TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links

Saverio Mascolo
Politecnico Di Bari

Claudio Casetti
Politecnico Di Torino

Mario Gerla, M. Y. Sanadidi, and Ren Wang
UCLA Computer Science Department

Abstract

TCP Westwood (TCPW) is a sender-side modification of the TCP congestion window algorithm that improves upon the performance of TCP Reno in wired as well as wireless networks. The improvement is most significant in wireless networks with lossy links, since TCP Westwood relies on end-to-end bandwidth estimation to discriminate the cause of packet loss (congestion or wireless channel effect) which is a major problem in TCP Reno. An important distinguishing feature of TCP Westwood with respect to previous wireless TCP "extensions" is that it does not require inspection and/or interception of TCP packets at intermediate (proxy) nodes. Rather, it fully complies with the end-to-end TCP design principle. The key innovative idea is to continuously measure at the TCP source the rate of the connection by monitoring the rate of returning ACKs. The estimate is then used to compute congestion window and slow start threshold after a congestion episode, that is, after three duplicate acknowledgments or after a timeout. The rationale of this strategy is simple: in contrast with TCP Reno, which "blindly" halves the congestion window after three duplicate ACKs, TCP Westwood attempts to select a slow start threshold and a congestion window which are consistent with the effective bandwidth used at the time congestion is experienced. We call this mechanism *faster recovery*. The proposed mechanism is particularly effective over wireless links where sporadic losses due to radio channel problems are often misinterpreted as a symptom of congestion by current TCP schemes and thus lead to an unnecessary window reduction. Experimental studies reveal improvements in throughput performance, as well as in fairness. In addition, friendliness with TCP Reno was observed in a set of experiments showing that TCP Reno connections are not starved by TCPW connections. Most importantly, TCPW is extremely effective in mixed wired and wireless networks where throughput improvements of up to 550% are observed. Finally, TCPW performs almost as well as localized link layer approaches such as the popular Snoop scheme, without incurring the O/H of a specialized link layer protocol.

This research was supported by NSF under Grant ANI-9983138 on High Speed Networks Performance Measurements and Analysis.

ACM SIGMOBILE 7/01 Rome, Italy

1. Introduction

Effective error and congestion control for heterogeneous (wired and wireless) networks has been an active area of research recently. End-to-end, Link Layer, and Split Connection approaches have been suggested and their relative merits extensively assessed in recent studies [17]. One conclusion drawn from these studies is that while end-to-end schemes are not as effective as local recovery techniques in handling wireless losses, they are promising since significant gains can be achieved without extensive support at the network layer in routers and base stations [3]. With these motivations, we propose in this paper a TCP-based end-to-end (E2E) approach to error recovery and congestion control in wired/wireless networks and study its performance.

The well-known challenge in providing TCP congestion control in mixed environments is that current TCP implementations rely on packet loss as an indicator of network congestion. In the wired portion of the network a congested router is indeed the likely reason of packet loss; on a wireless link, on the other hand, a noisy, fading radio channel is the more likely cause of loss. This creates problems in TCP Reno since it does not possess the capability to distinguish and isolate congestion loss from wireless loss. As a consequence, TCP Reno reacts to wireless loss with a drastic reduction of the congestion window, hence of the sender transmission rate, when the best strategy would in fact be to increase the retransmission rate.

Approaches to address this problem have been discussed and compared in the excellent work by Balakrishnan et al. [3]-[5]. Three alternative approaches: E2E, Split Connection, and Localized Link Layer methods were carefully contrasted. The best performing approach was shown to be a localized link layer solution applied directly to the wireless links. A clever "snooping" protocol is introduced. The protocol appropriately called "Snoop" monitors the packets flowing over the wireless link as well as their related acknowledgments. The protocol entities cache copies of TCP data packets and monitor the ACKs in the reverse direction. If a packet loss is detected (i.e., through duplicate acknowledgments, DUPACKs), the cached copy is used for local retransmission, and any packet carrying feedback information back to the TCP sender is extracted so as to avoid "premature" retransmission at the TCP sender. The protocol is effective in reducing E2E retransmissions, and, more importantly, in preventing the associated reduction in congestion window size.

Snoop, however, has its own limitations. First, it requires a snoop proxy in the base station. Also, if the TCP sender is the mobile, the TCP code must be modified to respond to Explicit Loss Notification (ELN) packets from the base station.
In view of the limitations introduced by link layer solutions, it is of interest thus to explore E2E recovery solutions that are independent of the link layer, and thus more versatile.

In this paper we propose to handle wireless losses using a modified version of TCP Reno. This new version, which we named TCP Westwood (or TCPW for short), enhances the window control and backoff process. Namely, a TCPW sender monitors the acknowledgment reception rate and from it estimates the data packet rate currently achieved by the connection. Whenever a sender perceives a packet loss (i.e. a timeout occurs or 3 duplicate ACKs are received), the sender uses the bandwidth estimate to properly set the congestion window (cwin) and the slow start threshold (ssthresh). By backing off to cwin and ssthresh values that are based on the estimated available bandwidth (rather than simply halving the current values as Reno does), TCP Westwood avoids overly conservative reductions of cwin and ssthresh; and thus it ensures a *faster recovery*. Experimental studies reveal the benefits of the intelligent backoff strategy in TCPW: better throughput, goodput, and delay performance, as well as fairness even when competing connections differ in their end-to-end propagation times. In addition, our studies of TCPW friendliness when coexisting with TCP Reno is reassuring since we have observed that TCP Reno connections are not starved in the presence of TCPW connections. Most importantly, TCPW is very effective in handling wireless loss. This is because TCPW uses the current estimated rate as reference for resetting the congestion window. The current rate is only marginally impacted by loss (as long as loss is a relatively small fraction of data rate). The simulation results presented in Section 4 confirm this claim. For example, a throughput improvement of up to 550% over TCP Reno has been observed.

Other TCP variants that use bandwidth estimation to set the congestion window have been proposed before. To our knowledge, however, such schemes require the intervention of the network layer. For example, the BA-TCP (Bandwidth Aware-TCP) scheme [10] relies on intermediate routers to take measurements of available bandwidth and compute the “fair share” for the TCP connections. The fair share value is piggybacked in the TCP header and conveyed to the TCP source. The latter uses it to appropriately set its cwin and ssthresh parameters. BA-TCP and TCPW are similar in their reliance on bandwidth information to set congestion control parameters. However, while BA-TCP requires new network layer functions to measure available bandwidth and compute fair share, TCPW relies only on information readily available in the current TCP header. TCPW does not require any support from lower layers, and thus strictly adheres to layer separation and modularity principles.

The TCPW bandwidth estimation and congestion control algorithms are discussed in Section 2 and 3 below. TCPW performance behavior in wired and in mixed networks is studied in Sections 4 and 5. Section 6 concludes the paper.

2. End-to-End Bandwidth Measurement

2.1 The ACK-based measurement procedure

A fundamental design philosophy of the Internet TCP congestion control algorithm is that it must be performed end-to-end. The network is considered as a "black box". A TCP source cannot receive any explicit congestion feedback from the network. Therefore the source, to determine the rate at which it can transmit, must probe the path by progressively increasing the input load (through the slow start and congestion avoidance phases) until implicit feedback, such as timeouts or duplicate acknowledgments, signals that the network capacity has been reached.

The importance of the end-to-end principle [8] can never be overemphasized. In fact, it is this principle that guarantees the delivery of data over any kind of heterogeneous network.

The key idea of TCP Westwood is to exploit TCP acknowledgment packets to derive rather sophisticated measurements. We propose that a source perform an end-to-end estimate of the bandwidth available along a TCP connection by measuring and averaging the rate of returning ACKs.

After a congestion episode (i.e. the source receives three duplicate ACKs or a timeout) the source uses the measured bandwidth to properly set the congestion window and the slow start threshold, starting a procedure that we will call *faster recovery*. When an ACK is received by the source, it conveys the information that an amount of data corresponding to a specific transmitted packet was delivered to the destination. If the transmission process is not affected by losses, simply averaging the delivered data count over time yields a fair estimation of the bandwidth currently used by the source.

When duplicate ACKs (DUPACKs), indicating an out-of-sequence reception, reach the source, they should also count toward the bandwidth estimate, and a new estimate should be computed right after their reception.

However, the 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, allowing for corrections when the next cumulative ACK is received. For the sake of simplicity, we assume all TCP segments to be of the same size.

Following this assumption, we will further assume that sequence numbers are incremented by one per segment sent, although the actual TCP implementation keeps track of the number of bytes instead: the two notations are interchangeable if all segments have the same size.

It is important to notice that, immediately after a congestion episode, followed either by a timeout or, in general, *n* duplicate ACKs, the bottleneck is at saturation and the connection delivery rate is equal to the share of the best-effort bandwidth (i.e., saturation bandwidth) available to that connection. At steady state, under proper conditions, as stated in Section 3.3,

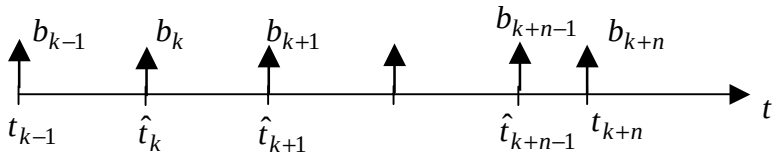

Figure.1. Bound on the maximum sampling interval obtained by inserting virtual sample

this actually should be the "fair share". The saturation condition is confirmed by the fact that packets have been dropped, an indication that one or more intermediate buffers are full. Before a congestion episode, the used bandwidth is less than or equal to the available bandwidth because the TCP source is still increasing its window to probe the network capacity.

As a result, TCP Westwood adjusts its input by taking into account the network capacity that is available to it at the time of congestion, whereas the current TCP Reno simply halves the congestion window.

2.2 Filtering the ACK reception rate.

If an ACK is received at the source at time t_k, this implies that a corresponding amount of data d_k has been received by the TCP receiver. Therefore, we can measure the following *sample* of bandwidth used by that connection as:

$$b_k = \frac{d_k}{t_k - t_{k-1}}$$

where t_{k-1} is the time the previous ACK was received. Since congestion occurs whenever the low-frequency input traffic rate exceeds the link capacity [15], 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 critical to filter out the noise due to delayed acknowledgments.

The choice of the filter is important. According to our experience, simple exponential averaging of the kind used by TCP for RTT estimation is unable to efficiently filter out high-frequency components of the bandwidth measurements. We propose the following discrete time filter which is obtained by discretizing a continuous low-pass filter using the Tustin approximation [19] We obtain:

$$\hat{b}_k = \frac{\frac{2\tau}{t_k - t_{k-1}} - 1}{\frac{2\tau}{t_k - t_{k-1}} + 1}\hat{b}_{k-1} + \frac{b_k + b_{k-1}}{\frac{2\tau}{t_k - t_{k-1}} + 1} \quad (1)$$

where $\hat{b}_k$ is the filtered measurement of the available bandwidth at time $t = t_k$, and $1/\tau$ is the cut-off frequency of the filter. The structure of filter (1) is simple. To understand how it works it is useful to consider a constant interarrival time $t_k - t_{k-1} = \Delta_k = \tau/10$. Then, filter (1) becomes a filter with constant coefficients

$$\hat{b}_k = a\hat{b}_{k-1} + \frac{(1-a)}{2}[b_k + b_{k-1}] \quad (2)$$

where $a = 0.9$. The particular form (2) shows that the new value $\hat{b}_k$ is made by the 90% of the previous value $\hat{b}_{k-1}$ plus the 10% of the arithmetic average of the last two sample b_k and b_{k-1}. Even though the filter (2) is useful to explain how the average is computed, it cannot be used because in packet-switched networks the interarrival time is not constant. To counteract the effect of time-varying interarrival time, the coefficients of filter (1) depend on $t_k - t_{k-1}$. The effect of using time-varying coefficients is easy to understand. When the interarrival time increases, the last value $\hat{b}_{k-1}$ should have less significance, since it represents an older value, whereas the significance of recent samples should be higher. This is exactly what happens with filter (1): the *a* coefficient decreases when the interarrival time increases meaning that the previous value $\hat{b}_{k-1}$ has less significance with respect to the last two recent samples which are multiplied by (1–*a*).

Finally, filter (1) has a cut-off frequency equal to $1/\tau$. This means that 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 or equal to $\tau/2$ is necessary. But, since the ACK stream may be irregular (for instance, no ACKs are returned when the sender is idle), the sampling frequency constraint cannot be guaranteed. To guarantee the Nyquist constraint and thus preserve the low pass filter effect, we establish that if a time τ/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 Fig. 1 below, 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, $b_{k+j} = 0$ for j=0,n-1 are the virtual samples and $b_{k+n} = \frac{d_{k+n}}{t_{k+n} - t_{k+n-1}}$ is the sample computed when the ACK is received at t_{k+n}.

It is interesting to look at the form that filter (1) takes in the persistent absence of ACKs from $t=t_k$: that is, in the absence of ACKs the estimated bandwidth exponentially goes to zero.

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

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

…..

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

In our experiments, *m* was set to 2.

2.3 On the effects of delayed and cumulative ACKs on bandwidth measurement

As previously stated, DUPACKs should count toward the bandwidth estimation, since their arrival indicates a successfully received segment, albeit in the wrong order. As a consequence, a cumulative ACK should only count as one segment's worth of data since duplicate ACKs ought to have already been taken into account. However, the matter is further complicated by the issue of *delayed ACKs*. The standard TCP implementation provides for an ACK being sent back once every other in-sequence segment received, or if a 200-ms timeout expires after the reception of a single segment [18]. The combination of delayed and cumulative ACKs can potentially disrupt the bandwidth estimation process.

We therefore stress two important aspects of the bandwidth estimation process:

a. The source must keep track of the number of DUPACKs it has received before new data is acknowledged;
b. The source should be able to detect delayed ACKs and act accordingly.

The approach we have chosen to take care of these two issues can be found in the *AckedCount* procedure, detailed below, showing the set of actions to be undertaken upon the reception of an ACK, for a correct determination of the number of packets that should be accounted for by the bandwidth estimation procedure, indicated by the variable *acked* in the pseudocode. The key variable is *accounted*, which keeps track of the received DUPACKs. When an ACK is received, the number of segments it acknowledges is first determined (*cumul_ack*). If *cumul_ack* is equal to 0, then the received ACK is clearly a DUPACK and counts as 1 segment towards the BWE; the DUPACK count is also updated. If *cumul_ack* is larger than 1, the received ACK is either a delayed ACK or a cumulative ACK following a retransmission event; in that case, the number of ACKed segments is to be checked against the number of segments already accounted for (*accounted_for*). If the received ACK acknowledges fewer or the same number of segments than expected, it means that the "missing" segments were already accounted for when DUPACKs were received, and they should not be counted twice. If the received ACK acknowledges more segments than expected, it means that although part of them were already accounted for by way of DUPACKs, the rest are cumulatively acknowledged by the current ACK; therefore, the current ACK should only count as the cumulatively acknowledged segments. It should be noted that the last condition correctly estimates the delayed ACKs (*cumul_ack* = 2 and *accounted_for* = 0).

```
PROCEDURE AckedCount

cumul_ack      =      current_ack_seqno      -
last_ack_seqno;
if (cumul_ack = 0)
  accounted_for=accounted_for+1;
  cumul_ack=1;
endif

if (cumul_ack > 1)
  if (accounted_for >= cumul_ack)
    accounted_for=accounted_for-cumul_ack;
    cumul_ack=1;
  else if (accounted_for < cumul_ack)
         cumul_ack=cumul_ack-
accounted_for;
         accounted_for=0;
  endif
endif

last_ack_seqno=current_ack_seqno;
acked=cumul_ack;

return(acked);

END PROCEDURE
```

3. TCP Westwood

In this Section we describe how the bandwidth estimation can be used by the congestion control algorithm executed at the sender side of a TCP connection. As will be explained, the congestion window dynamics during slow start and congestion avoidance are unchanged, that is they increase exponentially and linearly, respectively, as in current TCP Reno.

The general idea is to use the bandwidth estimate *BWE* to set the congestion window (*cwin*) and the slow start threshold (*ssthresh*) after a congestion episode.

We start by describing the general algorithm behavior after *n* duplicate ACKs and after coarse timeout expiration.

3.1 Algorithm after n duplicate ACKs

The pseudocode of the algorithm is the following:

```
if (n DUPACKs are received)

  ssthresh =  (BWE*RTTmin)/seg_size;
  if (cwin>ssthresh) /* congestion avoid.
*/
    cwin = ssthresh;
  endif

endif
```

Note that *seg_size* identifies the length of a TCP segment in bits.

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 slow start threshold is set equal to the available pipe size when the bottleneck buffer is empty, which is *BWE*RTTmin*, the congestion window is set equal to the *ssthresh* and the congestion avoidance phase is entered again to gently probe for new available bandwidth. The

value *RTTmin* is set as the smallest RTT sample observed over the duration of the connection. This setting allows the queue be drained after a congestion episode. During the slow-start phase we are still probing for the available bandwidth. Therefore the BWE we obtain after *n* duplicate ACKs is used to set the slow start threshold. After *ssthresh* has been set, the congestion window is set equal to the slow start threshold only if *cwin*>*ssthresh*. In other words, during slow start, *cwin* still features an exponential increase as in the current implementation of TCP Reno.

3.2 Algorithm after coarse timeout expiration

The pseudocode of the algorithm is

```
if (coarse timeout expires)
  ssthresh = (BWE*RTTmin)/seg_size;

  if (ssthresh < 2)
    ssthresh = 2;
  endif;

  cwin = 1;
endif
```

The rationale of the algorithm is again simple. After a timeout *cwin* and *ssthresh* are set equal to 1 and BWE, respectively, so that the basic Reno behavior is still captured, while a speedy recovery is granted by the ssthresh being set to the bandwidth estimation at the time of timeout expiration.

3.3 TCP Westwood convergence to fair share

An important goal of any TCP implementation is for every connection to get its "fair share" of the bottleneck. We will use an informal argument similar to that used for Reno in [14] to show that TCPW achieves the fair share. Consider the case of two connections with the same RTTs. Suppose, for the sake of example, that the RTT is X packet transmission times, and the bottleneck has X buffers. One connection, say A, starts first. Its window "cycles" between X and 2X (as per the TCPW algorithm described earlier in this Section), each cycle terminating when buffer overflows. Later, connection B starts, first in slow start mode, and then in congestion avoidance mode. In congestion avoidance, during each cycle the windows A and B grow approximately at the same rate, i.e., one segment per RTT. Eventually, the bottleneck buffer overflows, terminating the cycle. One can show that the window at overflow is:

$W_i = R_i (b/C + RTT)$, for i = A,B;

where R is the achieved rate (i.e., BWE); b is the bottleneck buffer size; and C is the bottleneck trunk capacity.

This is a general property true for all TCP protocols, and in particular for TCPW. After overflow, TCPW reduces the windows to new values W_i' as follows:

$W_i' = R_i (RTT)$ for i = A, B

Thus, the ratios of the windows A and B are preserved after overflow. Yet, the ratio W_B/W_A keeps increasing during congestion avoidance. Consequently, the B window and throughput ratchet up at each cycle. Equilibrium is reached when the two connections have the same windows and the same bandwidth fair share. Figure 2 graphically illustrates the convergence to the fix point $W_A=W_B$.

This informal proof is validated by actual simulation results. It can be generalized to many simultaneous connections (all with the same RTT). It can also be applied to the case when the bottleneck is affected by random errors equally hitting all connections.

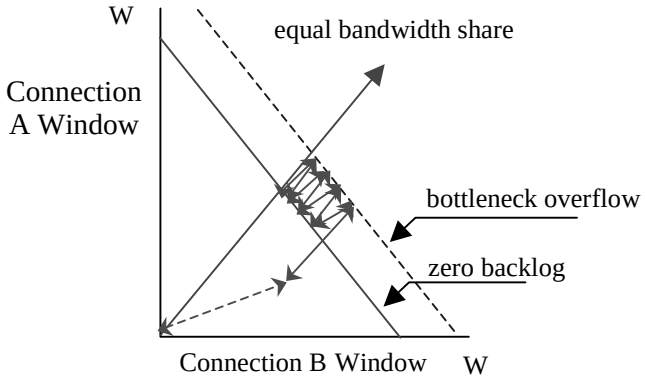

Figure 2. Convergence toward fair bandwidth sharing.

4. TCP Westwood Performance, Fairness, and Friendliness

In this Section, we report on the basic performance behavior of TCPW, its fairness among a number of TCPW connections sharing a bottleneck link, and its friendliness to coexisting connections of other TCP variants, such as Reno.

First, the effectiveness of the bandwidth estimation algorithm is studied using a single TCP connection and a fluctuating UDP traffic rate. TCPW window dynamics (*cwin*, *ssthresh* and sequence numbers) are then considered. TCPW performance behavior is compared to the standard and widely used TCP Reno as well as to TCP SACK [16].

All simulations presented in this paper were run using the LBL network simulator, 'ns' ver.2 [19].

New simulation modules for TCP Westwood were written and they are available at [20], while existing modules for simulations involving TCP Reno and TCP Sack were used. All simulated TCP receivers implement delayed-ACKs. Notice that this introduces a complication for our bandwidth estimation algorithm as delayed ACKs represent noise to be filtered, as explained in Section 2.3.

Each scenario, involving different bottleneck link capacity, RTT or number of concurrent connections, includes a single-bottleneck link as is common in the literature. Intermediate node buffer capacity is always set equal to the bandwidth-delay product for the scenario under study. The packet size is set to 400 bytes in all experiments. The ACK arrival pattern is

repetitive for each RTT in absence of packet losses (errors or buffer overflow). Thus, the interval τ should span one or more RTTs. Experimentally, we have observed that performance is not very sensitive to the choice of τ as long as τ>RTT. In our experiments, we set τ equal to 500ms.

4.1 Bandwidth estimation effectiveness

In this Section, we test the effectiveness of the proposed bandwidth estimation algorithm. For this purpose we consider a single TCPW connection sharing the bottleneck link with UDP connections. Packets are queued and transmitted on the link in FCFS order. In addition to demonstrating the accuracy of the bandwidth estimation algorithm, this scenario also illustrates the capability of a TCP Westwood connection to use the bandwidth left over by dynamic UDP flows. The configuration simulated here features a 5 Mb/s bottleneck link with a one-way propagation delay of 30ms. One TCP connection shares the bottleneck link with two ON/OFF UDP connections, and TCP and UDP packets are assigned the same priority. Each UDP connection transmits at a constant bit rate of 1 Mb/s while ON. Both UDP connections start in the OFF state; after 25s, the first UDP connection is turned ON, joined by the second one at 50s; the second connection follows an OFF-ON-OFF pattern at times 75s, 125s and 175s; at time 200s the first UDP connection is turned off as well. The UDP connections remain silent until the end of the simulation. The TCPW connection sends data throughout the simulation.

The scenario above is intended to demonstrate the effectiveness of the feedback control used in TCPW when subjected to "step" and "impulse" stimuli. The behavior of the bandwidth estimation process is shown in fig. 3.

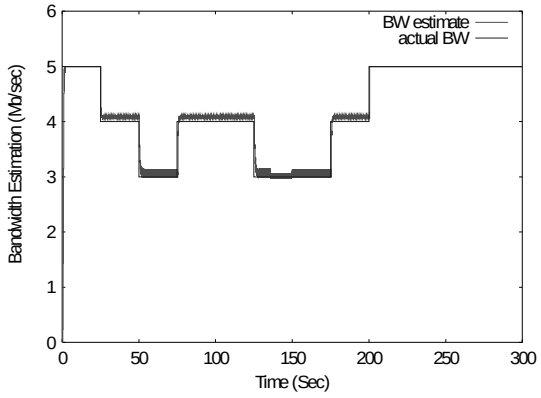

Figure 3. TCPW with concurrent UDP traffic: bandwidth estimation

4.2 TCPW fairness

Fair bandwidth sharing implies that all connections are provided with similar opportunity to transfer data. Our experiments show that TCPW fairness is at least as good, if not better, than that provided by the widely-used TCP Reno. In the sample results below we show that two flows with different E2E round trip times (RTT) share the bandwidth more effectively under TCPW than under TCP Reno.

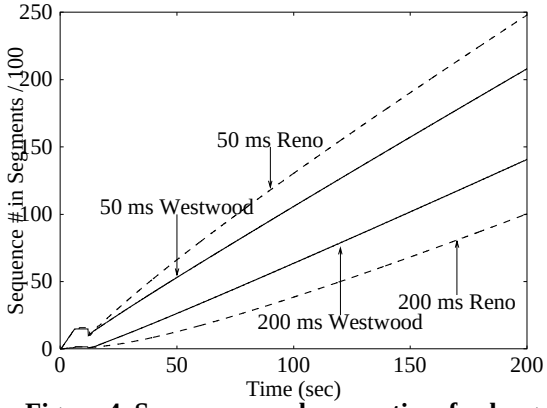

Figure 4. Sequence numbers vs. time for long and short RTT connections without RED

We ran simulations in which connections were subject to 50ms and 200ms RTT, respectively. Figure 4 and Figure 5 show sequence number progress for TCPW and Reno connections without and with RED, respectively. In all cases the short connection progresses faster as expected. We note however that TCPW provides better fairness than Reno across different propagation times. The reason for the superior fairness exhibited by TCPW is that the long connection suffers less reduction in *cwin* and *ssthresh*. In Reno, *cwin* reduction is independent of RTT. The results in Figure 5 show that both protocols benefit from RED, as far as fairness is concerned. Remarkably, the improvement in TCPW due to RED was higher than the improvement in Reno.

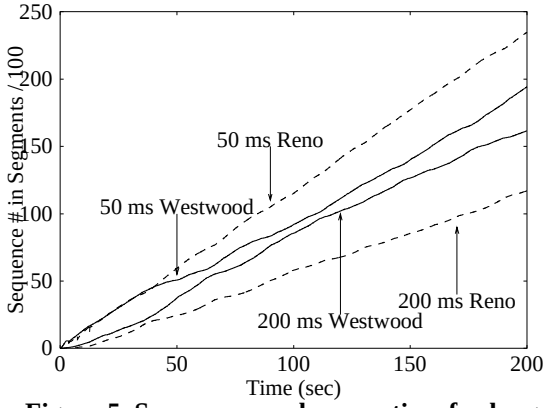

Figure 5. Sequence numbers vs. time for long and short RTT connections with RED

4.3 TCPW friendliness

Friendliness is another important property of a TCP protocol. TCPW must be "friendly" to other TCP variants. That is, TCPW connections must be able to coexist with connections from TCP variants while providing opportunities for all connections to progress satisfactorily. At least, TCPW connections should not result in starvation of connections running other TCP variants. Better yet, the bandwidth share of TCPW connections should be equal to their fair share.

We ran simulation experiments with the following parameters: 2-Mbps bottleneck link, 20 flows total, all flows with 100ms RTT. With all 20 connections running TCPW, the average throughput per connection was 0.0994 Mbps. All 20 Reno

connections resulted in an average throughput of 0.0992 Mbps. As predicted, we got the same results for the two schemes. We then ran 10 Reno with 10 Westwood connections sharing the same 2Mbps bottleneck link over a path of 100ms RTT. The average throughput for a TCPW connection went up to 0.1078, and that of a Reno connection went down to 0.0913. This shows that TCPW behavior departs from "fair share" by 16% (TCPW gains 8% and TCP Reno loses 8%). This unfairness is rather moderate and it can be tolerated as it allows for coexistence with Reno.

To probe the friendliness issue further, we also carried out actual measurements using our TCPW Linux implementation in our lab. Figure 6 shows the topology of our lab test bed. The link emulator is used to vary the link propagation time and error characteristics.

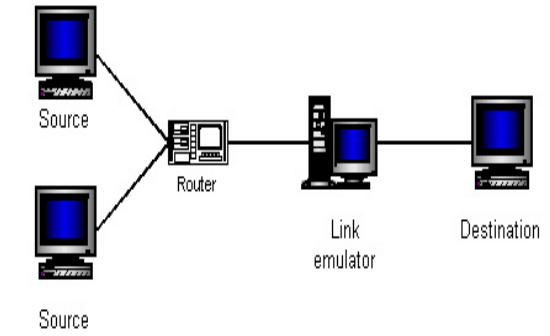

Figure 6. Experimental test bed layout

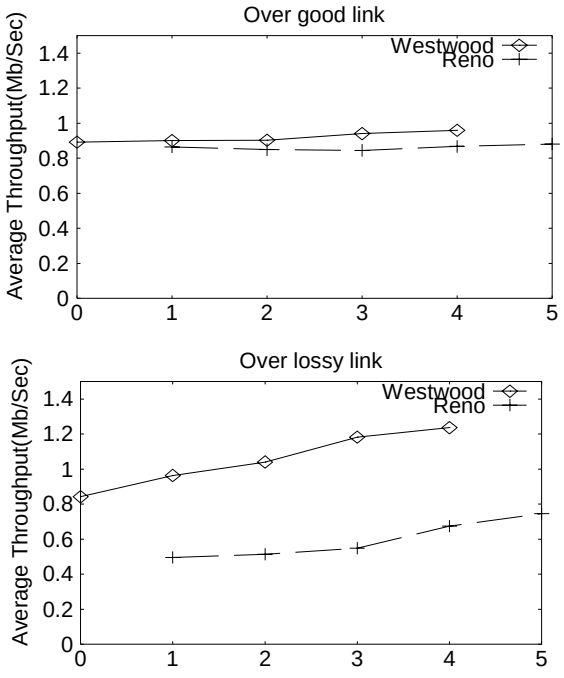

Figure 7. Average throughput vs. No. of Reno connections over good and lossy link (5 connections total)

We measured the throughput for a total of 5 connections with a variable Reno/ TCPW mix. Then, to evaluate the friendliness of TCPW under stress, we introduce a relatively high error rate on the bottleneck link, namely 1% packet loss (See Figure 7). This error rate is actually appropriate for wireless links as we shall discuss later. Note that TCP West shines in presence of line errors, so, friendliness in the error situation is even more difficult to establish than in error free operation.

The RTT was 100ms. Error rates and propagation delays are introduced in our test bed via a link/network emulator. Figure 7 shows the average throughput per connection for TCPW and for Reno. The lower average throughput line is that of the Reno connections. The horizontal axis represent the number of Reno connections in the mix. For example, at the point marked 3 on the horizontal line, the measurement experiment includes 3 Reno connections and 2 TCPW connections The results in Figure 7 illustrate two important points. First, TCPW has a significant edge in a high-error-rate environment: 5 TCPW get 10% more throughput than 5 TCP Reno. We will press more on this later. Secondly, friendliness is preserved. Even though TCPW has an advantage over Reno in error-prone environments, Reno connections were not starved. In fact, the introduction of TCPW connections into the mix reduces the average throughput of a Reno connection only by a minimal amount. Thus, for practical purposes, we can claim that TCPW is friendly.

5. TCPW Performance In Mixed (Wired and Wireless) Networks

TCPW is being proposed in this paper as an end-to-end solution to error and congestion control in mixed wired and wireless networks. In view of this claim, a number of different scenarios are studied below to show the benefits of using TCPW in such wired/wireless environments. Independent and correlated loss models are used. Ground radio as well as satellite scenarios are developed and studied.

5.1 Independent loss model in ground radio environment

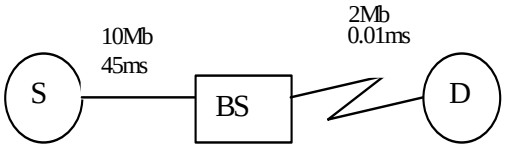

Figure 8. A simple Simulation Topology

Figure 8 shows a topology of a mixed network with a wired portion including a 10-Mbps link between a source node and a base station. The propagation time over the wired link is initially assumed to be 45ms. Later, the propagation time is varied from 0 to 250ms to represent a variety of wired network environments ranging from campus to intercontinental connections.

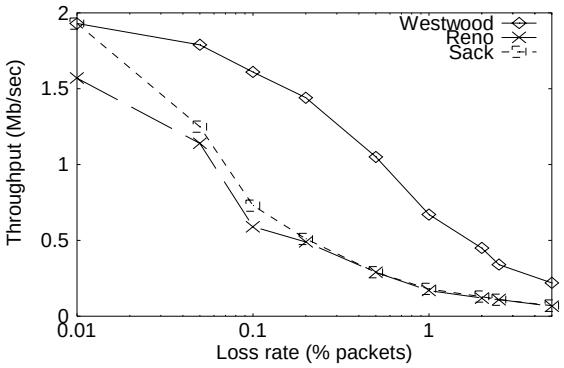

Figure 9. Throughput vs. error rate of the wireless link

The wireless portion of the network is a very short 2-Mbps wireless link with a propagation time of 0.01ms. The wireless link is assumed to connect the base station to a destination mobile terminal. Errors are assumed to occur in both directions of the wireless link.

We compare the throughput of TCPW to that of Reno and SACK assuming independent (Bernoulli) errors ranging from 0 to 5% packet loss probability. The error model assumed here is equivalent to the "exponential error" model in which the time between successive errors is exponentially distributed [3]. The range of error rates assumed here is also similar to the range used in [3]. The results in Figure 9 show that TCPW gains up to 394 % over Reno or SACK. This gain occurs at a realistic packet error probability of 1%.

To assess TCPW throughput gain and its relation to the E2E propagation time, we ran simulations with the wired portion propagation time varying from 0 to 250ms. The results in Figure 10 show a significant gain for TCPW of up to 567%, at a propagation time of 100ms. 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. TCPW reaches maximum improvement over Reno and Sack as the propagation time increases to about 100ms. After that, in this experiment, the gain starts to decrease as the feedback information used to estimate the available bandwidth arrives too late to be of significant help to TCPW.

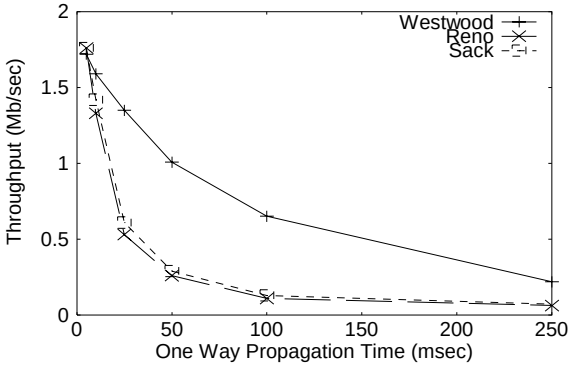

Figure 10. Throughput vs. one-way propagation delay

Simulation results in Figure 11 show that TCPW gains also increase significantly as the bottleneck link transmission speed increases (again, because what matters is the window size determined by the bandwidth-delay product). Thus, TCPW is more effective than TCP Reno in utilizing the Gbps bandwidth provided by new-generation, high-speed networks. Figure 11 shows that the improvement obtained via TCPW increases to approximately 550% when the wireless link speed reaches 8 Mbps. The error model is still Bernoulli with parameter 0.5%, and the E2E propagation time is 45ms.

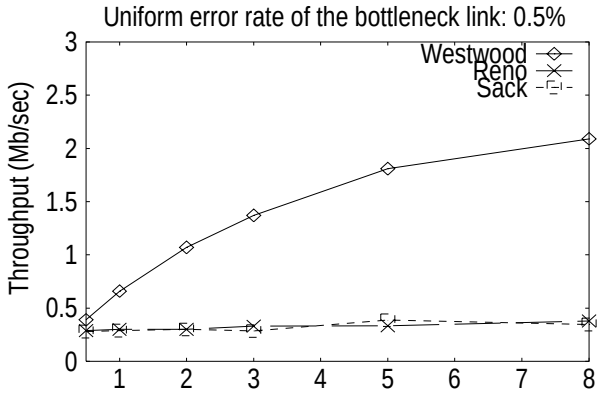

Figure 11 Throughput vs. link capacity

Window dynamics of TCPW and of TCP Reno are presented in Figure 12 and Figure 13 below. The graphs show the improved window dynamics in TCPW. The cwin and ssthresh values are consistently higher than the corresponding values in Reno, thus yielding higher throughput .

Next we compare TCPW to Snoop, the leading local strategy shown to provide the biggest improvement over TCP Reno [4]. Published results show that Snoop provides approximately a 400% improvement over an E2E approach based on TCP Reno when the error rate is 1 bit in 64 KBytes and the round trip propagation time is 135ms. Our simulations with similar parameter values show that TCPW provides a 382%

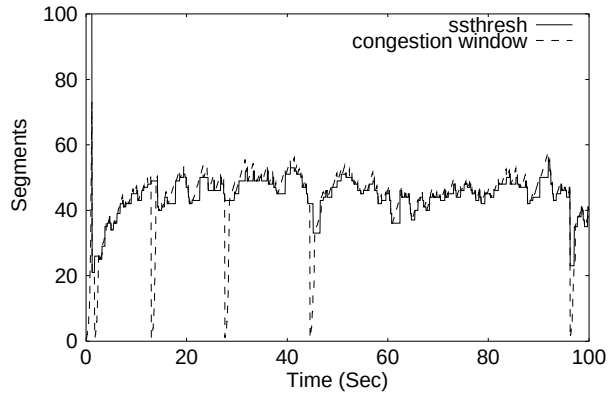

Figure12. TCP Westwood over lossy link—cwin and ssthresh

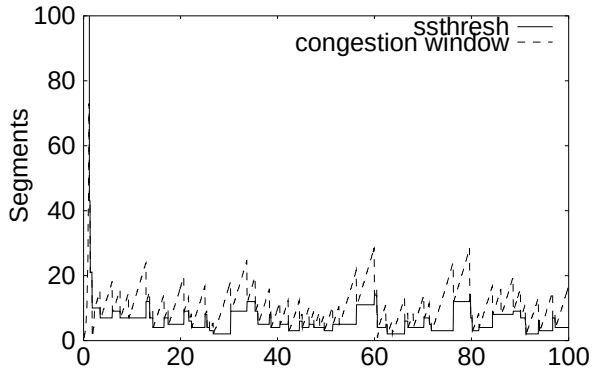

Figure 13. TCP Reno over lossy link—cwin and ssthresh

improvement over Reno. This shows that TCPW and Snoop gains are remarkably (and enticingly) close. We plan to probe further the issue of effectiveness of local versus E2E error recovery via simulation and measurements. From the qualitative and protocol implementation standpoint, however, we note that TCPW is completely end-to-end, and does not require any support from network or link layers. It does not have the scalability problems that Snoop may encounter as the number of mobile terminals increase. Further, the effectiveness of Snoop in wireless subnets including multiple base stations and handoffs is not clear.

Explicit Loss Notification (ELN) is an E2E scheme that is introduced and assessed in [3]. Basically, the method provides explicit notification from TCP receiver to TCP sender that a loss *due to a link error* has occurred. The lost packet is also identified to the Sender TCP entity. Using the same parameter values above (1 bit in 64 Kbytes error rate, and 135ms propagation time), ELN is shown to provide a gain of approximately 200% over Reno. In comparison, TCPW provides 382%, closer to Snoop performance. Further, ELN assumes that the destinations can detect errors on a link and identify the packet and its TCP source. These assumptions are not likely to be uniformly satisfied for various error causes and various link technologies, thus the limitation in versatility of ELN in addition to its limited gain over Reno.

We compared, via simulation, TCPW to BA-TCP [10], an alternative strategy where the routers explicitly measure and relay the bandwidth available for each connection back to the TCP sender. At 40ms round trip time, and 1 bit in 100KB error rate, BA-TCP's improvement over Reno is 202%. For TCP Westwood, the throughput improvement at the same parameter values is 161%. This is quite remarkable considering that BA-TCP measures the bandwidth actually available for a connection at the bottleneck router, while TCPW works with no support from routers to estimate the available bandwidth at the bottleneck. Note that the router functionalities required by BA-TCP are not available in today's routers.

5.2 Burst error models in a ground radio environment

To study TCPW performance with correlated errors, we use the 2-state Markov models following [1][5]. In such models, burst errors occur at a high rate due to a variety of conditions associated mostly with terminal mobility. Such conditions include variable fading, blackouts due to shadowing, and the like. Figure 14 below depicts the 2-state Markov model. The wireless link is assumed to be in one of two states: Good or Bad. In the Good state, a bit (or packet) error Bernoulli model is assumed. The time intervals between bit errors is thus exponentially distributed (memoryless channel errors). In addition, a link is assumed to stay in the Good state a time interval that is exponentially distributed with parameter λ_{gb} . The time spent in the Bad state is also exponentially distributed but with parameter λ_{bg}. In the Bad state we assume that errors are still Bernoulli, however, the rates of errors in the Bad state are much higher. For the simulation experiments below we vary the error rate in the bad state depending on the specific link conditions we want to study. To represent fading conditions, the bit error rate is assumed to range from 0 to 30%. For blackouts, the error rate is 100%.

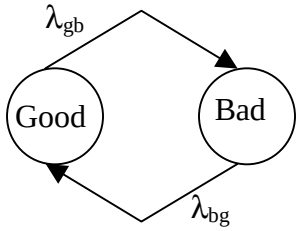

Figure 14. 2-state Markov Model for Burst Error Characterization

Simulation results using the 2-state Markov models show that TCPW improves throughput for links with fading and blackouts as discussed below.

(a) Fading:

Let the Bad state represent fading conditions, and let the mean duration of Good and Bad states be 8 and 4 sec respectively. The error rate in the Good state is assumed to be 0.001% packet loss, and the error rate in the Bad state is varied from 0 to 30% packet loss rate. The results in Figure 15 show the improvement obtained with TCPW over Reno or SACK. TCPW increases throughput by up to 300%. This is achieved when the error rate in the Bad state is 5 %. When the error rate is higher, all protocols perform poorly.

When the error rate is less than 5%, TCPW provides about 150% improvement. We also varied the link speed to determine its impact on the protocols performance.

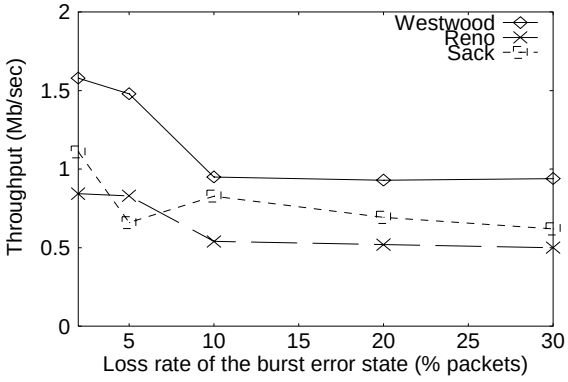

Figure 15. Throughput vs. error rate of the bad state

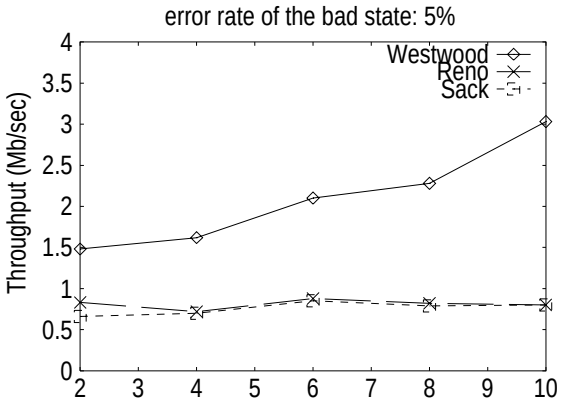

Figure. 16. Throughput vs. link capacity in 2-state error model

Figure 16 shows that TCPW improvement increases as the wireless link (bottleneck link in this case) increases (as expected, since a similar trend was observed also in wired links). At 10 Mbps link speed, a 400% throughput improvement is achieved at the same error rate in Bad state of 5% packet loss.

(b) Blackouts:

Let us now assume that the Bad state represents a blackout, where a base station becomes temporarily not visible to a terminal due to mobility. The mean duration for the Good state is 4 sec; the mean duration of the Bad state varies between 0 and 0.5 sec, Figure 17 shows the throughput improvement obtained by TCPW to be 167 % over Reno and SACK when the mean blackout duration is 0.1 sec. For longer blackouts, TCP timeouts occur and all protocols are equally affected.

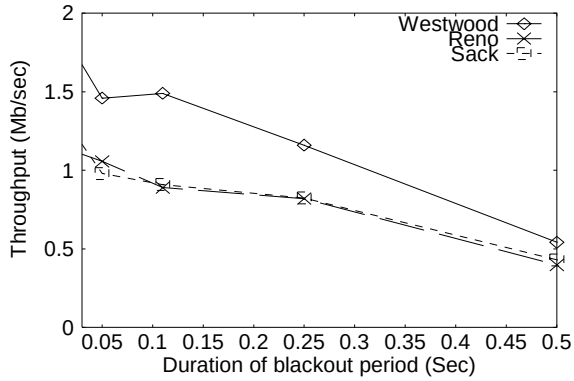

Figure 17 Throughput vs. average duration of blackout

5.3 LEO Satellite Model

Another environment where TCPW is likely to be valuable is the LEO satellite system. LEO Satellites present an environment with varying link quality and relatively long propagation delay. Also, in the future, higher transmission speeds are expected. That is where TCPW would be most beneficial.

We considered for this study a scenario where a single hop, up to the satellite and down to an earth terminal, connects a terminal to a gateway and from there to the terrestrial network. One way (e.g. terminal to gateway) propagation time is assumed to be 100ms. The error rate is assumed 0.1% in normal operating conditions. Occasionally, if the LEO system supports satellite diversity, a handoff to a different LEO satellite (different orbit) becomes necessary to overcome the blocking due to buildings, thick foliage etc. During handoff, we assume all packets are lost. In our model, the handoff from one satellite to another needs 100ms to complete, and the period between handoffs is 4s, say. Figure 18 below shows the major improvements of TCPW over Reno and SACK, especially at high speeds.

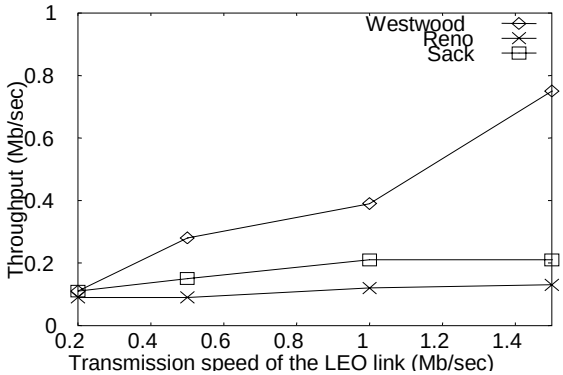

Figure 18. Throughput vs. link capacity of the Satellite link

6. Conclusions and Future Research

In this paper we have proposed a new version of the TCP protocol, TCP Westwood, (TCPW for short) aimed at improving performance under random or sporadic losses. TCPW has been tested through simulation, showing considerable throughput gains in almost all wireless scenarios.

In retrospect, the new scheme can be viewed as one more step in the TCP evolution. TCP Tahoe resets *cwin* to one after a loss. TCP Reno halves *cwin* after three duplicate ACKs. TCP Westwood introduces a "faster" recovery mechanism to avoid over-shrinking *cwin* after three duplicate ACKs. It does so by taking into account the end-to-end estimation of the bandwidth available to TCP. The use of bandwidth estimation feedback to control the congestion window has an effect that goes beyond faster recovery. Namely, TCP window congestion control is based not solely on packet loss (which itself is an ambiguous congestion indicator in presence of wireless links), but also on the bandwidth estimate at the time of loss. The benefits of using bandwidth estimation (in addition to packet loss) have been amply demonstrated in a very broad range of wireless scenarios.

The issue of friendliness, raised by previous reviewers of this work, has been addressed. A qualitative proof of fair behavior under appropriate conditions has been provided. “Unfriendly” trends due to TCPW “aggressiveness” have been detected in our experiments, but were shown to be contained and never severe enough to lead to starvation.

The code modifications required to implement TCP Westwood are comparable to the ones implemented in the transition from TCP Tahoe to TCP Reno. As in the Tahoe to Reno transition, a

major advantage of the TCP Westwood modification is that it affects only the source TCP (as opposed to other variants such as TCP SACK that require also destination modifications). This allows a TCP Westwood source to interwork with arbitray destinations in the Internet.

Work is in progress in many directions. Some of these directions were indicated by the anonymous referees – we thank them for their thorough reviews and valuable comments. We are planning to include in TCPW the NewReno feature that allows efficient recovery from multiple losses in the same window. We are aware that in some cases the bottleneck link is in the backward path, from receiver to sender. In such cases, the bottleneck must be ”fairly” shared among Data Packets (in some connections) and ACKs (in some other connections). We plan to attack this problem by defining first a suitable measure of “fairness” between Data and ACK streams. If fairness is defined as equal throughput for all connections regardless whether Data or ACK bound, and Data packet size is the same for all connections, one can show that TCPW provides a fair solution – at equilibrium all connections measure the same BWE. The comparison of TCPW with link level techniques such as Snoop deserves further study. It is clear that link level recovery is in general much more powerful than end to end recovery since it isolates and corrects the loss “in loco”. For instance, suppose that the bottleneck is in the wired network and one of the connections sharing the bottleneck goes over a wireless, lossy link. With E2E recovery (TCPW and TCP Reno alike) the wireless connection is heavily penalized with respect to the others. With link layer recovery (eg, Snoop) fair sharing is enforced. Next, TCPW performs poorly when random packet loss rate exceeds a few percent. Snoop, on the other hand, is quite robust to high error rates. We are now investigating TCPW enhancements that will in part correct these deficiencies.. We plan to further refine our bandwidth estimation and filtering method, in order to improve TCPW “friendliness”. Finally, we intend to pursue the development of control theoretical models that will enable us to study the stability of TCPW as a function of the various systems parameters.

7. Acknowledgments

The authors take great pleasure in acknowledging the valuable contribution to this work by Tienshiao Ma, Bryan Ng, Giovanni Pau and Cathy Yang. who valiantly implemented TCPW under Linux and conducted the lab measurements experiments reported above.

8. References

[1] Abouzeid, A.A, S. Roy, M. Azizoglu. Stochastic Modeling of TCP over Lossy link. INFOCOM 2000, Tel Aviv, Israel, March 2000.

[2] Åström, K. J., B. Wittenmark, Computer controlled systems, Prentice Hall, Englewood Cliffs, N. J., 1997].

[3] Balakrishnan, H., 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.

[4] Balakrishnan, H., S. Seshan, E. Amir and R. H. katz. Improving TCP/IP Performance Over Wireless Networks. MOBICOM’95, Berkeley, CA, USA, November 1995

[5] Balakrishnan, H., and Randy H. Katz. Explicit Loss Notification and Wireless Web Performance. Proc. IEEE GLOBECOM’98 Internet Mini-Conference, Sydney, Australia, November 1998.

[6] Bonald, T., Comparison of TCP Reno and TCP Vegas: Efficiency and Fairness. In Proceedings of PERFORMANCE'99, Istanbul, Turkey, October 1999.

[7] Casetti, C., M. Gerla, S. Lee, S Mascolo and M. Sanadidi, "TCP with Faster Recovery", MILCOM 2000, Los Angeles, CA, October 2000.

[8] Clark, D., "The design philosophy of the DARPA Internet protocols", Proc. of Sigcomm88 in ACM Computer Communication Review, vol. 18, no. 4, pp. 106-114, 1988.

[9] Gerla, M., R. Lo Cigno, S. Mascolo, and W. Weng. Generalized Window Advertising for TCP Congestion Control. CSD-TR 990012, UCLA, CA, USA, February 1999.

[10] Gerla, M., W. Weng and R. Cigno, "Bandwidth feedback control of TCP and real time sources in the Internet", GLOBECOM‘2000, San Francisco, CA, USA, November 2000.

[11] T. Henderson, Satellite Transport Protocol Specification. Technical Report, University of California, Berkeley, 1999.

[12] Hengartner, U., J. Bolliger, and T. Gross. TCP Vegas Revisited. In Proceedings of IEEE INFOCOM'2000, Tel Aviv, Israel, March 2000.

[13] Jacobson, V., Congestion Avoidance and Control. ACM Computer Communications Review, 18(4):314--329, August 1988.

[14] Kurose, J., and K. Ross, “Computer Networking: A Top-Down Approach Featuring the Internet”, Addison Wesley, 2000

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

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

[17] Mitzel, D., Overviw of 2000 IAB Wireless Internetworking Workshop, RFC 3002.

[18] Stevens, W.R. TCP/IP Illustrated, vol. 1. Addison Wesley, Reading, MA, USA, 1994

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

[20] TCP Westwood modules for ns-2. URL: http://www.telematics.polito.it/casetti/tcp-westwood.