Bandwidth Estimation Schemes for TCP over Wireless Networks

Antonio Capone, *Member*, *IEEE*, Luigi Fratta, *Fellow*, *IEEE*, and Fabio Martignon, *Student Member*, *IEEE*

Abstract—The use of enhanced bandwidth estimation procedures within the congestion control scheme of TCP was proposed recently as a way of improving TCP performance over links affected by random loss. This paper first analyzes the problems faced by every bandwidth estimation algorithm implemented at the sender side of a TCP connection. Some proposed estimation algorithms are then reviewed, analyzing and comparing their estimation accuracy and performance. As existing algorithms are poor in bandwidth estimation, and in sharing network resources fairly, we propose TIBET (Time Intervals based Bandwidth Estimation Technique). This is a new bandwidth estimation scheme that can be implemented within the TCP congestion control procedure, modifying only the sender-side of a connection. The use of TIBET enhances TCP source performance over wireless links. The performance of TIBET is analyzed and compared with other schemes. Moreover, by studying TCP behavior with an ideal bandwidth estimation, we provide an upper bound to the performance of all possible schemes based on different bandwidth estimates.

Index Terms—TCP, bandwidth estimation, wireless.

1 Introduction

The Transmission Control Protocol (TCP) has proved efficient in classical wired networks, showing an ability to adapt to modern, high-speed networks and new scenarios for which it was not originally designed. However, the extraordinary success of modern wireless access networks, such as cellular networks, wireless local area networks, and of new applications for mobile computing environments, poses new challenges to the TCP congestion control scheme.

The existing versions of TCP, like Reno or NewReno, experience heavy throughput degradation over channels with high error rate, such as wireless channels. The main reason for this poor performance is that the TCP congestion control mechanism cannot distinguish between packet losses occurring randomly in wireless channels and those due to network congestion. Therefore, the TCP congestion control mechanism reduces, even when not necessary, the transmission rate [1], [2].

TCP maintains two state variables to regulate the transmission rate: congestion window (*cwnd*) and slow start threshold (*ssthresh*). The latter sets the *cwnd* value that discriminates between the slow-start and congestion avoidance phases. At the beginning of the connection, the source increases *cwnd* exponentially (slow start) until the network drops packets and a condition of congestion is recognized. In response to this, TCP Reno sets *ssthresh* to one half of the bytes in flight. When *cwnd* reaches the new *ssthresh* value, TCP enters a congestion avoidance phase during which *cwnd* is increased linearly.

This scheme assumes that *ssthresh* gives an estimate of the available bandwidth, and uses the congestion avoidance phase to probe gently for extra bandwidth. However, the bandwidth estimate is accurate only if the first packet loss occurs when the sending rate has reached the available rate. If the loss is due to transmission error, as in wireless channels, *ssthresh* can be set erroneously to a small value, thus limiting the sending rate and degrading the throughput performance.

To avoid such limitation and degradation, several schemes have been proposed and are classified in [3], as end-to-end protocols, where loss recovery is performed by the sender, split connection protocols, that break the end-to-end connection into two parts at the base station, and link-layer protocols based on a combination of ARQ and FEC techniques. The link-layer schemes have been shown to improve significantly the performance of TCP sources when transmitting over wireless links [3].

However, end-to-end techniques, even if not as effective as link-layer protocols, can achieve further gain in performance by using more sophisticated bandwidth estimation algorithms in the TCP congestion control scheme. A possible way to reduce the throughput degradation due to transmission errors is to consider the bytes in flight when a loss is detected in addition to the past history of the connection [4], [5].

This paper first discusses the problem of end-to-end bandwidth estimation for TCP, and points out issues that could affect both the estimation accuracy and its impact on TCP tunable parameters. Then, estimation algorithms proposed in the literature are reviewed and analyzed. Although not necessarily related to the wireless channel environment, various bandwidth estimation algorithms have been proposed to set the first value of *ssthresh* [6], [7], [8]. The algorithm implemented by TCP Vegas is described in [9], and those adopted by TCP Westwood are presented in [4], [10]. Our analysis of the estimation algorithms proposed for TCP Westwood reveals an overestimation of the available bandwidth, leading to aggressive

- *The authors are with the Department of Electronics and Information, Politecnico di Milano, p.zza Leonardo da Vinci 32, 20133 Milan, Italy. E-mail: {capone, fratta, martignon}@elet.polimi.it.*

Manuscript received 29 Jan. 2003; revised 19 June 2003; accepted 15 Dec. 2003.
For information on obtaining reprints of this article, please send e-mail to: tmc@computer.org, and reference IEEECS Log Number TMC-0004-0103.

and unfair behavior that prevents the smooth introduction of the new TCP version into the Internet.

To obtain more accurate, unbiased, and stable bandwidth estimates, needed for a fair sharing of the network resources, we propose a new algorithm, TIBET (Time Intervals based Bandwidth Estimation Technique). Like the algorithms used in TCP Vegas and TCP Westwood, TIBET requires modifications only at the sender-side since there is no need for cooperation from the peer TCP. To show the benefits of the proposed scheme in networks affected by independent or correlated losses, typical of a wireless environment, we compare the performance of TIBET with that of other schemes. Moreover, by studying TCP behavior with an ideal bandwidth estimation, we provide an upper bound to the performance of all possible schemes based on different bandwidth estimates.

The paper is structured as follows: Section 2 studies the problem of bandwidth estimation performed at the sender-side, and reviews the existing estimation techniques. Section 3 presents TIBET and shows, by simulation, that it copes efficiently with the problems presented in Section 2. Section 4 shows TIBET performance on wireless links, affected by both independent and correlated losses. Also provided is an upper bound to the performance of these schemes, achieved by studying the behavior of TCP with an ideal bandwidth estimation. Section 5 concludes the paper.

2 Bandwidth Estimation

The greater the amount of information available, the better the estimate by the TCP protocol of the bandwidth available for a connection; this results in a better and fairer utilization of network resources. This is the principle followed by several bandwidth estimation techniques proposed in the literature.

One approach, proposed in [6], [8], is to monitor the time spacing between received acknowledgements (ACKs) at the sender. A sample of the ACK bandwidth, i.e., the bandwidth promised by the ACK to the sender [11], is obtained by dividing the amount of acknowledged bytes by the interarrival time between consecutive ACKs. Some filtering techniques can be added to the sample sequence to smooth fast variations, and to reduce the impact of random losses. As proposed in [4], the TCP slow start threshold (*ssthresh*) is related to the *byte-equivalent* of the estimated bandwidth (Bwe) according to the following relation:

$$Ssthresh = Bwe \cdot RTT_{min}, \tag{1}$$

where RTT_{min} is the lowest Round Trip Time (RTT) measured by the TCP connection. This value can be considered an estimate of the Round Trip Time of the connection when the network is not congested.

2.1 Estimation Problems

Due to the peculiar transmission timing of the packets injected into the network by the TCP, and to the uncertainty with which a TCP source measures time intervals and estimates the minimum RTT, the following problems arise:

- Clustering [11], [12],
- ACK Compression [11], [12],
- TCP coarse-grained clocks [9], [13], and
- Rerouting [14].

2.1.1 Clustering

It is well-known that packets belonging to different TCP connections sharing the same link do not intermingle; therefore, many consecutive packets of the same connection can be observed on a single channel [11], [12]. This means that each connection uses the full bandwidth of the link for the time needed to transmit its cluster of packets. Thus, to correctly estimate the bandwidth in use, a TCP source must observe its own link utilization for a time longer than the time needed for entire cluster transmission, and the filtering technique, adopted to smooth the bandwidth samples, must operate for a long enough time interval to take all the samples into account. The appropriate minimum observation time depends on how many connections share the link, and on the cluster size that, in turn, depends on the bandwidth-delay product.

2.1.2 ACK Compression

ACK compression occurs when the time spacing between the received ACKs is altered by the congestion of the routers on the return path [12]. In fact, when a packet cluster reaches its destination a cluster of ACKs is generated. If these ACKs encounter a congested node, they lose their original time spacing since, during their forwarding, they are spaced by the short ACK segment transmission time. The result is ACK compression, that can lead to overestimation of the bandwidth in use. Such error depends on the ratio between the length of the full-size TCP data packets and the length of the ACK packets. In a typical situation where ACKs are 40 bytes long (the length of the TCP/IP headers) and data packets are 1,500 bytes long, the overestimation of the available bandwidth based on ACK bandwidth can be 37.5 times the actual value. Most TCP implementations adopt delayed ACKs, and the overestimation can even double its actual value. Therefore, ACK compression, commonly observed in real network operation [11], cannot be neglected.

2.1.3 TCP Coarse-Grained Clocks

TCP must translate the estimated bandwidth into parameters used in its congestion control scheme. It has been shown [7] that the optimal value for the slow start threshold is equal to the packets in flight in a pipe when the TCP rate equals the available bandwidth, i.e., when its transmission window is equal to the bandwidth-delay product. As pointed out earlier, the slow start threshold (*ssthresh*) can be set equal to the byte-equivalent of the estimated bandwidth (Bwe) according to (1).

However, TCP measures RTT with a *coarse-grained* clock [13]. As a consequence, the precision of the RTT_{min} estimate strongly depends on the TCP clock granularity, G. For example, if TCP runs over a LAN with a propagation delay equal to $G/10$, the RTT_{min} is set equal to G, a value 10 times higher than the correct one. Therefore, even if the estimated bandwidth value is correct, the *ssthresh* would be set to 10 times its correct value, thus leading to a very aggressive behavior of the connection.

2.1.4 Rerouting

When, during a connection, the routing-path changes, the hosts are not notified directly. However, if the new route has a shorter propagation delay, the RTT_{min} in (1) is updated correctly. On the contrary, if the new route has a longer propagation delay, the connection is not able to distinguish whether the increased RTT is due to sudden network congestion or to a new longer route, thus resulting in a wrong RTT_{min} estimate.

2.2 Estimation Algorithms

The literature proposes several bandwidth estimation schemes for TCP congestion control. Here, we review some of them, pointing out their characteristics and their ability to cope with the problems mentioned above.

It should be pointed out that the only quantity measured efficiently with a sender-side-only algorithm is the bandwidth *used* by the TCP source, not that *available*. Here, we define these two quantities following the guidelines of [8]: *available bandwidth* is the maximum rate at which a TCP connection, exercising correct congestion control, would transmit ideally; *used bandwidth* is the rate at which the source is actually sending data.

2.2.1 Packet-Pair Algorithm

The Packet Pair algorithm [6] and its variants have been proposed to be used by TCP sources at the beginning of the connection. Their main goal is to set the first value of the *ssthresh* in order to mitigate the effect of multiple losses due to the high default value commonly used [7]. Though the *ssthresh* should be set to the byte equivalent of the *available* bandwidth, the proposed schemes estimate the *bottleneck* bandwidth, that can be tracked more easily by analyzing the timing structure of received acknowledgments (ACKs). The *Packet Pair* algorithm is based on the assumption that, if two data packets are sent with closely spaced timing (back-to-back), their interarrival time at the receiver directly reflects the bottleneck bandwidth along the path. If, also, the returning path is uncongested, the corresponding ACKs are received at the TCP sender with the same spacing. The TCP source can thus estimate the bottleneck bandwidth by dividing the length of the sent data packets by the interarrival time between the corresponding ACKs.

Several enhancements have been proposed recently to the Packet Pair algorithm [15], [16] to improve the estimate of the bandwidth available along a path. However, most of these techniques are based on active schemes that inject additional traffic in the path and are not considered in this paper since we focus our analysis on passive techniques that simply exploit regular TCP traffic.

2.2.2 TCP Vegas Estimation Algorithm

A more sophisticated bandwidth estimation scheme, active throughout the connection time, has been adopted in TCP Vegas [9]. This scheme computes the difference between the *expected* and the *actual* flow rate that are defined by $cwnd/RTT_{min}$ and $cwnd/RTT$, respectively. RTT_{min} is the minimum RTT measured by the TCP source.

TCP Vegas adjusts the congestion window size based on the observation that when the network is not congested, the actual flow rate is close to the expected one, while, when the network is congested, the actual rate is smaller than the expected flow rate. More precisely, whenever an ACK is received, TCP Vegas computes the quantity $diff = (expected\_Rate - actual\_Rate) \cdot RTT_{min}$. The congestion window size ($cwnd$) is then increased by one if $diff < 1$, decreased by one if $diff > 3$ and left unchanged if $1 \leq diff \leq 3$.

Although the TCP Vegas algorithm often leads the congestion window size to an equilibrium point [9], it can, in homogeneous scenarios, fail to achieve fairness since competing connections can converge to different *cwnd* parameter values.

Moreover, in order to estimate the propagation delay of the network path in use, TCP Vegas measures the minimum RTT, and it can therefore suffer from the rerouting and persistent congestion problems, as pointed out in [14].

In spite of its improved performance, TCP Vegas has not yet been introduced into the Internet mainly because, in mixed scenarios with TCP Reno sources, the TCP Vegas sources would receive very little throughput [14].

2.2.3 TCP Westwood Estimation Algorithms

TCP Westwood, recently proposed in [17], [4], [18], [10], estimates the available bandwidth by measuring the rate of acknowledgments. This estimate is used to set the *ssthresh* and the *cwnd* after congestion events, such as the receipt of three duplicate ACKs or coarse timeout expirations. This recovery mechanism avoids the blind halving of the sending rate of TCP Reno after packet losses and enables TCP Westwood to achieve a high link utilization in the presence of the random, sporadic loss typical of wireless links.

The algorithm adopted by TCP Westwood, as reported in [17], considers the sequence of bandwidth samples *sample_BWE[k]* obtained using the ACK arrivals and evaluates a smoothed value, *BWE[k]*, by low-pass filtering the sequence of samples, as described by the following pseudocode:

```
Algorithm WESTWOOD 1:

if (ACK is received)
sample_BWE[k] = (acked * pkt_size * 8)
                 /(now - last_ACK_time);
BWE[k]= (1 - beta)*(sample_BWE[k] +
          sample_BWE[k-1])/2 +
          + beta*BWE[k-1];
endif
```

where *acked* is the number of segments acknowledged by the last ACK, *pkt_size* is the segment size in bytes, *now* is the current time, *last_ACK_time* is the time the previous ACK was received, *k* and *k-1* indexes indicate the current and previous value of the variables, and *beta* is the pole used for the filtering (in [17] a value of $beta = 19/21$ is suggested).

We have shown in [5] that Algorithm Westwood 1 usually provides a biased estimate mainly because it filters the bandwidth samples directly with a fixed pole filter. This cannot provide an unbiased value just as the arithmetic average of the bandwidth samples is not equal to the average bandwidth.

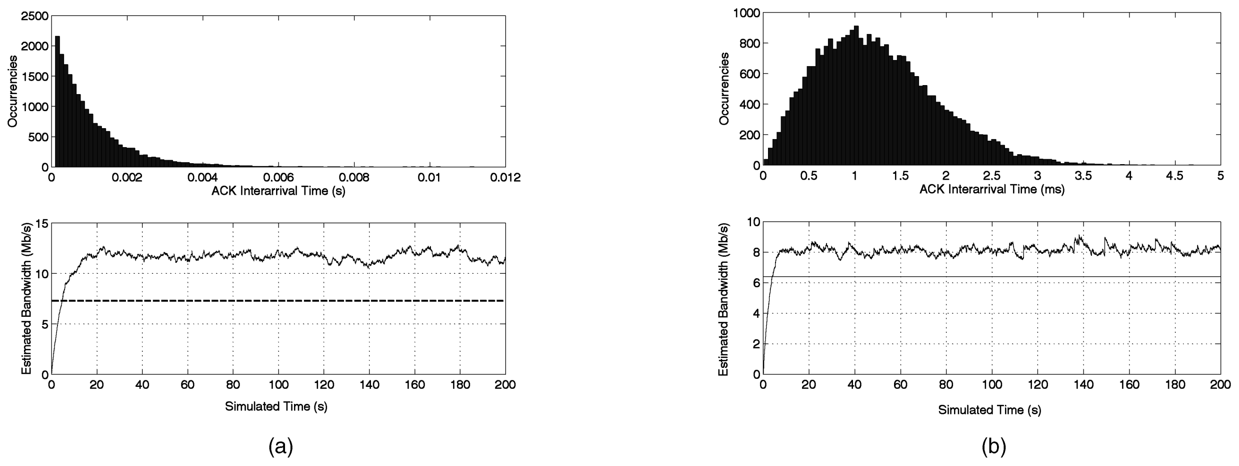


Fig. 1. Algorithm Westwood 2 filter: estimated bandwidth values with constant packet length equal to 1,000 bytes and random independent interarrivals: (a) exponentially distributed and (b) Rayleigh distributed. The dotted lines in the two lower figures represent the correct bandwidth estimate equal to the packet length divided by the average interarrival time.

The improved version of the TCP Westwood algorithm proposed in [4] and described in the following pseudocode adopts a nonlinear filtering technique:

```
Algorithm WESTWOOD 2:

if (ACK is received)
ACK_interval = now - last_ACK_time;
sample_BWE[k] = (acked * pkt_size * 8)
                  /ACK_interval;
pole= (2*tau - ACK_interval)/(2*tau +
        ACK_interval);
BWE[k]= (1 - pole)*(sample_BWE[k] +
          sample_BWE[k-1])/2 +
          + pole*BWE[k-1];
endif
```

We analyzed the performance of this algorithm. First, we tested the filter with constant packet lengths (equal to 1,000 bytes) and sequences of interarrival times with various probability distributions. Fig. 1 shows the estimates produced by the filter for exponential and Rayleigh distributed interarrival times. The dotted lines in the two lower figures represent the correct bandwidth estimate.

In both the scenarios, it was found that the TCP Westwood filtering algorithm consistently overestimated the available bandwidth.

We then evaluated the performance of TCP sources adopting algorithm Westwood 2, by simulation using the Network Simulator, "ns" ver.2 [19]; this was the simulator used for all the results presented in this paper. We considered 10 TCP Westwood connections sharing a single 10 Mbit/s link with a RTT of 100 ms. It was assumed, as for all the numerical results presented in the paper, that the TCP Maximum Segment Size is 1,000 bytes, that the TCP receiver implements the Delayed ACK algorithm as recommended in [20], and that the bottleneck buffer queue can contain a number of packets equal to the bandwidth-delay product of the connection. The time trace of the estimated bandwidth (Fig. 2) shows fast variations, and its average value (2.23 Mb/s) is higher than the fair share. To check algorithm behavior in the presence of ACK Compression, we also considered a scenario with a congested return path. Fig. 3 shows the time trace obtained considering a 2 Mbit/s link and two TCP Westwood connections sending data packets in the two directions. In the time interval between 40 and 140 seconds the TCP connection in the opposite direction transmits packets and a dramatic increase in the estimated values can be observed, showing that TCP Westwood is unable to account for ACK compression.

To investigate the effect of a nonaccurate estimate on the overall performance, we considered a mixed scenario where 10 TCP connections using either TCP Westwood or TCP Reno share a 10 Mb/s link. By simulation, we measured, for each connection, the goodput defined as the bandwidth actually used for successful transmission of useful data (payload). Fig. 4 shows the average goodputs of TCP Westwood and TCP Reno connections, versus the number of TCP Reno connections. It can be seen that, while in the nonmixed scenarios both TCP Reno and TCP Westwood achieve a fair sharing, in the mixed scenario, the TCP Westwood connections behave more aggressively. The TCP Westwood sources always achieve a goodput higher than the fair share, with a consequent starvation of the TCP Reno sources. Such unfair behavior, already discovered in TCP Vegas, prevents the smooth introduction of TCP Westwood into the Internet.

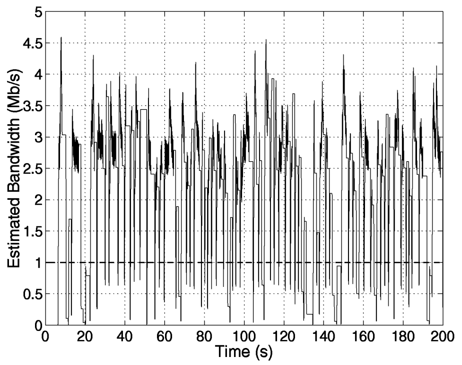


Fig. 2. TCP Westwood: estimated bandwidth with 10 connections sharing a single 10 Mbit/s link.

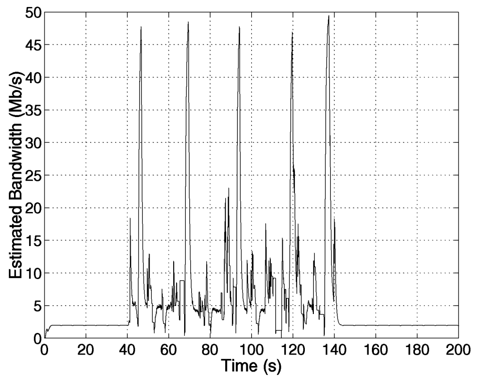


Fig. 3. TCP Westwood: estimated bandwidth in the presence of ACK compression. Two connections cross a 2 Mbit/s link in opposite directions.

Recently, the estimation algorithm of TCP Westwood was modified further [18], [10], the new approach adopting time varying coefficients in the filter with both adaptive gain and adaptive sampling. The following pseudocode specifies the modified approach [18], [10]:

```
Algorithm WESTWOOD 3:

sample_BWE[k]=(Bytes received in T[k])/T[k]
pole[k]=(2*tau[k] - ACK_interval)/(2*tau[k] +
        ACK_interval);
BWE[k]= pole[k]*BWE[k-1] + (1 - pole[k])*
        sample_BWE[k];
```

where *tau[k]* (the parameter that determines filter gain) adapts to path conditions, as does the bandwidth sample *sample_BWE[k]*, calculated over a time interval $T[k]$. The expressions used to obtain $tau[k]$ and $T[k]$ are specified in [10].

We tested the accuracy of this filter with the same sequences of independent random interarrival times used for the results in Fig. 1. The bandwidth estimated by Westwood 3, shown in Figs. 5a and 5b, is improved with respect to Westwood 2 (see Fig. 1). However, the estimate is still biased.

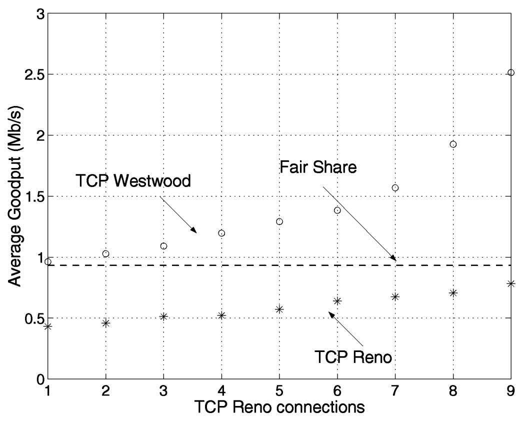


Fig. 4. Link utilization of TCP Westwood and TCP Reno sources sharing the same 10 Mb/s link.

It is important to note that, no matter how sophisticated the filter implementation, directly filtering the bandwidth samples can, in any case, result in a biased estimate, as is proven in the following.

Let L and T be the random variables representing the number of bits acknowledged by the ACK and the interarrival time between consecutive ACKs, respectively.

The algorithm directly filtering the bandwidth samples considers the random variable $Z = \frac{L}{T}$ and evaluates its expected value. Since L and T are statistically independent, $E[Z] = E[L] \cdot E[\frac{1}{T}]$. To compute $E[\frac{1}{T}]$, we expand the function $f(T) = \frac{1}{T}$ about the point $E[T]$. $E[Z]$ is then given by:

$$\begin{aligned} E[Z] &= E[L] \cdot \sum_{n=0}^{+\infty} (-1)^n \frac{E[(T - E[T])^n]}{(E[T])^{n+1}} \\ &= \frac{E[L]}{E[T]} + E[L] \cdot \sum_{n=2}^{+\infty} (-1)^n \frac{E[(T - E[T])^n]}{(E[T])^{n+1}}, \end{aligned} \tag{2}$$

where the first term, $E[L]/E[T]$, represents the average bandwidth used by the TCP source. The second term represents the bias whose entity is dominated by the

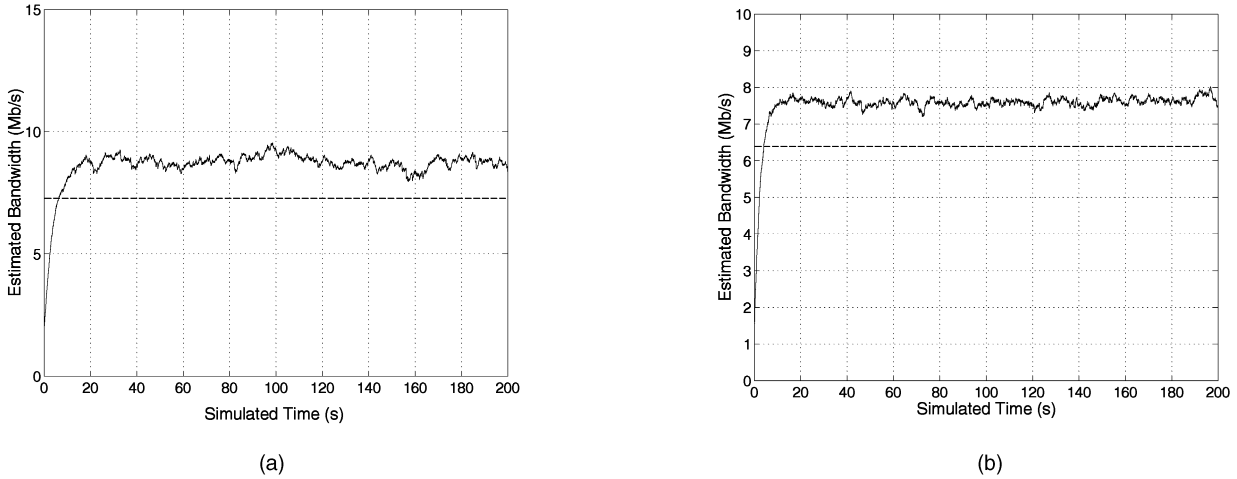


Fig. 5. Westwood 3 filter: estimated bandwidth values with constant packet length equal to 1,000 bytes and random independent interarrivals (a) exponentially distributed and (b) Rayleigh distributed. The dotted lines represent the correct bandwidth estimate.

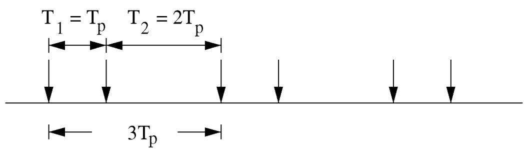


Fig. 6. ACK's arrival pattern.

variance of T, usually quite high due to ACK clustering. The region of convergence of the series (2) is [0,2E[T]].

As an example, Fig. 6 shows a simple and typical situation where each ACK acknowledges a constant number of bits, L_p, and the ACK arrivals follow a periodic pattern.

The average bandwidth used by the connection is given simply by $E[L]/E[T] = 2L_p/3T_p$, while $E[Z] = (L_p/T_1 + L_p/T_2)/2 = (L_p/T_p + L_p/2T_p)/2 = 3L_p/4T_p$, that is higher than the bandwidth really used. The bias value, $L_p/12T_p$, is very well approximated by the first term obtained for $n = 2$, equal to $2L_p/27T_p$.

2.2.4 Core Stateless Fair Queueing Estimation Algorithm

A nonlinear technique to estimate bandwidth directly from bandwidth samples was first used in the Core Stateless Fair Queueing estimation algorithm (CSFQ) [21], that was originally designed to run on IP routers.

We implemented this algorithm at the sender-side of a TCP connection and found appealing results even when performed end-to-end. It was then implemented at the TCP level, filtering the rate of returning ACKs and then setting the *ssthresh* according to (1) after congestion events.

The bandwidth estimation algorithm is described by the following equation:

$$Bwe[k] = (1 - e^{\frac{-T[k]}{K}}) * \frac{L[k]}{T[k]} + e^{\frac{-T[k]}{K}} * Bwe[k-1], \qquad (3)$$

where Bwe is the low-pass filtered estimated bandwidth, $L[k]$ is the number of bytes acknowledged by the last ACK, $T[k]$ is the last ACK interarrival, $L[k]/T[k]$ is the instantaneous rate of the ACK stream, and K is a time constant (in [21] it is recommended to choose K in the range between 0.1 and 0.5 seconds). Note that k and $k-1$ represent the actual and the previous values of the variables.

We simulated the CSFQ estimate algorithm referring to the usual scenario with 10 TCP connections sharing a 10 Mb/s channel and further assumed that a 5 Mb/s UDP flow is active in the time interval between 300 and 400 seconds. The one-way propagation delay is equal to 50 ms, and the queue is able to contain a number of packets equal to the bandwidth-delay product.

The time traces of the estimated bandwidth for $K = 0.5s$ and $K = 20s$ are shown in Fig. 7, together with the fair-share value represented by the dotted line. Although the estimate is not accurate the bias is negligible.

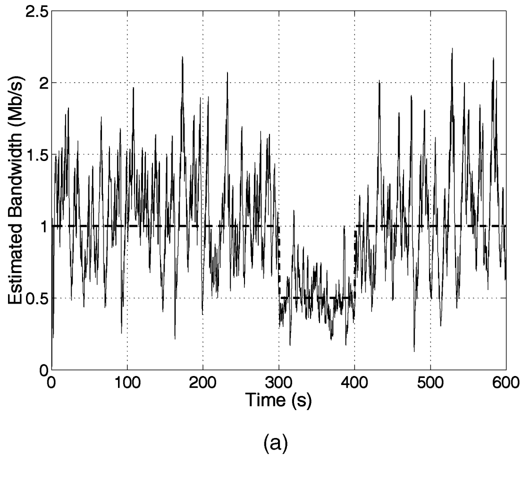
A comparison of the results in the two cases highlights a trade off between estimate stability and time responsiveness. The estimate oscillations reduce drastically as K increases, however, time responsiveness to network changes becomes weak for large values of K, e.g., for $K = 20s$, the bandwidth variation due to the UDP flow is estimated too late (Figs. 7a and 7b).

The importance of unbiased estimates for TCP congestion control has been confirmed by the results (not reported for the sake of brevity) of mixed scenarios with TCP Reno, where there was better fairness than TCP Westwood even with small values of K.

3 TIBET

In this section, we present TIBET (Time Intervals based Bandwidth Estimation Technique) [5], a new technique that correctly estimates the bandwidth used by the TCP source, even in the presence of packet clustering and ACK compression. TIBET also enables the TCP connections to track changes in the available bandwidth quickly.

To explain the rationale of TIBET, let us refer to the example in Fig. 8, where transmissions occurring in a period T are considered. Let n be the number of packets belonging to a connection and $L_1, L_2 \dots L_n$ the lengths, in bits, of these packets. The average bandwidth, Bw, used by the connection is given by



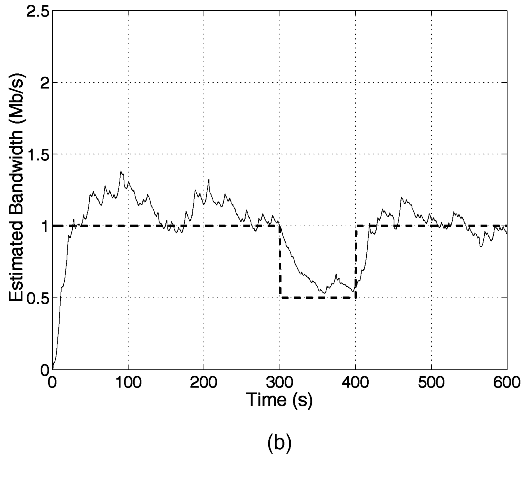


Fig. 7. CSFQ bandwidth estimation with (a) K = 0.5 seconds and (b) K = 20 seconds.

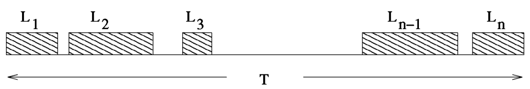


Fig. 8. Pattern of packet transmission.

$$Bw = \frac{1}{T}\sum_{i=1}^{n} L_i = \frac{n\overline{L}}{T} = \frac{\overline{L}}{\frac{T}{n}}. \tag{4}$$

The estimate of the average bandwidth can be obtained by performing a runtime sender-side estimate of the average packet length, $\overline{L}$, and the average interdeparture time, $\frac{T}{n}$, separately. As shown when studying Westwood, this approach is not affected by any bias. Moreover, since intervals greater than one Round Trip Time are used to estimate the average interdeparture time, TIBET is not affected by interdeparture times T close to zero as usually happens when TCP sources transmit a group of packets. Note that algorithms like TCP Westwood use bandwidth samples and are therefore affected by such short interdeparture times.

3.1 Estimation Scheme

The bandwidth estimation scheme can be applied either to the stream of transmitted packets or the stream of received ACKs. The pseudocode of the algorithm applied to the stream of transmitted packets is:

```
if (Packet is sent)
sample_length[k] = (packet_size*8);
sample_interval[k] = now - last_sending_time;
Average_packet_length[k] =
      alpha*Average_packet_length[k-1]
      + (1-alpha)*sample_length[k];
Average_interval[k] = alpha*
      Average_interval[k-1]
      + (1-alpha )*sample_interval[k];
Bwe[k] = Average_packet_length[k]/
          Average_interval[k]
endif
```

where *packet_size* is the segment size in bytes, *now* is the current time, *last_sending_time* is the time of the previous packet transmission, and *k* and *k-1* indicate the current and previous values of the variables. *Average_packet_length* and *average_interval* are the low-pass filtered measures of the packet length and the interdeparture times, respectively. *Alpha* ($0 \leq alpha \leq 1$) is the pole of the two low-pass filters. Bwe is the estimated value of the used bandwidth. The value of *alpha* has a critical impact on TIBET performance: If $alpha$ is set to a low value, TIBET is highly responsive to changes in the available bandwidth, but the oscillations of $Bwe[k]$ are quite large. On the contrary, if $alpha$ approaches 1, TIBET produces more stable estimates, and is less responsive to network changes. After having tested TIBET on several network scenarios, we reached the conclusion that $alpha$ equal to 0.99 provides a good compromise between responsiveness and stability.

The algorithm applied to the stream of received ACKs differs from the one above only in the expressions used to calculate *sample_length* and *sample_interval*:

```
sample_length[k] = (acked * packet_size * 8);
sample_interval[k] = now - last_ack_time;
```

where *last_ack_time* is the time when the last ACK was received, and *acked* is the number of segments acked by the ACK.

When congestion occurs, $cwnd$ and $ssthresh$ are updated according to (1), the bandwidth first being estimated by one of the two above procedures. The overall procedure is specified by the following pseudocode:

```
if (3 duplicate ACKs are received)
        ssthresh = Bwe * RTT_min
        if (cwnd > ssthresh)
        cwnd = ssthresh
        end if
end if

if (retransmission timeout expires)
        ssthresh = Bwe * RTT_min
        cwnd = 1
end if
```

All the results reported in this paper were obtained by applying the bandwidth estimation scheme to the stream of transmitted packets.

3.2 Estimation Accuracy and Fairness

The bandwidth estimated by TIBET was measured in a simulation scenario: 10 TCP connections over a 5 Mbit/s link and a drop-tail managed bottleneck queue that could contain a number of packets equal to the bandwidth-delay product. The measured time trace of the estimated bandwidth is shown in Fig. 9a.

Although the average value is close to the fair share, equal to 500 kbit/s, the oscillations are deep, and to smooth them, and ensure an estimate closer to the right value, further Bwe sample filtering is proposed:

$$\begin{aligned} Bwe[k] = &\left(1 - e^{\frac{-T[k]}{T_0}}\right) * \frac{Average\_packet\_length[k]}{Average\_interval[k]} \\ &+ e^{\frac{-T[k]}{T_0}} * Bwe[k-1], \end{aligned} \tag{5}$$

where $T[k]$ is the time interval between the last two estimates and T_0 is a time constant ($T_0 = 1s$ in our simulations). Binding the value of the pole to $T[k]$, we perform adaptive filtering: this entails exploiting the oscillations of the Bwe signal in such a way as to follow the variations in bandwidth quickly. This filter is basically the same as that proposed for the CSFQ scheme (see (3)).

With this adaptive filter, the estimate is smoothed, practically overlapping the fair share curve as shown in Fig. 9b.

In order to show the accuracy of the TIBET estimation scheme and compare it with TCP Westwood, we considered the same sequences of independent random interarrival times as used in Fig. 1. The bandwidth estimated by TIBET, Figs. 10a and 10b, is very close to the correct value, shown by the straight dotted line. The improvement with respect to Westwood (see Figs. 1 and 5) is remarkable.

We also tested TIBET behavior in a simulated scenario where a single TIBET source performs a file transfer over a

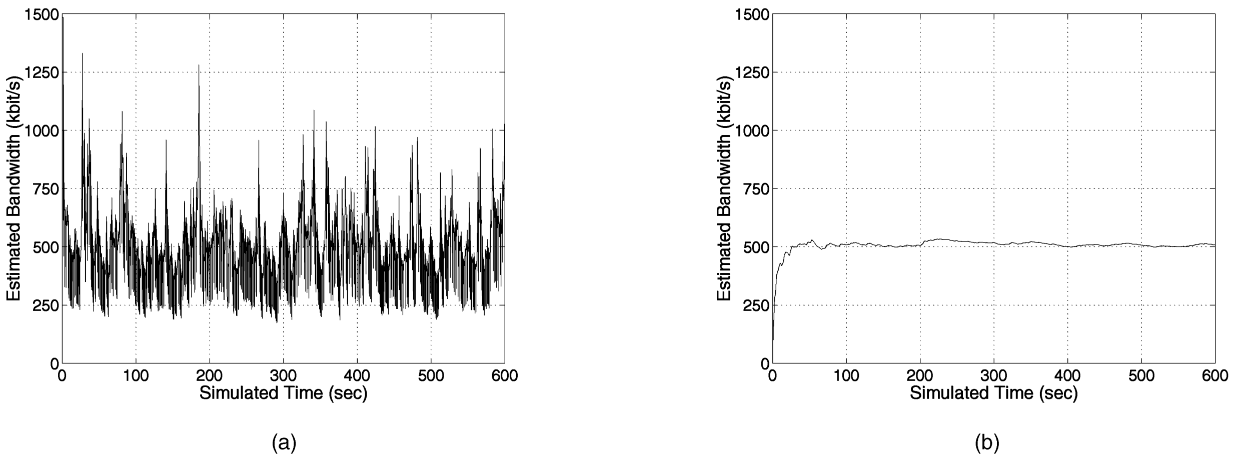


Fig. 9. TIBET bandwidth estimate (a) without adaptive filtering and (b) with adaptive filtering.

10 Mb/s link, with a RTT of 100 ms, sharing the link with two UDP ON/OFF sources. The dotted line in Fig. 11a shows the bandwidth not used at that time by UDP sources, while the bandwidth estimated by TIBET is represented by the solid line. The two curves almost overlap, proving the correctness of the TIBET estimation algorithm and its ability to follow step variations in the available bandwidth.

We have also considered a scenario with a single TIBET connection transmitting over a link with capacity equal to 10 Mb/s. Two hundred seconds after the beginning of the transmission, a UDP source starts transmitting with a constant bit rate equal to the 95 percent of the link capacity, thus causing a sudden surge in link traffic. We considered the behavior of a TIBET source with the bandwidth estimation scheme applied either to the stream of transmitted packets or to the stream of received ACKs. Fig. 11b shows the sending rate and the receiving rate estimate produced by these two different TIBET implementations in the interval between 190 and 210 seconds. The two rates differ slightly only for few seconds just after the beginning of the UDP connection. This is due to the window flow control strategy of TCP sources that reduce their transmission rate when acknowledgements return slowly. Therefore, if there is congestion along the path between the TCP sender and receiver, the large delays experienced by the ACKs will cause a natural slowdown of the transmitter data rate to the actual receive rate.

To better characterize the behavior of the two different TIBET implementations in this scenario, we have also measured the number of segments transmitted by each of them in the same interval around $t = 200s$. The TIBET connection estimating the sending rate transmitted a total of 8,592 segments, and the one estimating the receive rate transmitted only two segments less. A similar behavior has been observed when considering the same network scenario, with a single TIBET source and 20 TCP Reno connections that start transmitting 200 seconds after the TIBET connection.

A further advantage of TIBET is that its bandwidth estimate is not affected by the ACK compression effect. To prove this, let us consider a scenario with a congested return path the same as that adopted in Fig. 3 for TCP Westwood; we considered a 2 Mbit/s link with two TCP connections sending data packets in the two directions. In the time interval between 40 and 140 seconds both the TCP

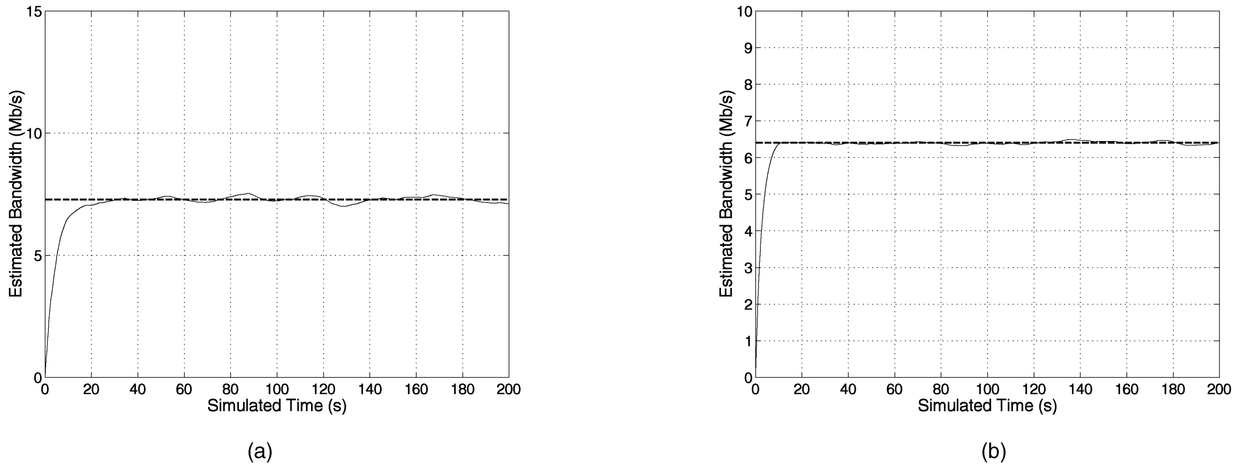
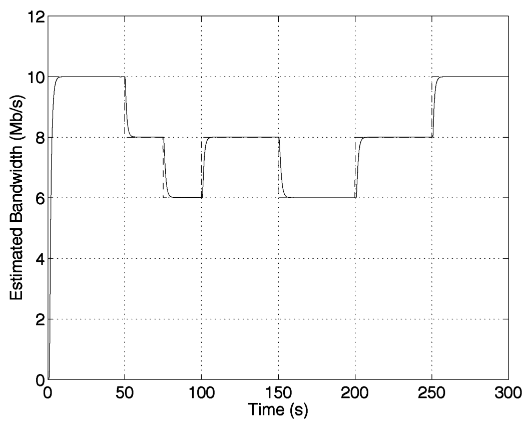
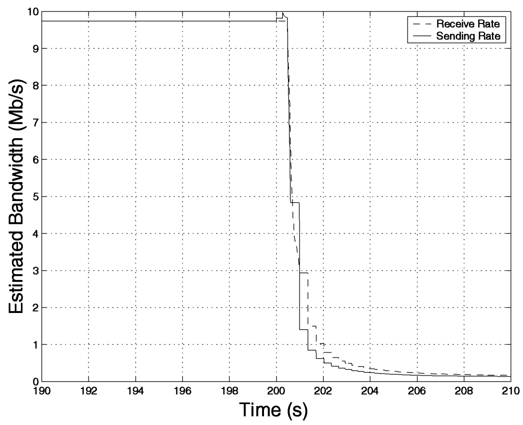


Fig. 10. TIBET filter: estimated bandwidth values with constant packet length equal to 1,000 bytes and random independent interarrivals (a) exponentially distributed and (b) Rayleigh distributed. The dotted lines represent the correct bandwidth estimate.



(a)



(b)

Fig. 11. (a) TIBET bandwidth estimate with concurrent UDP traffic with variable rate. (b) Comparison between sending and receive rate over a 10 Mb/s link with a sudden surge of UDP traffic.

connections transmit packets, and ACK compression is observed. The bandwidth estimate shown in Fig. 12 is not at all affected by the ACK compression, proving the much higher robustness of TIBET, compared to TCP Westwood (see Fig. 3).

In our implementation, TCP sources use the estimated bandwidth only after congestion events. Such a choice is supported by the poor performance that is evident when there is frequent updating of *ssthresh* (the numerical results are not reported for the sake of brevity). The main reason for this is that frequent *ssthresh* updating tends to force the TCP source into congestion avoidance, preventing it from following the variations in the available bandwidth.

So far, we have shown that TIBET can actually achieve an accurate estimate of the used bandwidth. Now, a check must be made of the TCP sources performance using TIBET in mixed scenarios where the sources use different TCPs. For this purpose, we simulated the same scenario as used for Westwood to obtain the results of Fig. 4, where TIBET sources substitute TCP Westwood sources. The average goodputs of TIBET and TCP Reno connections are shown in Fig. 13a. The goodput achieved by both algorithms is very close to the fair share for the full range of sources. A goodput very close to the fair share has been obtained in the same scenario when TCP Reno connections are activated in a link already loaded by TIBET connections.

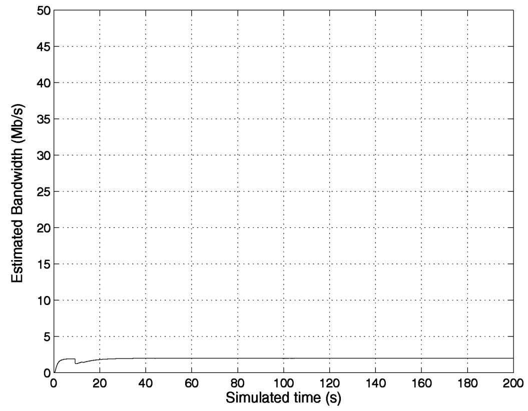
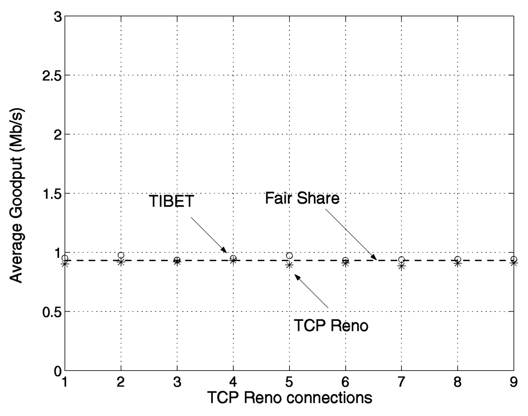


Fig. 12. Bandwidth estimated by a single TIBET source over a 2 Mb/s link in the presence of ACK compression.

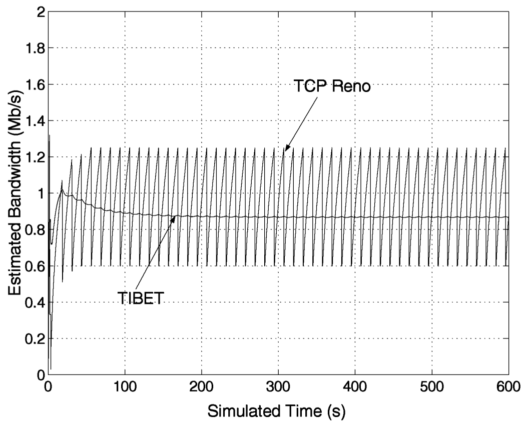
We also extended our simulation campaign to similar scenarios covering link bandwidths ranging from a few kbit/s up to 150 Mbit/s, and with a varying number of competing connections. The results obtained confirm that TIBET achieves the same level of fairness as TCP Reno. We further observed that TIBET improves its performance when the number of connections sharing the bottleneck link increases, since the estimate variance reduces. Moreover, the presence of constant rate flows, such as UDP flows for IP telephony or video conference, causes TIBET to perform better since packet cluster size is reduced.

From the simulation results related to the scenarios considered, we can claim that TIBET shows a quite fair behavior toward TCP Reno, even if a mathematical proof is not yet available. However, to further support our claim, we have considered a simple scenario with a 2 Mbit/s link with round trip time equal to 100 ms shared by a TCP Reno and a TIBET connection. In Fig. 13b, we report the values of the TCP Reno instantaneous bandwidth estimate, obtained by dividing the congestion window value (in bits) by the current round trip time, and the TIBET bandwidth estimate. It turns out that TIBET estimate is very close to the average value of Reno. So, even if TIBET does not simply halve the window at congestion events, on average, its behavior is equivalent to that of Reno over error free links. As already mentioned, the only advantage of TIBET is that of adding memory to the bandwidth estimate, while keeping the same average value.

The effectiveness of TIBET in estimating RTT_{min} in presence of persistently congested links has been tested by considering a 10Mb/s channel with 20 active long-lived TCP Reno connections. In this steady-state condition, a single TIBET connection becomes active. The estimate of RTT_{min}, whose real value is 100 ms, starts from 200 ms, reduces to 130 ms after 1.5 seconds and reaches 105 ms in 80 seconds. In general, the estimate of RTT_{min} will converge to the real value if the queues along the path have a stationary behavior and, therefore, their busy periods have a finite length. However, the convergence rate can be very slow, especially when the network is persistently congested



(a)



(b)

Fig. 13. (a) TIBET fairness toward TCP Reno over a 10 Mbit/s link. (b) Comparison between TCP Reno current rate and TIBET bandwidth estimate over a 2 Mbit/s link.

by uncontrolled UDP flows. In the presence of TCP flows only, we expect that the estimate converges to the real value in a reasonable time as measured in our simulations.

As for every other algorithm using explicit bandwidth estimate for TCP congestion control, the coarse-grained clock problem described in Section 2 affects TIBET performance. To reduce such effects we propose, in TIBET, the following modified update of RTT_{min}, to be used when RTT_{min} is smaller than the clock granularity:

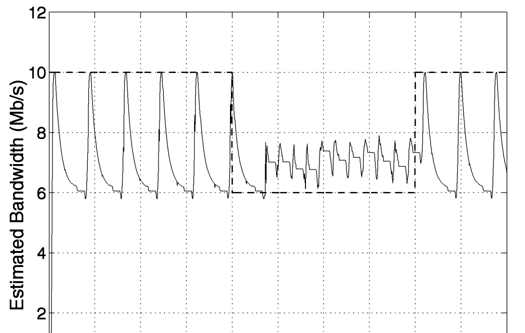
```
if (the connection experiences a congestion event)
    $n = n + 1$
    if ($n = N_{cong}$)
        $RTT_{min} = \gamma \cdot RTT_{min}$
        $n = 0$
    end if
end if
```

where N_{cong} is the congestion events threshold value and γ ($0 \leq \gamma \leq 1$) represents the reducing factor. To evaluate the effectiveness of the RTT_{min} updating algorithm, we considered the following scenario.

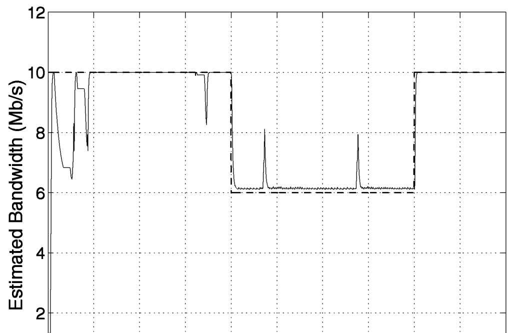
A single TCP source running the TIBET algorithm transmits over a 10 Mb/s link, with a RTT of 50 ms. In the 40 to 80 second time interval an UDP flow with the same priority as the TCP source transmits at a data rate of 4 Mb/s. The clock granularity is 500 ms, i.e., 10 times greater than the actual RTT of the connection, therefore also the RTT_{min} of the TCP source is set at 500 ms. The estimated bandwidth obtained by simulation for both versions of TIBET, without and with the updating are respectively shown in Figs. 14a and 14b. In both figures, the dotted line represents the actual bandwidth.

Without the algorithm (Fig. 14a), the connection is very aggressive and the link is often congested. With the algorithm, and assuming $N_{cong} = 5$, $\gamma = 0.5$, the bandwidth estimate is more accurate, and link congestion is more effectively controlled (Fig. 14b): the estimate, except for rare peaks, overlaps the actual value. It can be seen that N_{cong} and γ are not critical, as their values affect only the time responsiveness of the algorithm, not the TCP source steady state behavior.

The improvement, due to the better estimate, was evident in the average goodput achieved by the TCP

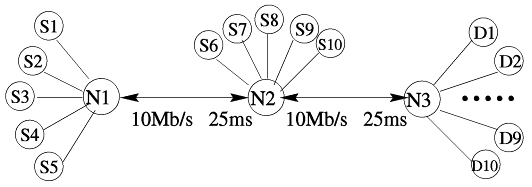


(a)



(b)

Fig. 14. Bandwidth estimate executed (a) in absence and (b) in the presence of the RTT_{min} updating algorithm.



(a)

	Reno	Sack	TIBET
S1–S5	490	520	575
S6–S10	1430	1400	1325

(b)

Fig. 15. (a) Network topology with differential Round Trip Times across the connections and (b) goodput achieved (kbit/s) by TCP Reno, TCP Sack, and TIBET in this topology.

connection. When no RTT_{min} updating is used, the goodput equals only 1.2Mb/s, but it increases to 5.8Mb/s when the updating algorithm is implemented.

It is worth noting that the proposed algorithm was not designed specifically for TIBET, it can be adopted to improve the performance of any TCP version exploiting bandwidth estimation algorithms.

Finally, we have measured the performance of various TCP algorithms when applied to connections with different RTTs. The considered network scenario, Fig. 15a, includes 10 TCP sources, $S1 - S5$ connected at node $N1$ and $S6 - S10$ connected at node $N2$, that transmit to the destinations $D1 - D10$, all connected at node $N3$, through 10Mb/s links having a one-way propagation delay equal to 25 ms. The RTT of the sources $S1 - S5$ is equal to 100 ms, while the RTT of sources $S6 - S10$ is equal to 50 ms.

The average goodputs, expressed in kbit/s, achieved by the connections of nodes $N1$ and $N2$ when using TCP Reno, TIBET, and TCP Sack are shown in Fig. 15b. TIBET, similarly to Westwood studied in [22], achieves better fairness between connections having different round trip times.

4 Wireless Links

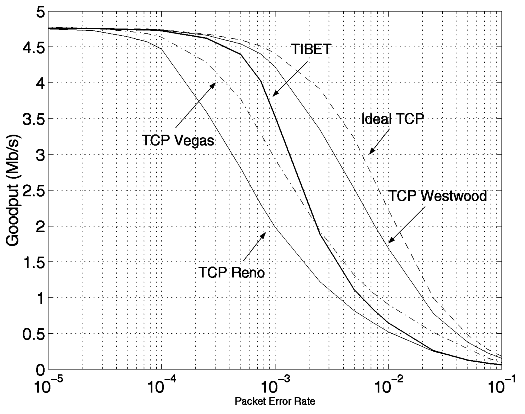
So far, this paper has looked at the accuracy of bandwidth estimation algorithms and the performance of TCP sources over error-free links. However, as such algorithms are mainly designed to achieve high throughput in the presence of links affected by random errors, a study was made of the performance of these algorithms over wireless links.

In order to measure TIBET performance, and compare it with other TCP versions, we considered several scenarios with two different types of connection: the long-lived TCP connections, typical of FTP file transfers, and short-lived connection, typical of HTTP connections. In the following, we present and discuss the results obtained by simulation.

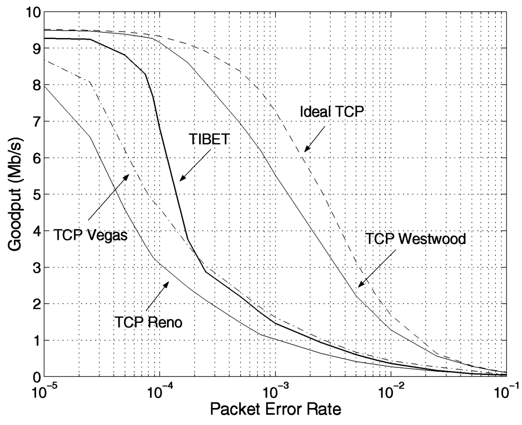
4.1 Long-Lived TCP Connections

For long-lived TCP connections performing FTP transfers, we considered four link scenarios with 5 and 10 Mbit/s capacity. The Round Trip Time is equal to 100 ms and the queue can contain a number of packets equal to the bandwidth-delay product. Independent errors occur at random, causing a packet error rate in the 10^{-5} to 10^{-1} range. For each channel, we measured the steady state goodput obtained by TCP Reno, TCP Vegas, TIBET, and TCP Westwood. The results are shown in Fig. 16.

It can be seen that, for all packet error rates and at all link speeds, TIBET achieves higher goodput than TCP Reno. This is due to the filtering process that takes the past history of the connection into account, preventing, most of the time, confusion between real network congestion signals due to queue overflow and signals due to link errors. In order to provide a more complete comparison, we also analyzed the performance achieved by TCP NewReno [23] and TCP Sack [24]. Their goodputs are not shown since, in the range of considered packet losses, they practically overlap the

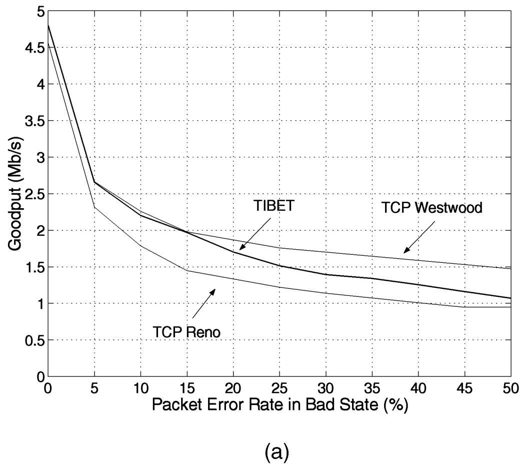


(a)



(b)

Fig. 16. Various TCP implementations' goodput versus packet error rate over a (a) 5 Mbit/s link and (b) 10 Mbit/s link.



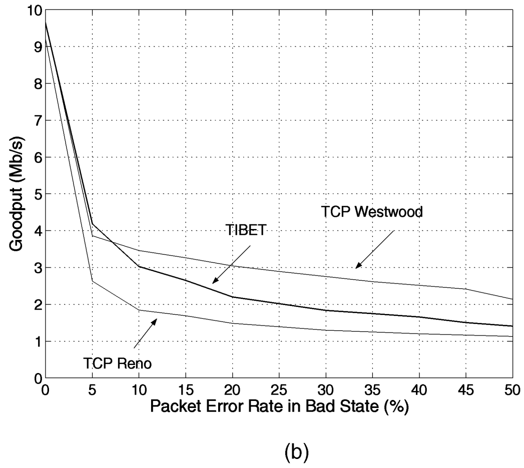


Fig. 17. TIBET, TCP Westwood, and TCP Reno goodput achieved over a link with capacity (a) 5 Mb/s and (b) 10 Mb/s, affected by correlated losses, as a function of the packet error rate in the Bad state.

goodput of TCP Reno, in agreement with what was pointed out in [25], [26].

The TIBET and TCP Vegas goodputs are very close: for small packet error rates, TIBET achieves the higher goodput, however, also, TCP Vegas shows almost the same performance for high link capacities and high packet error rates.

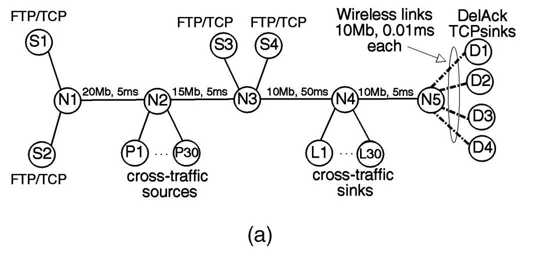
To account for the effects of multipath fading typical of wireless environments, we also investigated the behavior of TIBET and TCP Reno in the presence of links affected by correlated errors.

From the existing literature [27], we modeled the wireless link state (*Good* or *Bad*) with a two-state Markov chain. We considered two different scenarios with wireless link capacities equal to 5 and 10 Mb/s, a Round Trip Time equal to 100 ms, and an average duration of good and bad states equal to 1 and 0.05 seconds, respectively. In the good state, no packet loss occurs, while, in the bad state, the packet error rate varies from 0 to 50 percent to take into account various levels of fading. It can be seen from Fig. 17 that, also in this case, TIBET obtains a goodput higher than TCP Reno.

In all the considered scenarios, TCP Westwood obtained a higher goodput than any other TCP version. This is due to its overestimate of the available bandwidth that has the drawback of leading to aggressive behavior and unfair sharing of network resources, with respect to TCP Reno in wired links, as previously shown in Section 2.

4.2 Mixed Wired and Wireless Networks

To measure TIBET's performance in a more realistic scenario, we have considered the mixed wired/wireless network shown in Fig. 18a with four TCP connections traversing multiple wired links as well as a wireless link affected by independent random transmission errors. A cross-traffic, generated by 30 UDP connections between nodes $N2$ and $N4$, shares the bottleneck channel between $N3$ and $N4$ with the TCP traffic. Each UDP source switches between ON and OFF periods, whose durations are Pareto distributed with shape parameter equal to 1.5 and mean durations equal to 100 ms and 200 ms, respectively. During



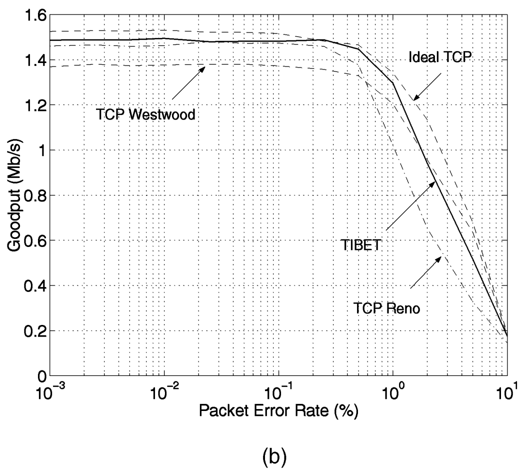
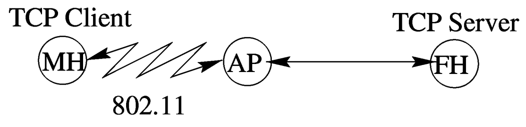
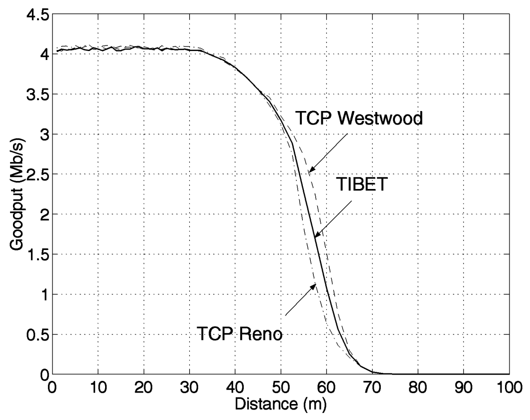


Fig. 18. (a) Mixed wired-wireless multihop network topology. (b) Goodput achieved by TIBET, TCP Westwook, TCP Reno, and Ideal TCP as a function of the packet error rate.



(a)



(b)

Fig. 19. (a) Mixed wired-wireless 802.11 network topology. (b) Goodput achieved by TIBET, TCP Westwood, and TCP Reno in this topology.

the ON period, each source transmits packets with 1,500 byte size at constant bit rate equal to 0.5 Mbit/s; while in OFF period, the UDP sources transmit no packet.

Fig. 18b shows the goodput achieved by the TCP connection $S1 - D1$ versus packet error rates and for different TCP versions. We observe that, even in this more complex scenario, TIBET achieves a higher goodput than TCP Reno, especially when the packet error rate increases.

4.3 Effect of Link Layer ARQ: The 802.11 Case

The scenario shown in Fig. 19a has been used to test TIBET performance over a link layer exploiting ARQ, with a mobile host (MH) transmitting to a fixed host (FH) through an access point (AP) using the Wireless LAN 802.11 protocol.

The wireless channel includes a power loss due to shadowing, modeled as a log-normal random variable with standard deviation equal to 4 dB and a path loss with distance exponent equal to 3. According to the 802.11 standard, the link rate is 11 Mb/s, the channel probing phase RTS/CTS (Request To Send/Clear To Send) is enabled, and the maximum number of retransmission of the ARQ level is set to 4. All other parameters of the mobile host and the access point were configured according to default values as proposed in the Monarch extensions to NS [28]. In this scenario, the effect of the wireless channel is mitigated by the ARQ mechanism that, however, introduces some delay jitter and cannot recover all losses due to the limited number of consecutive retransmissions.

The goodputs achieved by different TCP sources as function of the distance between the mobile host and the access point, see Fig. 19b, show a slight improvement of TIBET over TCP Reno.

4.4 Short-Lived TCP Connections

To study short-lived TCP connections, we considered, in line with the literature [29], a typical HTTP connection involving the transfer of a 10 kbyte file over a 5 Mb/s link affected by a 5 percent random packet loss, with a 100 ms Round Trip Time. We simulated several transfers and measured the duration of each file transfer.

The average time to complete the transfer was $3.85s$ for TIBET and $3.9s$ for TCP Reno, NewReno and Sack. Hence, for short file transfers, TIBET achieves the same results as the current TCP versions.

4.5 Throughput Upper Bound

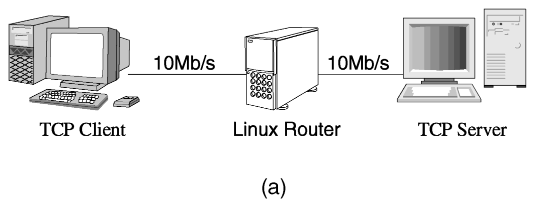
Having proved the advantage of TIBET over Reno, let us now address the issue of just how far the performance of TIBET is from the upper bound of all possible schemes exploiting the new bandwidth estimation approach. Such an upper bound is obtained by assuming an *Ideal TCP* that sets *cwnd* and *ssthresh* through knowledge of the exact bandwidth available along the path ($Exact\_Available\_Bw$) and the exact round trip time ($Exact\_RTT$) of the connection. The exact available bandwidth is equal to the link bandwidth, minus the transmission rate of UDP sources, divided by the total number of TCP connections sharing the link. This TCP source is ideal, as real bandwidth estimation algorithms implemented at TCP sources can only measure the *used bandwidth*, i.e., the transmission rate of the connection, that can be very different from the bandwidth available to the connection, especially over links affected by random losses.

Ideal TCP reacts to congestion signals by changing $cwnd$ and $ssthresh$ according to the TIBET algorithm described in Section 3:

```
if (3 duplicate ACKs are received)
        ssthresh = Exact_Available_Bw * Exact_RTT
        if (cwnd > ssthresh)
        cwnd = ssthresh
        end if
end if

if (retransmission timeout expires)
        ssthresh = Exact_Available_Bw * Exact_RTT
        cwnd = 1
end if
```

The throughput achieved by this ideal scheme was obtained by simulation, and is shown by the dotted curves in Figs. 16 and 18b. The high goodput degradation measured, even with the Ideal TCP, at high error rates proves that such



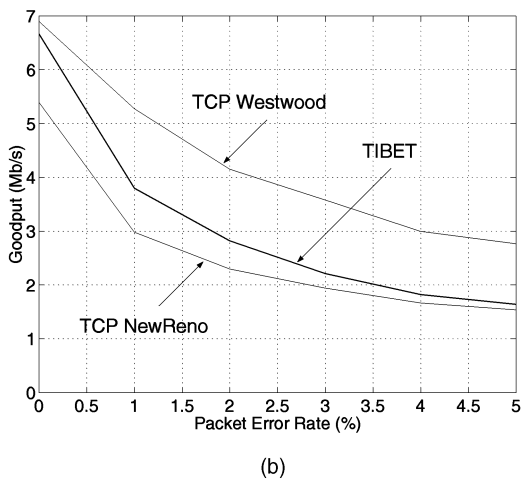


Fig. 20. (a) Test bed topology and (b) goodput achieved by TIBET, TCP Westwood, and TCP NewReno in the Test Bed.

losses are unavoidable and do not depend on the estimation algorithm used.

Note that, for high error rates, the goodput achieved is practically independent of channel bandwidth.

4.6 Implementation and Test Bed

To get more details on the TIBET implementation, we have built a test bed, shown in Fig. 20a, that consists of a PC server, a client, and a PC router, all connected by 10 Mb/s LAN cables. In the PC server, besides the TCP NewReno that is the current TCP implementation in the Linux kernel version 2.2-20, we have implemented both TIBET and TCP Westwood. The PC router emulates a wireless link with the desired delay and packet loss rate using the NIST Net software [30].

Running the test bed, we measured the goodputs achieved by the three TCP versions. Fig. 20b compares the steady-state goodput achieved by TIBET, TCP Westwood, and TCP NewReno connections transmitting data between the server and the client, with an emulated round trip time equal to 100 ms versus packet loss rates.

The measures on this real scenario validate the results obtained by simulation (see Fig. 16b) and provide a further support on the advantages of TIBET over TCP NewReno.

Note that, in this scenario, as well as in all the simulated scenarios presented in this section, TCP Westwood obtained a higher goodput than any other TCP version. Again, this behavior is due to its overestimate of the available bandwidth, that leads to aggressive behavior and unfair sharing of network resources.

5 Conclusions

In this paper, we have discussed and analyzed issues related to the use of enhanced bandwidth estimation algorithms for TCP congestion control. These algorithms, differently from that used in TCP Reno, add memory, considering the past history of the connection when a congestion event occurs. Such algorithms have the potential to improve the TCP throughput over wireless links as the available memory enables a lessening of the impact of channel loss.

However, for smooth adoption into the Internet, these new versions of TCP must achieve fair behavior when used with TCP Reno over wired links. We have discovered that a key to achieving a fair behavior is to base the TCP operation on an unbiased and accurate bandwidth estimate.

Thus, for real network scenarios, we studied the problems any algorithm must face to obtain an accurate estimate, and evaluated the performance of the schemes proposed in the literature.

Regrettably, we found that such schemes are often unable to give accurate estimates, and that, over wired links, this lack of accuracy leads the TCP sources to an unfair resource sharing with TCP Reno. To overcome this problem, we now propose a new algorithm, TIBET, that enables TCP to achieve both a higher throughput over wireless links and fair behavior over wired links. The performance of this new algorithm has been compared with that of other existing TCP, and it has been found that significant improvement is obtained using TIBET.

We also defined an ideal scheme that assumes the exact value of the bandwidth, and that provides, for all possible schemes based on the estimation approach, an upper bound to the throughput.

Although there is still room for improvement in TCP performance, the above mentioned bound shows that the throughput degradation in noisy channels is unavoidable and that the main factor limiting performance is random packet loss.

Acknowledgments

This work was partially supported by MIUR in the framework of FIRB VICOM and TANGO projects.

References

[1] M. Mathis, J. Semke, and J. Mahdavi, "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm," *ACM Computer Comm. Rev.*, vol. 27, no. 3, 1997.

[2] T.V. Lakshman and U. Madhow, "The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random Loss," *IEEE/ACM Trans. Networking*, vol. 5, no. 3, pp. 336-350, 1997.

[3] H. Balakrishnan, V.N. Padmanabhan, S. Seshan, and R.H. Katz, "A Comparison of Mechanisms for Improving TCP Performance over Wireless Links," *CIEEE/ACM Trans. Networking,* vol. 5, no. 6, pp. 759-769, Dec. 1997.
[4] S. Mascolo, M.Y. Sanadidi, C. Casetti, M. Gerla, and R. Wang, "TCP Westwood: End-to-End Congestion Control for Wired/Wireless Networks," *Wireless Networks J.,* vol. 8, pp. 467-479, 2002.
[5] A. Capone and F. Martignon, "Bandwidth Estimates in the TCP Congestion Control Scheme," *Proc. Tyrrhenian Int'l Workshop Digital Comm.,* Sept. 2001.
[6] S. Keshav, "A Control-Theoretic Approach to Flow Control," *Proc. ACM SIGCOMM,* pp. 3-15, Sept. 1991.
[7] J.C. Hoe, "Improving the Start-Up Behavior of a Congestion Control Scheme for TCP," *ACM SIGCOMM Computer Comm. Rev.,* vol. 26, no. 4, pp. 270-280, Oct. 1996.
[8] M. Allman and V. Paxson, "On Estimating End-to-End Network Path Properties," *Proc. ACM SIGCOMM,* 1999.
[9] L.S. Brakmo and L.L. Peterson, "TCP Vegas: End-to-End Congestion Avoidance on a Global Internet," *IEEE J. Selected Areas in Comm.,* vol. 13, no. 8, pp. 1465-1480, Oct. 1995.
[10] R. Wang, M. Valla, M.Y. Sanadidi, and M. Gerla, "Adaptive Bandwidth Share Estimation in TCP Westwood," *Proc. Globecom,* 2002.
[11] J.C. Mogul, "Observing TCP Dynamics in Real Networks," *Proc. ACM SIGCOMM Symp. Comm. Architectures and Protocols,* pp. 305-317, 1992.
[12] L. Zhang, S. Shenker, and D. Clark, "Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic," *Proc. SIGCOMM Symp. Comm. Architectures and Protocols,* pp. 133-147, Sept. 1991.
[13] V. Paxson and M. Allman, "Computing TCP's Retransmission Timer," RFC 2988, Nov. 2000.
[14] J. Mo, V. Anantharam, R.J. La, and J. Walrand, "Analysis and Comparison of TCP Reno and Vegas," *Proc. ACM GLOBECOM,* 1999.
[15] C. Dovrolis, P. Ramanathan, and D. Moore, "What do Packet Dispersion Techniques Measure?" *Proc. Infocom,* vol. 2, pp. 905-914, 2001.
[16] C. Dovrolis and M. Jain, "End-to-End Available Bandwidth: Measurement Methodology, Dynamics, and Relation with TCP Throughput," *Proc. ACM SIGCOMM,* Aug. 2002.
[17] S. Mascolo, C. Casetti, M. Gerla, and S.S. Lee, M. Sanadidi, "TCP Westwood: Congestion Control with Faster Recovery," UCLA CS Technical Report #200017, 2000.
[18] R. Wang, M. Valla, M.Y. Sanadidi, B. Ng, and M. Gerla, "Efficiency/Friendliness Tradeoffs in TCP Westwood," *Proc. IEEE Symp. Computers and Comm.,* July 2002.
[19] ns-2 network simulator (ver. 2), LBL, http://www.isi.edu/nsnam, 2004.
[20] M. Allman, V. Paxson, and W. Stevens, "TCP Congestion Control," RFC 2581, Apr. 1999.
[21] I. Stoica, S. Shenker, and H. Zhang, "Core-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocations in High Speed Networks," *Proc. ACM SIGCOMM,* Sept. 1998.
[22] L.A. Grieco, S. Mascolo, and R. Ferorelli, "Additive-Increase/Adaptive-Decrease Congestion Control: A Mathematical Model and Its Experimental Validation," *Proc. IEEE Int'l Symp. Computer and Comm.,* July 2002.
[23] S. Floyd and T.R. Henderson, "The NewReno Modifications to TCP's Fast Recovery Algorithm," IETF RFC 2582, vol. 26, no. 4, Apr. 1999.
[24] S. Floyd, M. Mathis, J. Mahdavi, and A. Romanow, "TCP Selective Acknowledgement Option," RFC 2018, Apr. 1996.
[25] M. Zorzi, A. Chockalingam, and R.R. Rao, "Throughput Analysis of TCP on Channels with Memory," *IEEE J. Selected Areas in Comm.,* vol. 18, pp. 1289-1300, July 2000.
[26] A. Chockalingam and M. Zorzi, "Wireless TCP Performance with Link Layer FEC/ARQ," *Proc. IEEE Int'l Conf. Comm.,* vol. 2, pp. 1212-1216, June 1999.
[27] A.A. Abouzeid, S. Roy, and M. Azizoglu, "Stochastic Modeling of TCP over Lossy Links," *Proc. INFOCOM,* Mar. 2000.
[28] Monarch Project, http://www.monarch.cs.rice.edu/, 2004.
[29] N. Cardwell, S. Savage, and T. Anderson, "Modeling TCP Latency," *Proc. INFOCOM,* pp. 1742-1751, 2000.
[30] NIST Net Home Page, http://snad.ncsl.nist.gov/itg/nistnet/, 2004.

Antonio Capone received the Laurea degree (MS degree equivalent) and the PhD degree in telecommunication engineering from the Politecnico di Milano in July 1994 and June 1998, respectively. In 2000, he was a visiting scientist at the University of California, Los Angeles. He is now an assistant professor in the Department of Electronics and Information at the Politecnico di Milano. His current research activities include packet access in wireless cellular network, routing and MAC for multihop wireless networks, congestion control and QoS issues of IP networks, network planning, and optimization. He is a member of the IEEE and the IEEE Communications and Vehicular Technology Societies.

Luigi Fratta received the doctorate degree in electronics engineering from the Politecnico di Milano, Italy, in 1966. From 1967 to 1970, he worked at the Laboratory of Electrical Communications, Politecnico di Milano. As a research assistant in the Computer Science Department, University of California, Los Angeles (UCLA), he participated in data network design under the ARPA project from 1970 to 1971. From November 1975 to September 1976, he was at the Computer Science Department of IBM Thomas J. Watson Research Center, Yorktown Heights, New York, working on modeling analysis and optimization techniques for teleprocessing systems. In 1979, he was a visiting associate professor in the Department of Computer Science at the University of Hawaii. In the summer of 1981, he was at the Computer Science Department, IBM Research Center, San Jose, California, working on local area networks. During the summers of 1983, 1989, and 1992, he was with the Research in Distributed Processing Group, Department of Computer Science, UCLA, working on fiber optic local area networks. During the summer of 1986, he was with Bell Communication Research working on metropolitan area networks. In 1994, he was a visiting scientist at NEC Network Research Laboratory, Japan. Since 1980, he has been a full professor in the Dipartimento di Elettronica e Informazione at the Politecnico di Milano. His current research interests include computer communication networks, packet switching networks, multiple access systems, modeling and performance evaluation of communication systems, local area networks, wireless cellular systems, and integrated services over IP networks. Dr. Fratta is a fellow of IEEE.

Fabio Martignon received the Laurea degree in telecommunication engineering from the Politecnico di Milano in October 2001. He is now a PhD student in the Department of Electronics and Information at the Politecnico di Milano. His current research activities include congestion control and QoS routing over IP networks. He is a student member of IEEE.