the graph:

1. A simulation with N TCP NewReno flows transmitting is run as the base case. The throughput of each flow, and the total utilization, are measured.
2. Another simulation is then run with half of the flows replaced with either TCPW BE or TCPW CRB. The new throughput and utilization are measured.

Let  $t_{R1}$  be the average throughput of the flows in the first simulation, and  $U_1$  be the total link utilization. Similarly, let  $t_{R2}$  be the average throughput of the NewReno flows in the second simulation, and  $U_2$  be the total link utilization (TCPW + NewReno.)

We define

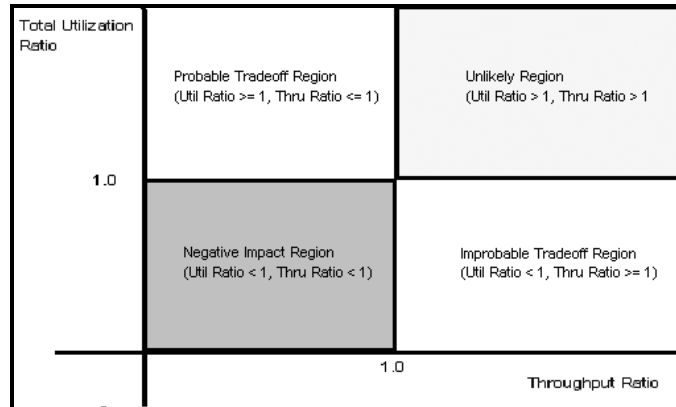
$$\text{"Utilization Gain Ratio"} G = U_2/U_1,$$

$$\text{"NewReno Throughput Ratio"} L = (t_{R2}/t_{R1}^*)$$

In the equation above  $t_{R1}^*$ , which is the average throughput of half of the NewReno flows in simulation 1, is used. This is used because even if the same TCP variant is used, each flow can achieve different throughput and thus half of the group will always "hurt" the other half, and by using half of the flows as the base case, we normalize the ratio to 1 if the "new" protocol is in fact the same as the old one. Also notice that  $t_{R1}^*$  depends on how we group the flows into two halves and which half do we pick. Ideally we should calculate all the possible ways to pick a half, calculate the Throughput Gain, and then take the average. In our experiments, however, we take the average of the best and worst case scenarios. We find the best and worst case by sorting the list of throughputs in ascending order, divide them into two halves, top and bottom. We then have the two extreme cases if we use either one of the averages as  $t_{R1}^*$ . If we name the average throughput of the two extreme cases as  $t_{R1-worst}$  and  $t_{R1-best}$ , we have:

$$\text{NewReno Protocol Throughput Ratio } L = (t_{R2}/t_{R1\text{-worst}} + t_{R2}/t_{R1\text{-best}}) / 2$$

For each network scenario of interest, we determine G and L, and place the point (G, L) on the Utilization Gain vs. Legacy Protocol Throughput Gain graph. A point can fall in one of four regions of the graph as shown in Figure 23. In the “Negative Impact Region,” both G and L are less than 1, which is undesirable. In the “Unlikely Region,” both G and L are greater than one, which is desirable because the new protocol helps the efficiency of everyone, but as the name suggests, it is unlikely. In most cases, however, the points will fall in the two tradeoff areas, in which an increase in G (or L) is compensated for by a decrease in L (or G.) With TCPW, we expect to see the points in the region where  $G > 1$  and  $L < 1$  (“Probable Tradeoff Region.”)

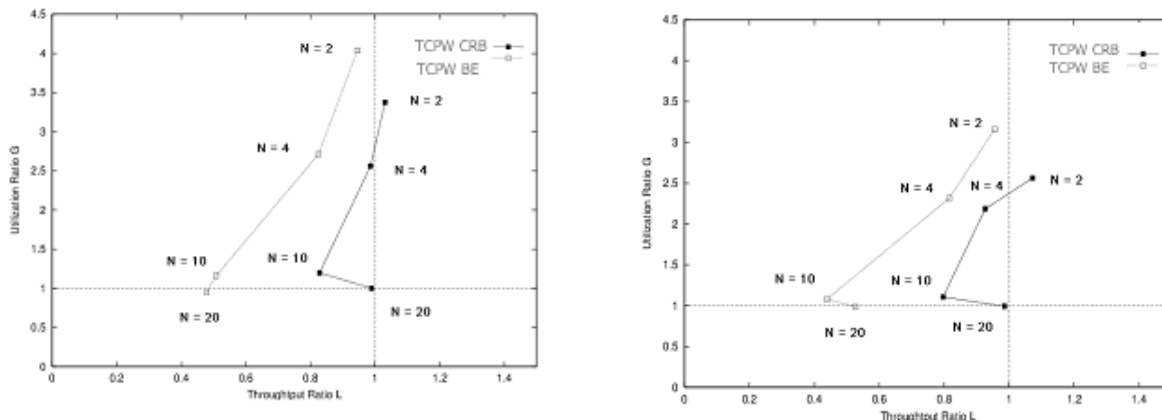


**Figure 23. Efficiency/Friendliness Tradeoff Graph**

The “target” points are anywhere on the line  $L = 1, G > 1$ , with G as large as possible. With TCPW BE, we expect to have a higher G and lower L, i.e. in the middle of the Probable Tradeoff Region. TCPW CRB should reside in an area much closer to  $L=1$ . This is indeed confirmed in our results presented below.

Two network scenarios in which TCPW has the most advantage (and thus drawing the most concern on friendliness) are simulated. The first one is a large leaky pipe (bottleneck bandwidth = 45Mbps, round trip propagation time = 74ms, random loss rate = 0.1%.) The second one is wired/wireless configuration, (wireless bottleneck bandwidth = 11Mbps, round trip propagation time = 74ms, random loss rate = 2%.) Both experiments show that TCPW CRB is friendly, yet provides relatively high utilization gain.

The Efficiency/Friendliness Tradeoff Graphs are shown in Figure 24 (a) and (b). For each experiments we show the points for different number of competing flows (N.) When N is small (e.g. 2, 4), most of the link is not utilized and the introduction of TCPW increases efficiency. With TCPW BE, however, NewReno experiences performance deterioration. On the other hand, TCPW CRB achieves less utilization gain without much effect on TCP NewReno. When N is large (e.g. 20) the NewReno flows collectively almost fully utilize the link by themselves. TCPW BE and CRB have no room to improve total efficiency. Note however that TCPW CRB in this case shares the bottleneck fairly with NewReno.

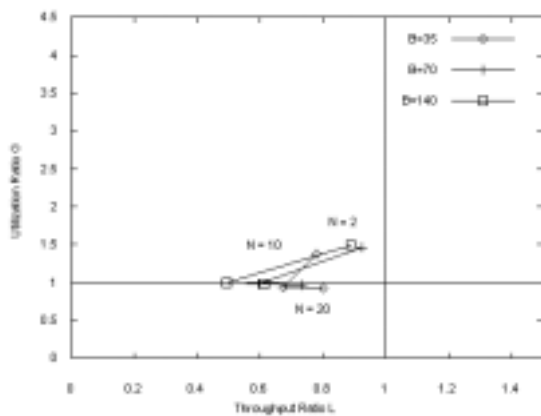


(a) (b)  
**Figure 24. Efficiency/Friendliness Tradeoff Graph (a) 45Mbps, 74ms, 0.1% error (b) 11 Mbps, 74ms, 2% error**

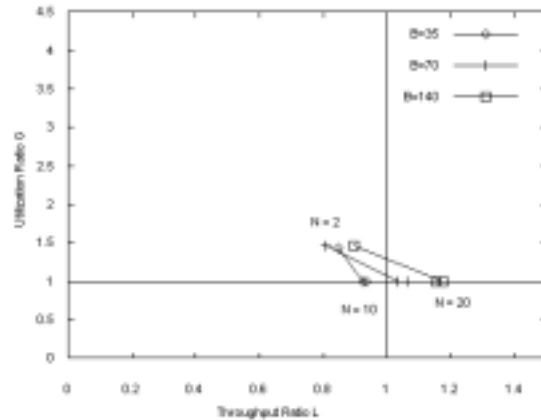
From the graphs the point  $N = 10$  for TCPW CRB is the farthest away from  $L=1$ . It appears that this is due to the fact that neither error nor buffer overflow is a dominant cause of loss. A possible approach to handle this issue is tuning of the threshold  $\theta$  that determines the selection between BE and RE (in the experiments  $\theta=1.4$  was used.) Nonetheless, from the graph above we can see that CRB provides significant improvement in friendliness and achieves its goal: to utilize the link more with little effect on others.

Since the buffer size of the intermediate routers is not configured in relation to any “pipe size” of the flows going through a router, it is useful to show that the improvement presented above will still be valid with different values of buffer size. In the following two studies we investigate CRB robustness to buffer size variation, on paths with and without random errors. We first present results for different error rates and different and show that the impact of buffer size variation is small when the paths are prone to random errors. In the case of an error-free path, we show that buffer size has more impact, and show that TCPW CRB is more robust to buffer size variation.

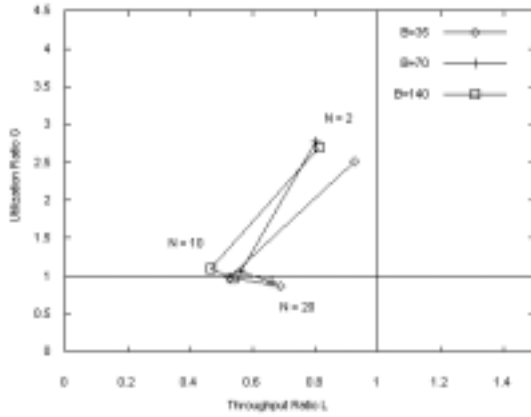
In the first study we extend the experiment presented in Figure 24(b) with different values of buffer size and error rates. The network configuration is the same as Figure 24(b) but the error rate can be either 0.1% or 1% while the buffer size can be 35(half the pipe size), 70(pipe size) or 140 packets (twice the pipe size). The results are shown in Figure 25. On the graphs, different lines indicate different buffer size, and different points indicate different number of flows. From the graphs we can first make two observations: (a) The points of TCPW CRB are closer to the target points and, (b) With a large  $N$  (number of total connections), meaning a high level of congestion, TCPW CRB settles at the point (1,1) while TCPW BE settles in points where  $G = 1$  and  $L \ll 1$  (unfriendly.) This further confirms the improvement achieved with TCPW CRB. Moreover, notice that the lines are not far apart from each other, meaning that the variation of buffer size does not affect the performance significantly in this case.



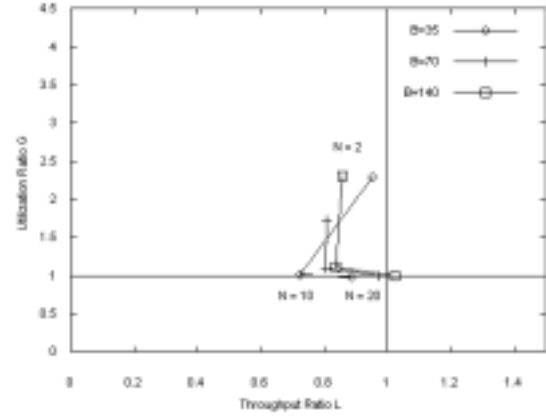
(a) TCP BE with 0.1% Error



(b) TCP CRB with 0.1% Error



(c) TCP BE with 1% Error

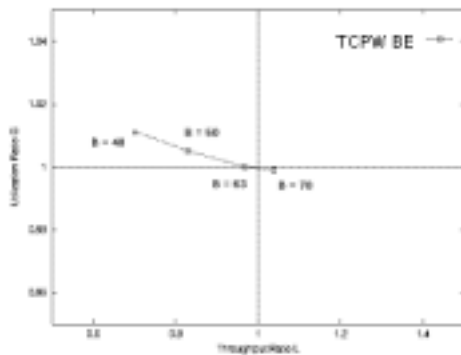


(d) TCP CRB with 1% Error

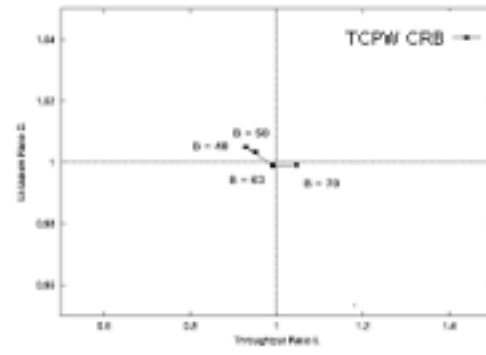
Figure 25. Buffer Size Impact on Efficiency/Friendliness

When there is no link error, TCP NewReno itself can already utilize most of the link. TCPW introduction in such a case will not improve total link utilization, and thus it is more important that TCPW be friendly in this case. Hence, the target point here is (1,1). We have seen previously (from Figure 22) that if TCPW and TCP NewReno compete over a bottleneck link with a small buffer size, TCPW tends to achieve a throughput higher than the fair share. In the case of large buffer size, TCP NewReno takes more bandwidth than the fair share. Here we present the Efficiency/Friendliness Tradeoff Graphs with varying buffer size and no error. The results are shown in Figure 26.

Simulations with 2 connections sharing a bottleneck link with 10 Mbps bandwidth and 70 ms roundtrip propagation time, giving a pipe capacity of 63 packets, are run. The buffer size was set to 4 different values. When the buffer size is equal to the pipe size, both schemes are close to the target point (1,1). However, when the buffer size varies, we can see that the dispersion of points is worse in TCPW BE than in TCPW CRB. Observe that when the buffer size is decreased from 63 to 40, TCP NewReno suffers a lot from TCPW BE with a throughput drop of about 30%. When coexisting with TCPW CRB in the same condition, TCP NewReno is able to maintain more than 90% of its original throughput. This, combined with the results from Figure 24 and 25, show that TCPW CRB is able to utilize unused bandwidth while being friendly to TCP NewReno for a range of link speeds, error rates and buffer sizes.



(a) TCPW BE



(b) TCPW CRB

Figure 26. Buffer Size Impact on Efficiency/Friendliness in Error Free Path

## 7. Conclusion and future work

This work has been motivated by the need to make TCPW friendly to legacy protocols such as NewReno. We have shown that friendliness as well as throughput efficiency is impacted by the choice of bandwidth estimation in TCPW. We have introduced two estimators, BE and RE and have carried out an extensive analysis of friendliness/throughput

tradeoffs in different network, channel and traffic conditions. Based on these tradeoffs, we have proposed a combined RE/BE estimation method—CRB—that allows the scheme to adapt to different types of losses (congestion or link errors), as well to manage the Efficiency/Friendliness tradeoff when TCPW and NewReno are simultaneously deployed in a network. A key feature is the method to detect the predominant cause for packet loss, and accordingly select an appropriate estimator. TCPW CRB was shown to provide good compromise between efficiency (i.e. increase in total link utilization) and friendliness (i.e. impact on legacy protocol).

Another important behavior uncovered by our study is the impact of buffer space on relative utilizations. When buffer space is large, NewReno appears to get more than its fair share. With small buffers, it is TCPW that gets more than its fair share. Robustness to buffer space is an important and desirable attribute of TCP protocols. AQM schemes seem to help reduce the impact of buffer size variation in TCPW; however, we are yet to evaluate this in detail.

The experimental results we presented in this paper provide preliminary insight into friendliness issues. Our work is continuing in order to study in more detail the impacts of buffer size, buffer management techniques (e.g. AQM), error rates, and link speeds on friendliness. Further, controlling the degree of friendliness through the parameter  $\theta$  may be worth investigating. This also points to the possibility of devising continuously adaptive sampling methods instead of the binary selection among RE and BE presented in this paper.

## References

- [AP99] M. Allman and Vern Paxson, “On Estimating End-to-End Network Path Properties”, *ACM/Sigcomm 1999*.
- [BB00] C. Boutremans, J.-Y. Le Boudec “A Note on the Fairness of TCP Vegas,” *Proceedings of International Zurich Seminar on Broadband Communications, Zurich, Switzerland, February 2000, pp. 163-170*
- [BK98] H. Balakrishnan and R. H. Katz, “Explicit Loss Notification and Wireless Web Performance,” In *Proceedings of IEEE GLOBECOM’98 Internet Mini-Conference, Sydney, Australia, November 1998*.
- [Bona99] T. Bonald, “Comparison of TCP Reno and TCP Vegas: Efficiency and Fairness,” In *Proceedings of PERFORMANCE’99, Istanbul, Turkey, October 1999*.
- [BP95] L.S. Brakmo and L.L. Perterson. “TCP Vegas: End-to-End Congestion Avoidance on a Global Internet.” *IEEE Journal on Selected Areas in Communication, Vol. 13, Nov. 8, October 1995*.
- [BPSK97] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz, “A Comparison of Mechanisms for Improving TCP Performance over Wireless Links,” *IEEE/ACM Transactions on Networking, December 1997*.
- [BSAK95] H. Balakrishnan, S. Seshan, E. Amir, and R. H. Katz, “Improving TCP/IP Performance Over Wireless Networks,” *MOBICOM’95, Berkeley, CA, USA, November 1995*.
- [CC96] R.L. Carter ND m.e. Crovella, “Measuring Bottleneck Link Speed in Packet Switch Networks”, *BU-CS-96-006, Technical Report, Boston University, 1996*.
- [CGLMS00] C. Casetti, M. Gerla, S. Lee, S. Mascolo, and M. Sanadidi, “TCP with Faster Recovery,” *MILCOM 2000, Los Angeles, CA, October 2000*.
- [CGMSW01] C. Casetti, M. Gerla, S. Mascolo, M. Y. Sanadidi, and R. Wang, “TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links,” In *Proceedings of Mobicom 2001, Rome, Italy, Jul. 2001*.
- [CK74] V. C. Cerf and R. E. Kahn, “A Protocol for packet Network Interconnections,” *IEEE Transactions on Communications, vol. COM-22, no. 5, pp. 637-648, May 1974*.
- [Clar88] D. Clark, “The design philosophy of the DARPA Internet protocols,” In *Proceedings of Sigcomm’88 in ACM Computer Communication Review, vol. 18, no. 4, pp. 106 - 114, 1988*.
- [DJ02] C. Dovrolis, M. Jain, (2002) “End-to-End Available Bandwidth: Measurement methodology, Dynamics, and Relation with TCP Throughput”, In *Proceedings of ACM SIGCOMM, August 2002*.
- [FF96] K. Fall and S. Floyd, “Simulation-based Comparisons of Tahoe, Reno, and SACK TCP,” *Computer Communication Review, V. 26 N. 3, July 1996, pp. 5-21*.
- [FJ93] S. Floyd and V. Jacobson, “Random Early Detection gateways for congestion Avoidance,” *IEEE/ACM transactions on Networking, August 1993*
- [Floy94] S. Floyd, “TCP and Explicit Congestion Notification,” *ACM Computer Communication Review, V. 24 N. 5, pp. 10-23, Oct. 1994*.
- [GLMW99] M. Gerla, R. Lo Cigno, S. Mascolo, and W. Weng, “Generalized Window Advertising for TCP Congestion Control,” *CSD-TR 990012, UCLA, CA, USA, February 1999*.
- [GMPG00] T. Goff, J. Moronski, D. S. Phatak, and V. Gupta, “Freeze-TCP: a True End-to-end TCP Enhancement Mechanism for Mobile Environments,” In *Proceedings of IEEE INFOCOM’2000, Tel Aviv, Israel, March 2000*.
- [HBG00] U. Hengartner, J. Bolliger, and T. Gross, “TCP Vegas Revisited,” In *Proceedings of IEEE INFOCOM’2000, Tel Aviv, Israel, March 2000*.

- [HMM98] G. Hasegawa, M. Murata and H. Miyahara, "Fairness and Stability of Congestion Control Mechanism of TCP," in *Proceedings of 11<sup>th</sup> ITC Special Seminar, October, 1998*
- [Hoe96] J. C. Hoe, "Improving the Start-up of A Congestion Control Scheme for TCP", *Proc. ACM SIGCOMM '96*, pp. 270-280.
- [Jaco88] V. Jacobson, "Congestion Avoidance and Control," *ACM Computer Communications Review*, 18(4) : 314 - 329, August 1988.
- [Jaco90] V. Jacobson, "Berkeley TCP evolution from 4.3-Tahoe to 4.3 Reno," *Proceedings of the 18<sup>th</sup> Internet Engineering Task Force, University of British Columbia, Vancouver, BC, Sept. 1990*.
- [Kesh91] S. Keshav "A Control-Theoretic Approach to Flow Control," *Proceeding of ACM SIGCOMM 1991*
- [KR00] J. Kurose and K. Ross, "Computer Networking: A Top-Down Approach Featuring the Internet", *Addison Wesley, 2000*.
- [KVR98] L. Kalampoukas, A. Varma, and K. K. Ramakrishnan, "Explicit Window Adaptation: A Method to Enhance TCP Performance," *In Proceedings of IEEE INFOCOM'98, San Francisco, Ca, USA, March/April 1998*.
- [LB00] K. Lai and M. Baker, "Measuring Link Bandwidths Using a Deterministic Model of Packet Delay", *Sigcomm 2000*.
- [LH95] 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, pp. 10 - 25, Feb. 1995.
- [LM97] T. V. Lakshman, U. Madhow, "The Performance of Networks with High Bandwidth-delay Products and Random Loss," *IEEE/ACM Transactions on Networking*, June 1997.
- [Low00] S. H. Low, "A Duality Model of TCP and Queue Management Algorithms," *Extended version of paper that appears in Proceedings of ITC Specialist Seminar on IP Traffic Measurement, Modeling and Management, September 18-20, 2000*.
- [ns2] ns-2 network simulator (ver.2.) LBL, *URL: <http://www.mash.cs.berkeley.edu/ns>*.
- [PF01] J. Padhye, S. Floyd, "On Inferring TCP Behavior", *ACM/SIGCOMM'01, Aug. 2001*
- [RFC2488] M. Allman, D. Glover, and L. Sanchez, "Enhancing TCP over Satellite Channels using Standard Mechanisms," *RFC 3488, Jan. 1999*.
- [RFC2582] S. Floyd, and T. Henderson, "The NewReno Modification to TCP's Fast Recovery Algorithm," *RFC 2582, Experimental, April 1999*.
- [RFC2861] M. Handley, J. Padhye and S. Floyd, "TCP Congestion Window Validation," *RFC 2861, June 2000*.
- [RFC2883] S. Floyd, J. Mahdavi, M. Mathis and M. Podolsky, "An Extension to the Selective Acknowledgement (SACK) Option for TCP," *RFC 2883, July 2000*.
- [TCPW] TCP Westwood modules for ns-2. *URL: <http://www.cs.ucla.edu/~mvalla/tcpw>*
- [Witt97] K. J., B. Wittenmark, "Computer controlled systems," *Prentice Hall, Englewood Cliffs, N. J., 1997*.
- [ZPGS01] A. Zanella, G. Procissi, M. Gerla, M.Y. Sanadidi, "TCP Westwood: analytical model and performance evaluation", *Technical Report, URL: <http://www.cs.ucla.edu/csd/pubs/pubs.html>*
- [ZSC91] L. Zhang, S. Shenker, and D. D. Clark, "Observations on the Dynamics of A Congestion Control Algorithm: the Effects of Two-way traffic", *Proc. ACM SIGCOMM '91, pp.133-147*.