Politecnico di Milano
Dipartimento di Elettronica e Informazione
Dottorato di Ricerca in Ingegneria dell'Informazione

Enhanced Bandwidth Estimation and Loss Differentiation in the TCP Congestion Control Scheme

Advisor: Prof. **Antonio CAPONE**
Faculty Advisor: Prof. **Luigi FRATTA**

Author:
Fabio MARTIGNON

XVII Ciclo

Abstract

The use of enhanced bandwidth estimation and loss differentiation 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 work first analyzes the problems faced by every bandwidth estimation and loss differentiation algorithm implemented at the sender side of a TCP connection. Some proposed algorithms are then reviewed, analyzing and comparing their estimation accuracy and performance in several realistic scenarios.

As we found that existing bandwidth estimation and loss differentiation algorithms exhibit poor performance, we first propose TIBET (Time Intervals based Bandwidth Estimation Technique). This is a new and efficient bandwidth estimation scheme that can be implemented within the TCP congestion control procedure, modifying only the sender-side of a connection.

Based on the accurate bandwidth estimate provided by TIBET, we then propose enhancements to existing loss differentiation algorithms, and in particular to the Vegas scheme, showing that it proves very accurate in discriminating the causes of packet losses in a wide range of operating conditions.

Therefore we propose to use the improved Vegas loss differentiation algorithm to enhance the TCP NewReno error-recovery scheme, thus avoiding unnecessary rate reduction that is caused by packet losses induced by bit corruption on the wireless channel. By extensive simulations and real Internet measures based on a TCP Linux implementation, we evaluate the performance of this enhanced TCP over a wide range of networks comprising wireless links, in terms of goodput, fairness, friendliness and transmission overhead. We show that it achieves higher goodput over wireless networks, still guaranteeing excellent

fairness and friendliness with classical TCP versions over wired links. Moreover, we show that our proposed TCP well approaches the upper goodput limit of an ideal scheme, where TCP is enlightened with perfect knowledge of the cause of packet losses.

Parts of this work have been published in the following papers [1, 2, 3, 4, 5]:

[1] A. Capone, L. Fratta, and *F. Martignon.* Bandwidth Estimation Schemes for TCP over Wireless Networks. *IEEE Transactions on Mobile Computing*, 3(2):129–143, 2004.

[2] S. Bregni, D. Caratti, and *F. Martignon.* Enhanced Loss Differentiation Algorithms for Use in TCP Sources over Heterogeneous Wireless Networks. In *Proceedings of GLOBECOM'03*, San Francisco, 1-5 Dec. 2003.

[3] A. Capone and *F. Martignon.* TCP with Bandwidth Estimation over Wireless Networks. In *Proceedings of VTC fall 2002*, September 2002.

[4] *F. Martignon.* Improving TCP Performance over Wireless Networks using Loss Differentiation Algorithms. In *IEEE Conference Mobile Wireless Ccommunication Networks MWCN 04*, October 2004.

[5] A. Capone and *F. Martignon.* Bandwidth Estimates in the TCP Congestion Control Scheme. In *Proceedings of Tyrrhenian International Workshop on Digital Communications (IWDC 2001)*, Taormina, Italy, Sep.2001.

Contents

Chapter 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, ad hoc 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. In other words, the assumption that packet loss is always an indicator of network congestion does not apply especially to heterogeneous networks that include wireless links, in which packet loss may be induced also by noise or any other reason than congestion. There, random loss due to bit corruption is misinterpreted: upon loss detection, the TCP sender reduces its transmission rate unnecessarily thus degrading the throughput performance [6, 7].

To avoid such limitation and degradation, several schemes have been proposed and are classified in [8], 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 [8].

However, end-to-end techniques, even if not as effective as link-layer protocols, can achieve further gain in performance by measuring network statistics and using enhanced error recovery techniques that differentiate the cause of packet losses based on TCP state variables and bandwidth estimates. In fact, the more information about current network conditions is available to TCP, the more efficiently it can use the network to transfer its data. In networks such as the Internet, TCP must form its own estimates of network properties based exclusively on measurements performed by the connection endpoints.

In this work we first develop accurate and efficient algorithms to solve two fundamental transport estimation problems:

- Bandwidth estimation, related to the development of end-to-end algorithms that measure the current TCP transmission rate and estimate the capacity available along the path from the source to the destination.

- Loss differentiation, that deals with the problem of classifying the cause of packet losses, as due to network congestion or to bit corruption on wireless links.

These two problems are interwoven as we will show that the most efficient Loss Differentiation Algorithms base their functioning on bandwidth measurements to estimate the network state.

We begin by discussing the problem of end-to-end bandwidth estimation for TCP, and we point out issues that could affect both the estimation accuracy and its impact on TCP tunable parameters. Then the 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 in [9, 10, 11]. The algorithm implemented by TCP Vegas is described in [12], and those adopted by TCP Westwood are presented in [13, 14]. Our analysis of the estimation algorithms proposed for TCP Westwood reveals an over-estimation of the available bandwidth, leading to

aggressive 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 [1, 3] (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.

Then we consider the issue of packet loss differentiation. To obtain an estimate of the cause of packet losses, some Loss Differentiation Algorithms (LDA) have been recently proposed [2, 15, 16]. These algorithms estimate the cause of packet losses based on TCP state variables and information from acknowledgement packets (ACK). Upon detection of a packet loss, the TCP sender bases on this estimate to decide the appropriate counteraction. The key feature for LDA schemes is to be accurate in ascribing the cause of packet losses, to allow TCP sources to take the appropriate reaction.

We evaluated the accuracy of several LDA schemes proposed in the literature (Vegas [12], Non Congestion Packet Loss Detection [17], Spike [18] and Flip Flop [16]) in various realistic scenarios, considering both wired and wireless links, and following an approach similar to that proposed in [19]. As LDA schemes presented in the literature exhibit poor performance in estimating the cause of packet losses, we propose enhancements to the Vegas, Non Congestion Packet Loss Detection (NCPLD) and Spike schemes, which achieve higher accuracy in all network scenarios.

To assess the accuracy of these schemes, we also provide an upper bound on the performance of LDA schemes, ideally assuming perfect knowledge of the cause of packet losses. It is then shown that our proposed enhanced schemes approach reasonably this bound.

Based on this observation, we propose to enhance the TCP NewReno error-recovery scheme using Loss Predictors (LP) [4], as proposed in [20, 21, 22], thus avoiding unnecessary rate reduction caused by packet losses induced by bit corruption on the wireless channel.

We evaluate the performance of this enhanced TCP (TCP NewReno-LP), showing that it achieves higher goodput over wireless networks, with both long-lived and short-lived TCP connections, while guaranteeing good friendliness with current TCP versions over wired links. We also evaluate the performance of TCP enhanced with ideal loss prediction, assuming perfect knowledge of the cause of packet losses, thus providing an upper bound. The TCP enhanced with Vegas loss predictor well approaches this ideal bound.

To support the results obtained through simulation, we have implemented the proposed enhanced transport protocols in three real test beds: the first one models an heterogeneous network that includes both wired and wireless links; the second one implements an ad hoc network while the third models a Cellular IP network with mobile hosts that experience temporary disconnections as they move from one area to the other of the network.

The results obtained with the test beds confirmed the performance gain achieved by TCP NewReno-LP as well as its friendliness towards classical TCP versions. As TCP NewReno-LP can be implemented by modifying the sender-side only of a TCP connection, this allows its immediate deployment in the Internet.

This work is structured as follows:

Chapter 2 introduces the Transmission Control Protocol, analyzing its congestion control and error recovery mechanisms. Chapter 3 studies the problem of bandwidth estimation performed at the sender-side of a TCP connection, and presents TIBET, a new and efficient bandwidth estimation algorithm; the performance of TCP sources enhanced with different bandwidth estimation algorithms is measured by simulation. Chapter 4 proposes enhanced Loss Differentiation Algorithms, and proposes TCP NewReno-LP, a new TCP version with an enhanced error recovery scheme based on a modified version of the Vegas Predictor. Chapter 5 analyses the performance of these enhanced Loss Differentiation

Algorithms, showing how they can achieve high accuracy in various simulated network scenarios. Chapter 6 measures and compares the performance of TCP sources enhanced with bandwidth estimation and loss differentiation. Also provided is an upper bound to the performance of these schemes, achieved by studying the behavior of TCP with ideal bandwidth estimation and loss differentiation capabilities. Chapter 7 presents the results obtained measuring the performance of the enhanced TCP sources in three real test beds. Finally, Chapter 8 concludes this work proposing issues left for future research.

Chapter 2

TCP Flow and Congestion Control

The *Transmission Control Protocol* (TCP) is the most widely used transport protocol in the Internet architecture. Indeed, the large majority of the traffic carried in the Internet is controlled by TCP.

TCP provides a connection oriented and reliable byte stream service, meaning that the two applications using TCP must establish a connection with each other before they can exchange data in a reliable way.

Figure 2.1 shows the position of TCP in the Internet architecture. While the *User Datagram Protocol* (UDP) provides a datagram service without guarantee of delivery, TCP provides a reliable service. Applications can use either UDP or TCP, based on their needs. Both TCP and UDP use the functionalities offered by the *Internet Protocol* (IP).

There are several reasons that brought to the development of a reliable transport protocol like TCP. At the network level, IP provides a datagram best-effort service, without any guarantee of packet delivery: packets can be lost due to transmission errors, congestion in routers or hardware failures. Moreover, as the IP routing can be dynamic, packets can be delivered out-of-order. However, software applications often need to transfer data in a reliable way between two or more hosts. TCP has the responsibility to offer such reliable service, hiding the unreliability of the IP network.

TCP offers to applications data segmentation and reassembly services, and implements flow and congestion control mechanism, as well as error recovery techniques, as we will

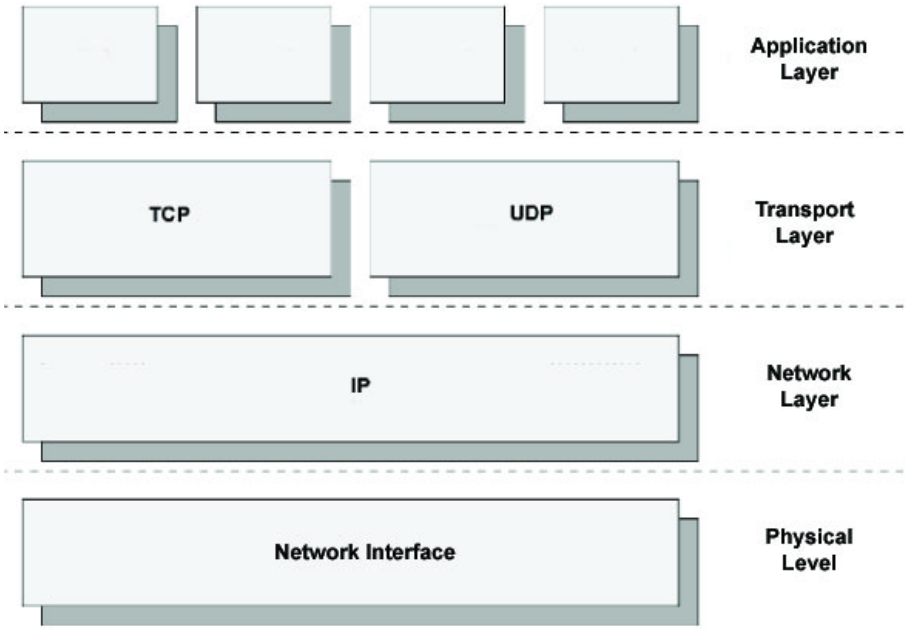


Figure 2.1: Internet Protocol Architecture.

detail in the following. The philosophy underlying the design of TCP is a black-box vision of the network: the network is not supposed to provide any explicit information to TCP. For this reason, TCP defines and updates several state variables (like the round trip time or the congestion window) that represent its actual view of the network state.

TCP implements a control on its transmission rate using a sliding window algorithm based on the receipt of *Acknowledgements* (ACK), and retransmits lost segments to achieve an errorless file transfer [23]. More precisely, TCP numbers every byte with a sequence number, and the number of the first byte contained in the segment is reported in the Sequence Number field of the segment itself, as illustrated in Figure 2.2 that shows the TCP header.

The segment also carries an Acknowledgement Number, that represents the number of the next byte that the TCP agent expects to receive from the peer entity. When a TCP source transmits a segment, it copies it in the transmission buffer and a timer is associated to such segment. When an ACK is received, the corresponding segment is erased from the transmission buffer. On the contrary, if the retransmission timeout associated to such packet expires before the corresponding ACK is received, the TCP sender retransmits the

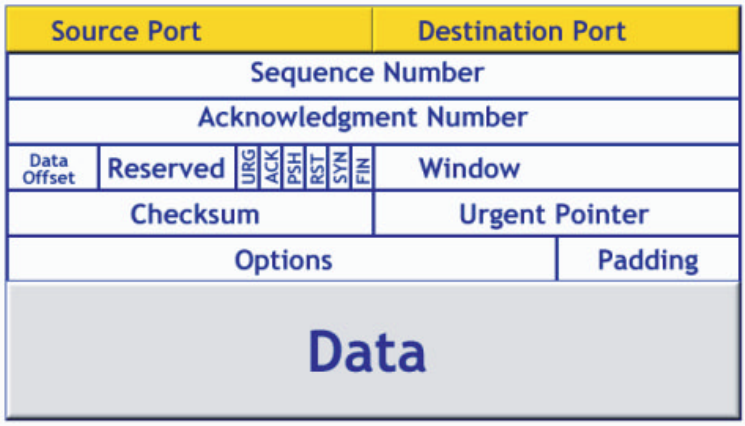


Figure 2.2: TCP Header.

segment.

2.1 Flow and Congestion Control Algorithms

To control the transmission of data between the TCP sender and the receiver, a sliding window algorithm is used. The TCP receiver performs a flow control by sending to the TCP sender information concerning the number of bytes, starting from the last acked byte, that it is willing to receive.

To illustrate the other algorithms that have been used in TCP to perform flow and congestion control it is necessary to define the following terms:

- A *segment* is any TCP/IP data or acknowledgment packet (or both).
- The *Maximum Segment Size* (MSS) is the size of the largest segment that can be transmitted in a TCP connection. This value can be based on the maximum transmission unit of the network, the path MTU discovery [24] algorithm, or other factors. During the connection startup, the TCP sender and receiver can specify the desired MSS. If not, the default value of 536 bytes is used. The size does not include the TCP/IP headers and options.
- The *Receiver Window* (rwnd) is the most recently advertised receiver window.

- *Congestion Window* (cwnd) is a TCP state variable that limits the amount of data a TCP source can send. At any given time, a TCP source must not send data with a sequence number higher than the sum of the highest acknowledged sequence number and the minimum of cwnd and rwnd.

- *The Flight Size* is the amount of data that has been sent but not yet acknowledged.

In the following Sections we define the four congestion control algorithms implemented in TCP: slow start, congestion avoidance, fast retransmit and fast recovery, developed in [25]. In some situations it may be beneficial for a TCP sender to be more conservative than the algorithms allow; however a TCP must not be more aggressive than the following algorithms allow (that is, it must not send data when the value of cwnd computed by the following algorithms would not allow the data to be sent).

2.2 Slow Start and Congestion Avoidance

The slow start and congestion avoidance algorithms are used by the TCP sender to control the amount of outstanding data being injected into the network. To implement these algorithms, two variables are added to the TCP per-connection state. The congestion window (cwnd) is a sender-side limit on the amount of data the sender can transmit into the network before receiving an acknowledgment (ACK), while the receiver's advertised window (rwnd) is a receiver-side limit on the amount of outstanding data. The minimum of cwnd and rwnd governs data transmission.

Another state variable, the slow start threshold (ssthresh), is used to determine whether the slow start or congestion avoidance algorithm is used to control data transmission, as discussed below.

Beginning transmission into a network with unknown conditions requires TCP to slowly probe the network to determine the available capacity, in order to avoid congesting the network with an inappropriately large burst of data. The slow start algorithm is used for this purpose at the beginning of a transfer, or after repairing loss detected by the retransmission timer.

The initial value of cwnd must be less than or equal to 2*MSS bytes and cannot be greater than 2 segments.

The initial value of ssthresh may be arbitrarily high (for example, some implementations use the size of the advertised window), but it may be reduced in response to congestion. The slow start algorithm is used when $cwnd < ssthresh$, while the congestion avoidance algorithm is used when $cwnd > ssthresh$. When cwnd and ssthresh are equal the sender may use either slow start or congestion avoidance.

During slow start, a TCP increments cwnd by at most MSS bytes for each ACK received that acknowledges new data. Slow start ends when cwnd exceeds ssthresh (or, optionally, when it reaches it, as noted above) or when congestion is observed.

During congestion avoidance, cwnd is incremented by 1 full-sized segment per round-trip time (RTT). Congestion avoidance continues until congestion is detected. One formula commonly used to update cwnd during congestion avoidance is given in the equation (2.1):

$$cwnd = cwnd + \frac{MSS^2}{cwnd} \tag{2.1}$$

This adjustment is executed on every incoming non-duplicate ACK. Equation (2.1) provides an acceptable approximation to the underlying principle of increasing cwnd by 1 full-sized segment per RTT. Note that for a connection in which the receiver acknowledges every data segment, equation (2.1) proves slightly more aggressive than 1 segment per RTT, and for a receiver implementing the Delayed-Acks mechanism, thus acknowledging every-other packet, equation (2.1) is less aggressive.

Another acceptable way to increase cwnd during congestion avoidance is to count the number of bytes that have been acknowledged by ACKs for new data. (A drawback of this implementation is that it requires maintaining an additional state variable.) When the number of bytes acknowledged reaches cwnd, then cwnd can be incremented by up to MSS bytes. Note that during congestion avoidance, cwnd cannot be increased by more than the larger of either 1 full-sized segment per RTT, or the value computed using equation (2.1).

Note that many implementations, including the TCP included in the Linux kernel, maintain cwnd in units of full-sized segments instead of in units of bytes. In these cases, equation (2.1) is difficult to apply and the above mentioned counting procedure is implemented.

When a TCP sender detects segment loss using the retransmission timer, the value of ssthresh is set to no more than the value given in equation (2.2):

$$ssthresh = max(FlightSize/2, 2 \cdot MSS) \tag{2.2}$$

As discussed above, FlightSize is the amount of outstanding data in the network.

Furthermore, upon a timeout cwnd is reset to 1 full-sized segment. Therefore, after retransmitting the dropped segment the TCP sender uses the slow start algorithm to increase the window from 1 full-sized segment to the new value of ssthresh, at which point congestion avoidance again takes over.

2.3 Fast Retransmit and Fast Recovery

When an out-of-order segment arrives at the TCP receiver, an immediate duplicate ACK is sent back to the sender. The purpose of this ACK is to inform the sender that a segment was received out-of-order and which sequence number is expected. From the sender's perspective, duplicate ACKs can be caused by a number of network problems. First, they can be caused by dropped segments: in this case, all segments after the dropped segment will trigger duplicate ACKs. Second, duplicate ACKs can be caused by the re-ordering of data segments by the network (not a rare event along some network paths [26]). Finally, duplicate ACKs can be caused by replication of ACK or data segments by the network. In addition, a TCP receiver can send an immediate ACK when the incoming segment fills in all or part of a gap in the sequence space. This will generate more timely information for a sender recovering from a loss through a retransmission timeout, a fast retransmit, or an experimental loss recovery algorithm, such as NewReno [27].

The TCP sender uses the fast retransmit algorithm to detect and repair loss, based on

incoming duplicate ACKs. The fast retransmit algorithm uses the arrival of 3 duplicate ACKs (4 identical ACKs without the arrival of any other intervening packets) as an indication that a segment has been lost. After receiving 3 duplicate ACKs, TCP performs a retransmission of what appears to be the missing segment, without waiting for the retransmission timer to expire.

After the fast retransmit algorithm sends what appears to be the missing segment, the fast recovery algorithm governs the transmission of new data until a non-duplicate ACK arrives. The reason for not performing slow start is that the receipt of the duplicate ACKs not only indicates that a segment has been lost, but also that segments are most likely leaving the network (although a massive segment duplication by the network can invalidate this conclusion). In other words, since the receiver can only generate a duplicate ACK when a segment has arrived, that segment has left the network and is in the receiver's buffer, so we know it is no longer consuming network resources. Furthermore, since the ACK clock [25] is preserved, the TCP sender can continue to transmit new segments (although transmission must continue using a reduced cwnd).

The fast retransmit and fast recovery algorithms are usually implemented together as follows.

1. When the third duplicate ACK is received, the ssthresh is set to no more than the value given in equation (2.2).

2. The lost segment is retransmitted and the cwnd is set to ssthresh plus 3*MSS. This artificially inflates the congestion window by the number of segments (three) that have left the network and which the receiver has buffered.

3. For each additional duplicate ACK received, the cwnd is incremented by a MSS. This artificially *inflates* the congestion window in order to reflect the additional segment that has left the network.

4. If allowed by the new value of cwnd and the receiver's advertised window, the TCP sender transmits a segment.

5. When the next ACK arrives that acknowledges new data, the cwnd is set equal to ssthresh (the value calculated in step 1). This is termed *deflating* the window. This ACK should be the acknowledgment elicited by the retransmission from step 1, one RTT after the retransmission (though it may arrive sooner in the presence of significant out-of-order delivery of data segments at the receiver). Additionally, this ACK should acknowledge all the intermediate segments sent between the lost segment and the receipt of the third duplicate ACK, if none of these were lost.

We note that this algorithm is known to generally not recover very efficiently from multiple losses in a single flight of packets [28]. We will evaluate numerically the performance of TCP is such scenarios, comparing it to the modified TCP versions proposed in this work. Recently, a set of measurements performed on various Web servers showed that only about the 44% of the segment retransmissions performed by the server were triggered by fast recovery algorithms, while the remaining 56% were due to retransmission timeout expirations.

The inefficiency of the fast recovery algorithm is put in evidence when the TCP connection uses small windows. This happens, for example, when the bandwidth-delay product[1] of the connection is small. In this case, as the number of packets in flight is low, it is unlikely for the TCP transmitter to receive 3 duplicate ACKs.

In all such cases, the segments loss cannot be recovered using fast retransmission and fast recovery, and it is necessary for the TCP sender to wait for a retransmission timeout to expire. In this case the *cwnd* is reset to 1 MSS and the *ssthresh* is halved, while the connection enters into the slow-start phase.

[1]The bandwidth-delay product of a TCP connection is defined as the product between the delay experienced by the connection and the capacity of the slowest link (bottleneck) along the connection's path [7].

2.4 The NewReno Modification to TCP's Fast Recovery Algorithm

The standard implementations of the Fast Retransmit and Fast Recovery algorithms illustrated above have been modified in TCP NewReno [RFC2581]. TCP NewReno introduces two variations to the 5-step algorithm presented in the previous Section: the utilisation of the variable "recover" in step 1, and the response to a partial or new acknowledgement in step 5. The modification defines a "Fast Recovery procedure" that begins when three duplicate ACKs are received and ends when either a retransmission timeout occurs or an ACK arrives that acknowledges all of the data up to and including the data that was outstanding when the Fast Recovery procedure began.

1. When the third duplicate ACK is received and the sender is not already in the Fast Recovery procedure, the ssthresh is set to no more than the value given in equation (2.2).

   In addition, the highest sequence number transmitted is recorded in the variable "recover".

2. The lost segment is retransmitted and the cwnd is set to ssthresh plus 3*MSS. This artificially inflates the congestion window by the number of segments (three) that have left the network and which the receiver has buffered.

3. For each additional duplicate ACK received, the cwnd is incremented by a MSS. This artificially *inflates* the congestion window in order to reflect the additional segment that has left the network.

4. If allowed by the new value of cwnd and the receiver's advertised window, the TCP sender transmits a segment.

5. When an ACK arrives that acknowledges new data, this ACK could be the acknowledgment elicited by the retransmission from step 2, or elicited by a later retransmission.

If this ACK acknowledges all of the data up to and including "recover", then the ACK acknowledges all the intermediate segments sent between the original transmission of the lost segment and the receipt of the third duplicate ACK. The cwnd is set to either (1) min (ssthresh, FlightSize + MSS) or (2) ssthresh, where ssthresh is the value set in step 1; this is termed "deflating" the window. (We note that FlightSize in step 1 referred to the amount of data outstanding in step 1, when Fast Recovery was entered, while FlightSize in step 5 refers to the amount of data outstanding in step 5, when Fast Recovery is exited.) If the second option is selected, the implementation should take measures to avoid a possible burst of data, in case the amount of data outstanding in the network was much less than the new congestion window allows. Then the TCP connections exits the Fast Recovery procedure.

If this ACK does *not* acknowledge all of the data up to and including "recover", then this is a partial ACK. In this case, the first unacknowledged segment is retransmitted. The congestion window is deflated by the amount of new data acknowledged, then one MSS is added back and a new segment is sent if permitted by the new value of cwnd. This "partial window deflation" attempts to ensure that, when Fast Recovery eventually ends, approximately ssthresh amount of data will be outstanding in the network. In this case the TCP connections does not exit the Fast Recovery procedure (i.e., if any duplicate ACKs subsequently arrive, execute Steps 3 and 4 above).

For the first partial ACK that arrives during Fast Recovery, also reset the retransmit timer.

Note that in Step 5, the congestion window is deflated when a partial acknowledgement is received. The congestion window was likely to have been inflated considerably when the partial acknowledgement was received. In addition, depending on the original pattern of packet losses, the partial acknowledgement might acknowledge nearly a window of data. In this case, if the congestion window was not deflated, the data sender might be able to send nearly a window of data back-to-back.

There are several possible variants to the simple response to partial acknowledgements described above. First, there is a question of when to reset the retransmit timer after a partial acknowledgement.

There is a related question of how many packets to retransmit after each partial acknowledgement. The algorithm described above retransmits a single packet after each partial acknowledgement. This is the most conservative alternative, in that it is the least likely to result in an unnecessarily-retransmitted packet. A variant that would recover faster from a window with many packet drops would be to effectively Slow-Start, requiring less than N round trip times to recover from N losses [10]. With this slightly-more-aggressive response to partial acknowledgements, it would be advantageous to reset the retransmit timer after each retransmission. Because we have not experimented with this variant in our simulations, we do not discuss this variant further in this work.

A third question involves avoiding multiple Fast Retransmits caused by the retransmission of packets already received by the receiver. Avoiding multiple Fast Retransmits is particularly important if more aggressive responses to partial acknowledgements are implemented, because in this case the sender is more likely to retransmit packets already received by the receiver.

As a final note, we would observe that in the absence of the Selective Acknowledgements (SACK) option, the data sender is working with limited information. One could spend a great deal of time considering exactly which variant of Fast Recovery is optimal for which scenario in this case. When the issue of recovery from multiple dropped packets from a single window of data is of particular importance, the best alternative would be to use the SACK option.

2.5 Computing TCP's Retransmission Timer

One of the most important tasks that a TCP connection must perform is to estimate a proper value of the *Retransmission TimeOut* (RTO). In the Internet, in fact, a segment sent by a TCP source can find on its path different scenarios: high-speed LANs, satellite links with high delay, low-capacity dialup lines, wireless links and so on. Hence, it

is impossible for the TCP sender to know a-priori the Round Trip Time (RTT) of the connection. Moreover, the RTT depends from the load of the network and, more specifically, of the routers traversed by the connections; as a consequence, the RTT can vary significantly during the connection's lifetime.

To adapt to the different network conditions, TCP implements an adaptive retransmission algorithm. More specifically, TCP tries to compute consistent values for the retransmission timeout based on measures of *round trip samples*, obtained by the TCP sender by measuring the interval between the transmission of one segment and the reception of the corresponding ACK.

The RTO is then computed based on the following equation:

$$RTO = SRTT + 4 * RTTVAR \tag{2.3}$$

where $SRTT$ (*Smoothed Round Trip Time*) is a low-pass filtered version of the round trip samples (RTT_M) measured by the TCP sender, calculated as follows:

$$SRTT[k] = (1 - \frac{1}{8}) * SRTT[k-1] + \frac{1}{8} * RTT_M[k] \tag{2.4}$$

$RTTVAR$ (*Round Trip Time Variation*), instead, represents an estimate of the standard deviation of the round trip samples, and it is calculated by low-pass filtering the quantity $|SRTT - RTT_M|$, according to the following equation:

$$RTTVAR[k] = (1 - \frac{1}{4}) * RTTVAR[k-1] + \frac{1}{4} * |SRTT - RTT_M| \tag{2.5}$$

It is important to notice that all the measures of the Round Trip Time are performed by TCP sources using a clock whose granularity, indicated with G, is often quite low [29]. In some TCP implementations, like BSD, G can reach values up to 500 ms; hence, if the actual RTT of the connection is lower than such value of G, as it happens for the majority of TCP connections, its value would be rounded up to 500 ms.

The low precision of these measures often implies an overestimate of the retransmission timeout. A series of tests performed over real TCP connections in the Internet [12] showed

that the average RTO computed by a TCP Reno connection is 1100 ms, while using a more precise clock would have lead to a correct estimate of less than 300 ms.

Clearly, this implies a degradation of TCP performance. Furthermore, we will show in the next Chapter that high values of G can seriously affect the bandwidth estimation processes performed at the TCP sender side. However, we will propose a simple technique that allows to achieve good precision even with high-granularity clocks.

To increase the precision of the estimated RTO, TCP can use the *timestamp* option described in [29]. In this way, the TCP sender inserts a timestamp in the segments' header. When the receiver processes an incoming segment, it copies the timestamp value in the header of the corresponding ACK. The TCP sender can thus obtain a precise estimate of the current RTT by simply subtracting the current time to the received timestamp value.

Evidently, as both TCP sender and receiver cooperate to implement this process, it is necessary to negotiate the utilization of the timestamp option during the three-way handshake phase.

Chapter 3

Bandwidth Estimation Algorithms

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 [9, 11], 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 [30], 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 [13], the TCP slow start threshold (*ssthresh*) is related to the *byte-equivalent* of the estimated bandwidth (Bwe) according to the following relation:

$$Ssthresh = Bwe \cdot RTT_{min} \tag{3.1}$$

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.

3.1 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 [30, 31]
- ACK Compression [30, 31]
- TCP coarse-grained clocks [12, 29]
- Rerouting [32]

3.1.1 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 [30, 31]. 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.

3.1.2 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 [31]. 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 1500 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 [30], cannot be neglected.

3.1.3 TCP coarse-grained clocks

TCP must translate the estimated bandwidth into parameters used in its congestion control scheme. It has been shown [10] 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 relation (3.1).

However, TCP measures RTT with a *coarse-grained* clock [29]. 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 ten times higher than the correct one. Therefore, even if the estimated bandwidth value is correct, the *ssthresh* would be set to ten times its correct value, thus leading to a very aggressive behavior of the connection.

3.1.4 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 equation (3.1) 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.

3.2 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 [11]: *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.

3.2.1 Packet-Pair Algorithm

The Packet Pair algorithm [9] 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 [10]. 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 [33, 34] 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 work since we focus our analysis on passive techniques that simply exploit regular TCP traffic.

3.2.2 TCP Vegas Estimation Algorithm

A more sophisticated bandwidth estimation scheme, active throughout the connection time, has been adopted in TCP Vegas [12]. 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 $diff_V = (expected\_Rate - actual\_Rate) \cdot RTT_{min}$. The congestion window size ($cwnd$) is then increased by one if $diff_V < 1$, decreased by one if $diff_V > 3$ and left unchanged if $1 \leq diff_V \leq 3$.

Although the TCP Vegas algorithm often leads the congestion window size to an equilibrium point [12], 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 [32].

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 [32].

3.2.3 TCP Westwood Estimation Algorithms

TCP Westwood, recently proposed in [35, 13, 36, 14], 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 [35], 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 pseudo-code:

Algorithm 3.2.1: ALGORITHM WESTWOOD 1 $(acked, pkt\_size, now)$

if (ACK is received)

$$\textbf{then} \begin{cases} sample\_BWE[k] = \frac{acked \cdot pkt\_size \cdot 8}{now - last\_ACK\_time} \\ BWE[k] = \beta \cdot BWE[k-1] + \frac{1-\beta}{2}(sample\_BWE[k] + sample\_BWE[k-1]) \end{cases}$$

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 [35] 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.

The improved version of the TCP Westwood algorithm proposed in [13] and described in the following pseudo-code adopts a nonlinear filtering technique:

Algorithm 3.2.2: ALGORITHM WESTWOOD 2 $(acked, pkt\_size, now)$

if (ACK is received)

$$\textbf{then} \begin{cases} ACK\_interval = now - last\_ACK\_time \\ sample\_BWE[k] = \frac{acked \cdot pkt\_size \cdot 8}{ACK\_interval} \\ pole = \frac{2\tau - ACK\_interval}{2\tau + ACK\_interval} \\ BWE[k] = pole \cdot BWE[k-1] + \frac{1-pole}{2} \cdot (sample\_BWE[k] + sample\_BWE[k-1]) \end{cases}$$

We analyzed the performance of this algorithm. First we tested the filter with constant packet lengths (equal to 1000 bytes) and sequences of interarrival times with various

probability distributions. Figure 3.1 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.

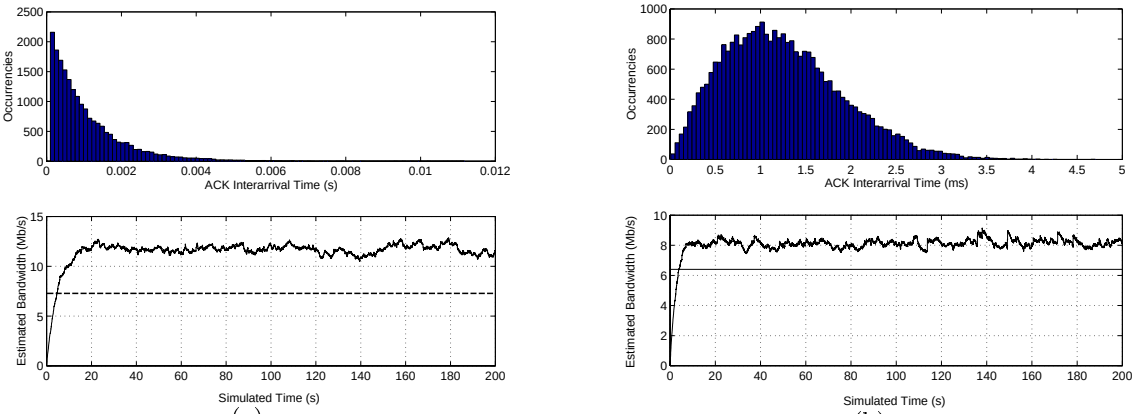


Figure 3.1: Algorithm Westwood 2 filter: estimated bandwidth values with constant packet length equal to 1000 bytes and random independent interarrivals: (a) exponentially distributed (b) Rayleigh distributed. The dotted lines in the two lower figures represent the correct bandwidth estimate, equal to the packet length divided by the average interarrival time.

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 [37]; this was the simulator used for all the results presented in this work. 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 this Chapter, that the TCP Maximum Segment Size is 1000 bytes, that the TCP receiver implements the Delayed ACK algorithm as recommended in [23], 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 (Figure 3.2) 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. Figure 3.3 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.

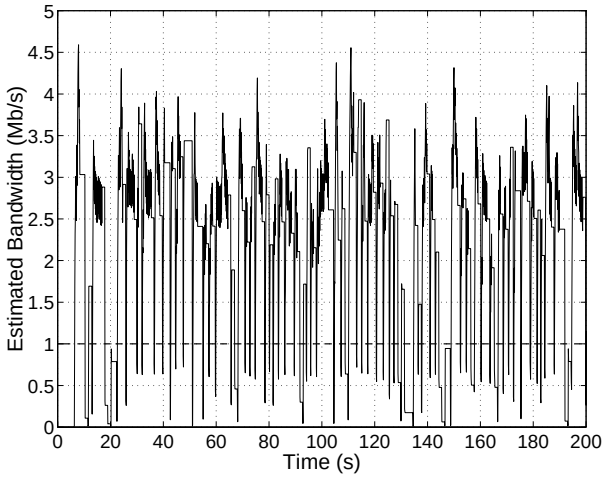


Figure 3.2: TCP Westwood: estimated bandwidth with 10 connections sharing a single 10 Mbit/s link.

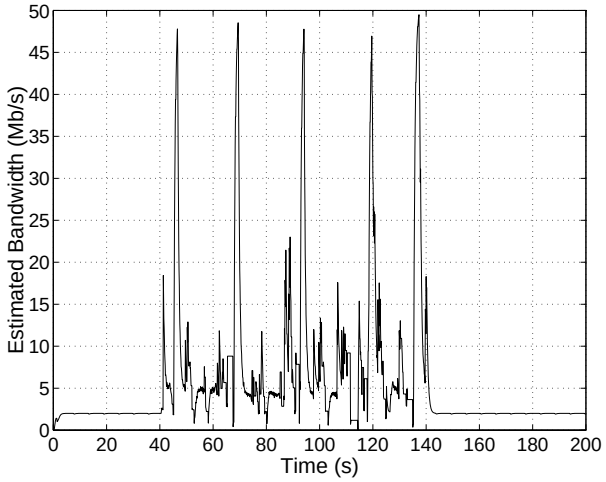


Figure 3.3: TCP Westwood: estimated bandwidth in the presence of ACK compression. 2 connections cross a 2 Mbit/s link in opposite directions.

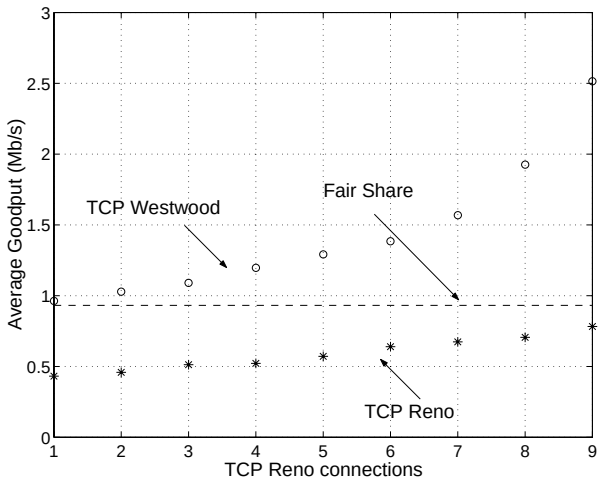


Figure 3.4: Link utilization of TCP Westwood and TCP Reno sources sharing the same 10 Mb/s link.

To investigate the effect of a non accurate 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). Figure 3.4 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 non mixed 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.

Recently, the estimation algorithm of TCP Westwood was modified further [36, 14], the new approach adopting time varying coefficients in the filter with both adaptive gain and adaptive sampling. The following pseudo-code specifies the modified approach [36, 14]:

Algorithm 3.2.3: ALGORITHM WESTWOOD 3 $(acked, pkt\_size, now)$

if (ACK is received)

$$\textbf{then} \begin{cases} sample\_BWE[k] = \dfrac{\text{Bytes received in T[k]}}{\text{T[k]}} \\ pole[k] = \frac{2\tau[k]-ACK\_interval}{2\tau[k]+ACK\_interval} \\ BWE[k] = pole[k] \cdot BWE[k-1] + (1 - pole[k]) \cdot sample\_BWE[k] \end{cases}$$

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 [14].

We tested the accuracy of this filter with the same sequences of independent random interarrival times used for the results in Figure 3.1. The bandwidth estimated by Westwood 3, shown in Figure 3.5(a) and (b), is improved with respect to Westwood 2 (see Figure 3.1). However, the estimate is still biased.

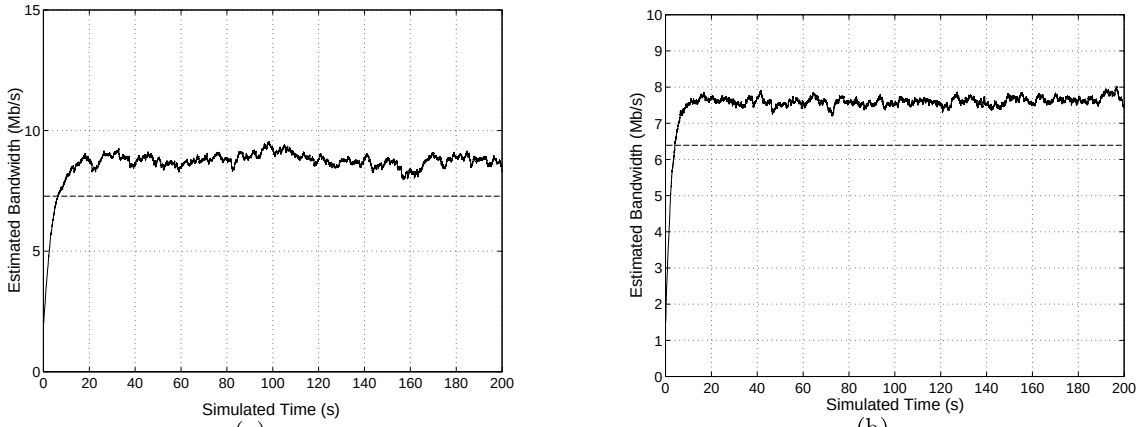


Figure 3.5: Westwood 3 filter: estimated bandwidth values with constant packet length equal to 1000 bytes and random independent interarrivals (a) exponentially distributed (b) Rayleigh distributed. The dotted lines represent the correct bandwidth estimate.

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 proved 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:

$$E[Z] = E[L] \cdot \sum_{n=0}^{+\infty} (-1)^n \frac{E[(T - E[T])^n]}{(E[T])^{n+1}} = \frac{E[L]}{E[T]} + E[L] \cdot \sum_{n=2}^{+\infty} (-1)^n \frac{E[(T - E[T])^n]}{(E[T])^{n+1}} \quad (3.2)$$

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 variance of T, usually quite high due to ACK clustering. The region of convergence of the series (3.2) is [0,2E[T]].

As an example, Figure 3.6 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.

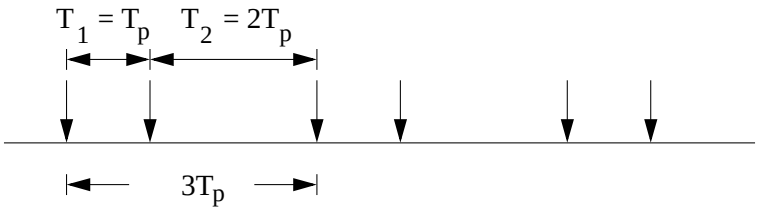


Figure 3.6: ACKs arrival pattern.

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 $2L_p/27T_p$.

3.2.4 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) [38], 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 equation (3.1) after congestion events.

The bandwidth estimation algorithm is described by the following equation:

$$Bwe[k] = (1 - e^{\frac{-T[k]}{K}}) * \frac{L[k]}{T[k]} + e^{\frac{-T[k]}{K}} * Bwe[k-1] \tag{3.3}$$

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 [38] 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 Figure 3.7, 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 $K = 20$, the bandwidth variation due to the UDP flow is estimated too late (Figure 3.7(a) and 3.7(b)).

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.

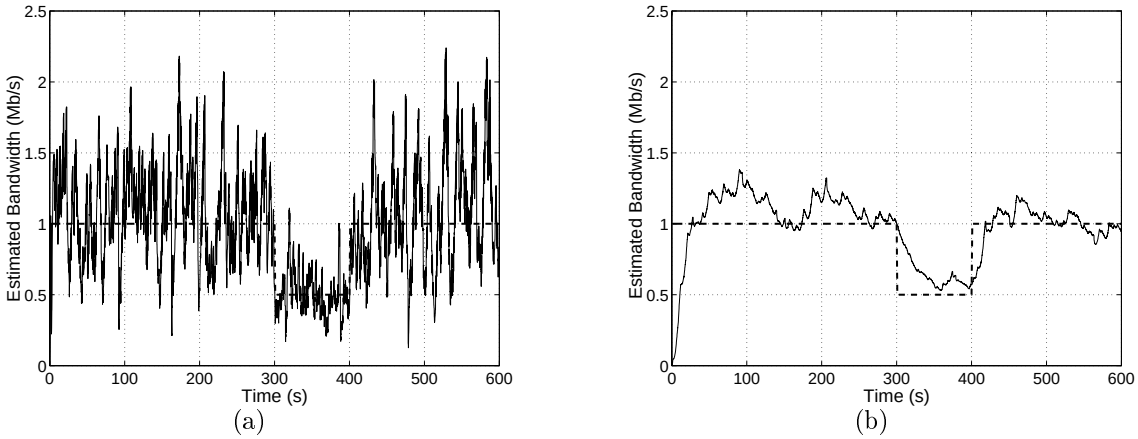


Figure 3.7: CSFQ bandwidth estimation with (a) $K = 0.5$ s (b) $K = 20$ sec

3.3 TIBET

In this section we present TIBET (Time Intervals based Bandwidth Estimation Technique) [1], 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.

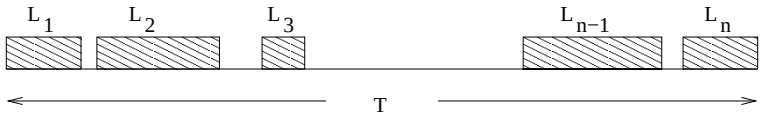


Figure 3.8: Pattern of packet transmission.

To explain the rationale of TIBET let us refer to the example in Figure 3.8 where transmissions occurring in a period T are considered. Let n be the number of packets belonging to a connection and $L_1, L_2...L_n$ the lengths, in bits, of these packets. The average bandwidth, Bw, used by the connection is given by

$$Bw = \frac{1}{T}\sum_{i=1}^{n} L_i = \frac{n\overline{L}}{T} = \frac{\overline{L}}{\frac{T}{n}} \tag{3.4}$$

The estimate of the average bandwidth can be obtained by performing a run-time 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.

3.3.1 Estimation Scheme

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

Algorithm 3.3.1: TIBET APPLIED TO SENT PACKETS (pkt_size, now)

if (Packet is Sent)

$$\textbf{then} \begin{cases} sample\_length[k] = pkt\_size \cdot 8 \\ sample\_interval[k] = now - last\_sending\_time \\ Average\_pkt\_length[k] = \alpha \cdot Average\_pkt\_length[k-1] + \\ \qquad\qquad +(1-\alpha) \cdot sample\_length[k] \\ Average\_interval[k] = \alpha \cdot Average\_interval[k-1] + \\ \qquad\qquad +(1-\alpha) \cdot sample\_interval[k] \\ Bwe[k] = \frac{Average\_packet\_length[k]}{Average\_interval[k]} \end{cases}$$

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_ pkt_ length* and *average_ interval* are the low-pass filtered measures of the packet length and the interdeparture times, respectively. α ($0 \leq \alpha \leq 1$) is the pole of the two low-pass filters. *Bwe* is the estimated value of the used bandwidth. The value of α has a critical impact on TIBET performance: if α 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 α 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 $\alpha = 0.99$ 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*:

Algorithm 3.3.2: TIBET APPLIED TO RECEIVED ACKS $(acked, pkt\_size, now)$

if (ACK is Received)

$$\textbf{then} \begin{cases} sample\_length[k] = acked \cdot pkt\_size \cdot 8 \\ sample\_interval[k] = now - last\_ack\_time \\ Average\_pkt\_length[k] = \alpha \cdot Average\_pkt\_length[k-1] + \\ \qquad\qquad +(1-\alpha) \cdot sample\_length[k] \\ Average\_interval[k] = \alpha \cdot Average\_interval[k-1] + \\ \qquad\qquad +(1-\alpha) \cdot sample\_interval[k] \\ Bwe[k] = \frac{Average\_packet\_length[k]}{Average\_interval[k]} \end{cases}$$

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 equation (3.1), the bandwidth first being estimated by one of the two above procedures. The overall procedure is specified by the following psudo-code:

Algorithm 3.3.3: TIBET $(ErrorRecovery)$

if (3 DUPACK are received)

$$\textbf{then} \begin{cases} ssthresh \leftarrow BWE \cdot RTT_{min} \\ \textbf{if } (cwnd > ssthresh) \\ \quad \textbf{then } \{ cwnd \leftarrow ssthresh \end{cases}$$

Retransmit the first unacknowledged segment

if (Retransmission Timeout expires)

$$\textbf{then} \begin{cases} ssthresh \leftarrow BWE \cdot RTT_{min} \\ cwnd \leftarrow 1 \end{cases}$$

Retransmit the first unacknowledged segment

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

3.3.2 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 Figure 3.9(a).

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:

$$Bwe[k] = (1 - e^{\frac{-T[k]}{T_0}}) * \frac{Average\_packet\_length[k]}{Average\_interval[k]} + e^{\frac{-T[k]}{T_0}} * Bwe[k-1] \qquad (3.5)$$

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 equation 3.3).

With this adaptive filter the estimate is smoothed, practically overlapping the fair share curve as shown in Figure 3.9(b).

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 Figure 3.1. The bandwidth estimated by TIBET, Figure 3.10(a) and 3.10(b), is very close to the correct value, shown by the straight dotted line. The improvement with respect to Westwood (see Figures 3.1 and 3.5) is remarkable.

We also tested TIBET behavior in a simulated scenario where a single TIBET source performs a file transfer over a 10 Mb/s link, with a RTT of 100 ms, sharing the link with two UDP ON/OFF sources. The dotted line in Figure 3.11(a) shows the bandwidth

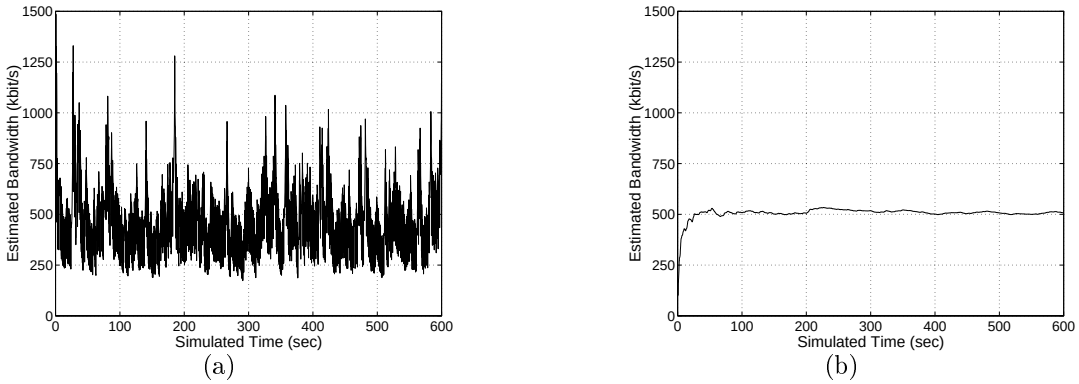


Figure 3.9: TIBET bandwidth estimate (a) without adaptive filtering (b) with adaptive filtering.

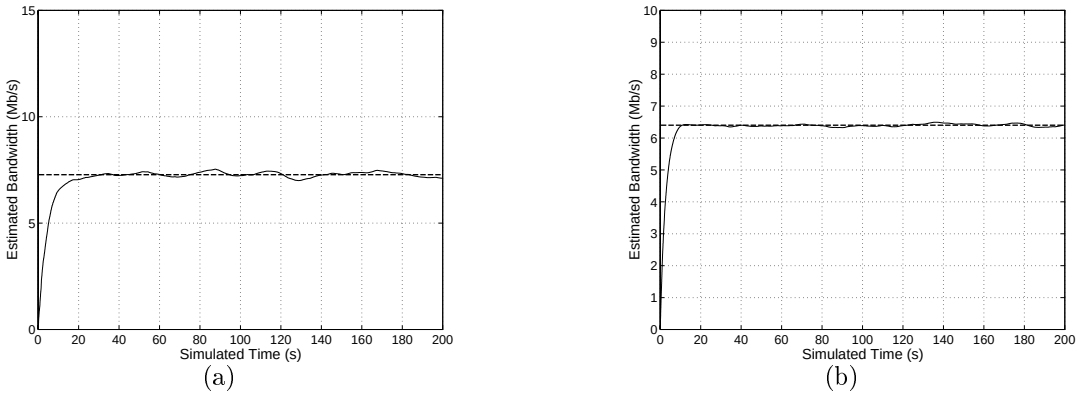


Figure 3.10: TIBET filter: estimated bandwidth values with constant packet length equal to 1000 bytes and random independent interarrivals (a) exponentially distributed (b) Rayleigh distributed. The dotted lines represent the correct bandwidth estimate.

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. 200 seconds after the beginning of the transmission, a UDP source starts transmitting with a constant bit rate equal to the 95% 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

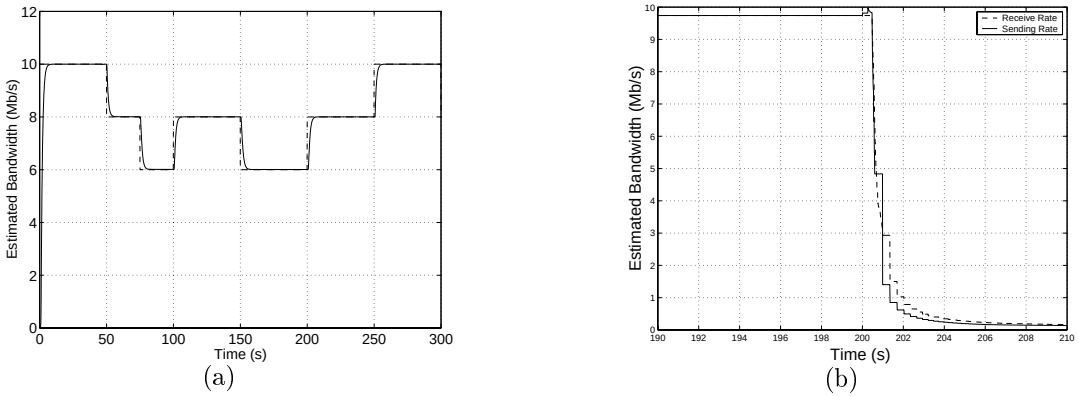


Figure 3.11: (a) TIBET bandwidth estimate with concurrent UDP traffic with variable rate (b) Comparison between sending and receive rate over a 10Mb/s link with a sudden surge of UDP traffic.

stream of transmitted packets or to the stream of received ACKs. Figure 3.11(b) 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 8592 segments, and the one estimating the receive rate transmitted only 2 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 second 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 Figure 3.3 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 connections transmit packets, and ACK compression is observed. The bandwidth estimate shown in Figure 3.12 is not at all affected by the ACK compression, proving the much higher robustness of TIBET, compared to TCP Westwood (see Figure 3.3)

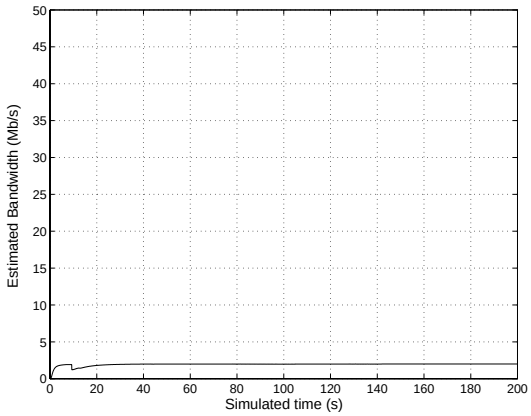


Figure 3.12: Bandwidth Estimated by a single TIBET source over a 2Mb/s link in the presence of ACK compression.

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 Figure 3.4, where TIBET sources substitute TCP Westwood sources. The average goodputs of TIBET and TCP Reno connections are shown in Figure 3.13(a). 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.

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.

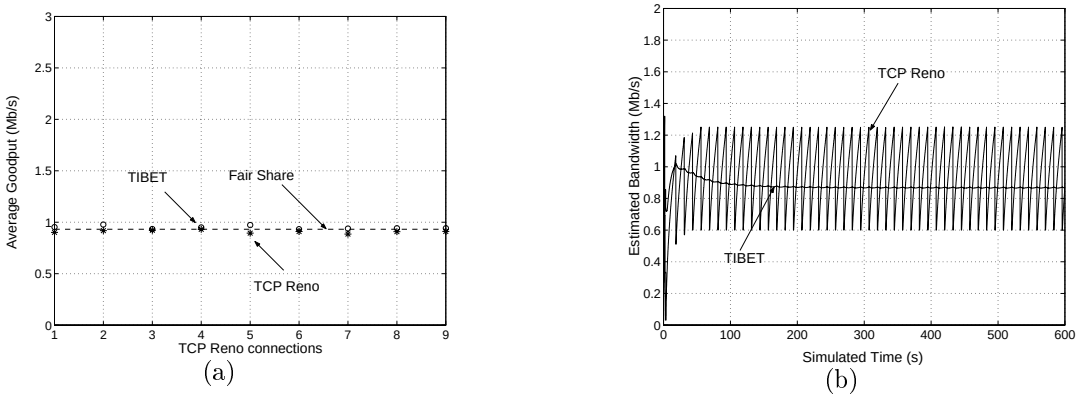


Figure 3.13: (a) TIBET fairness towards TCP Reno over a 10 Mbit/s link (b) Comparison between TCP Reno current rate and TIBET bandwidth estimate over a 2 Mbit/s link.

From the simulation results related to the scenarios considered we can claim that TIBET shows a quite fair behavior towards 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 Figure 3.13(b) 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, in the 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 second 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 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:

Algorithm 3.3.4: ENHANCED UPDATE OF $RTT_{min}(RTT_{min})$

if (The connections experiences a congestion event)

$$\textbf{then} \begin{cases} n \leftarrow n+1 \\ \textbf{if } (n = N_{cong}) \\ \quad \textbf{then} \begin{cases} RTT_{min} = \gamma \cdot RTT_{min} \\ n \leftarrow 0 \end{cases} \end{cases}$$

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 seconds 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

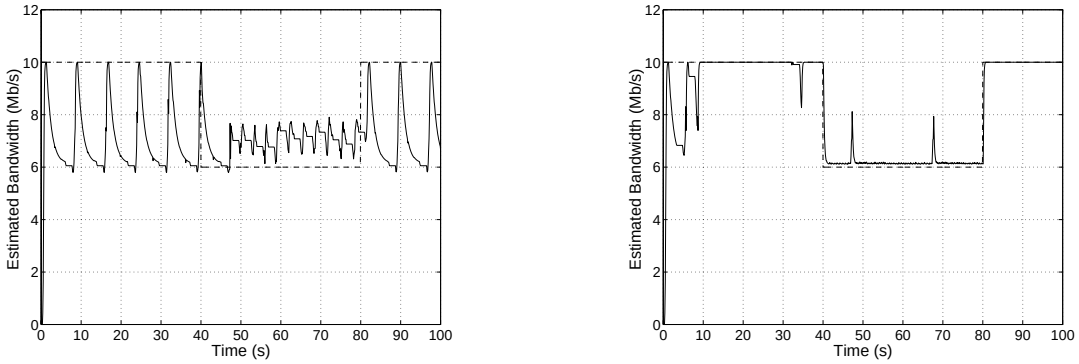


Figure 3.14: Bandwidth Estimate executed (a) in absence (b) in the presence of RTT_{min} updating algorithm.

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 Figure 3.14(a) and 3.14(b). In both figures, the dotted line represents the actual bandwidth.

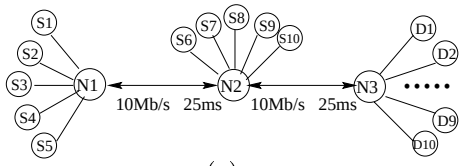
Without the algorithm (Figure 3.14(a)) the connection is very aggressive and the link is often congested. With the algorithm, and assuming $N_{cong} = 5$, $\gamma = 0.5$, the bandwidth estimate is more accurate, and link congestion is more effectively controlled (Figure 3.14(b)): the estimate, except 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 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, Figure 3.15(a), 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 25ms. The RTT of the sources $S1 - S5$ is equal to 100ms, while the RTT of sources $S6 - S10$ is equal to 50 ms.



(a)

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

(b)

Figure 3.15: (a) Network topology with differential Round Trip Times across the connections (b) Goodput achieved (kbit/s) by TCP Reno, TCP Sack and TIBET in this topology.

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 Figure 3.15(b). TIBET, similarly to Westwood studied in [39], achieves better fairness between connections having different round trip times.

3.3.3 Performance of TCP sources enhanced with TIBET

So far we have 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 this section we measure TIBET performance in some simple scenarios, to underline how an efficient bandwidth estimate can be exploited by a TCP source to increase its goodput over wireless networks.

More detailed performance evaluation and comparison with other TCP versions will be presented in Chapters 6 and 7, where both simulated results and real test bed measures are considered.

Uncorrelated Losses

For long-lived TCP connections performing FTP transfers we considered two 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 Figure 3.16.

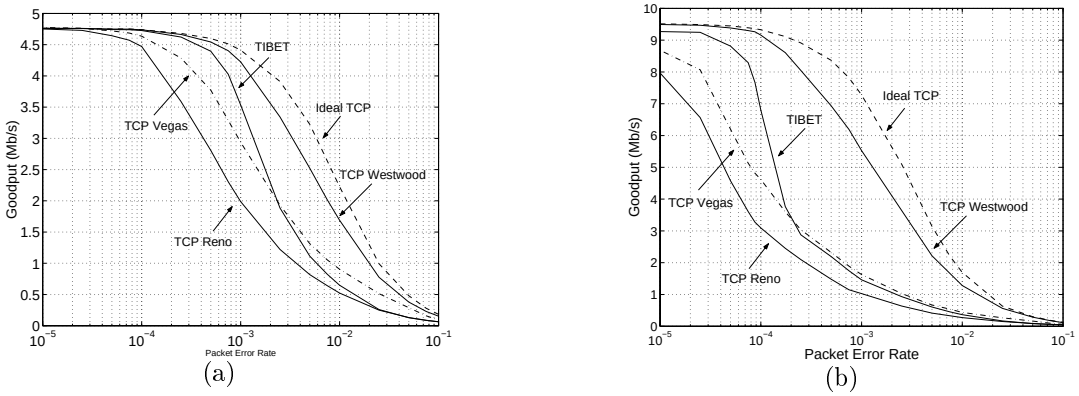


Figure 3.16: Various TCP implementations' goodput vs packet error rate over a (a) 5 Mbit/s link (b) 10 Mbit/s link.

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 [27] and TCP Sack [40]. Their goodputs are not shown since, in the range of considered packet losses, they practically overlap the goodput of TCP Reno, in agreement with what was pointed out in [41, 42].

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.

Correlated Losses

To account for the effects of multi-path 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 [43], 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% to take into account various levels of fading. It can be seen from Figure 3.17 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 3.2.3.

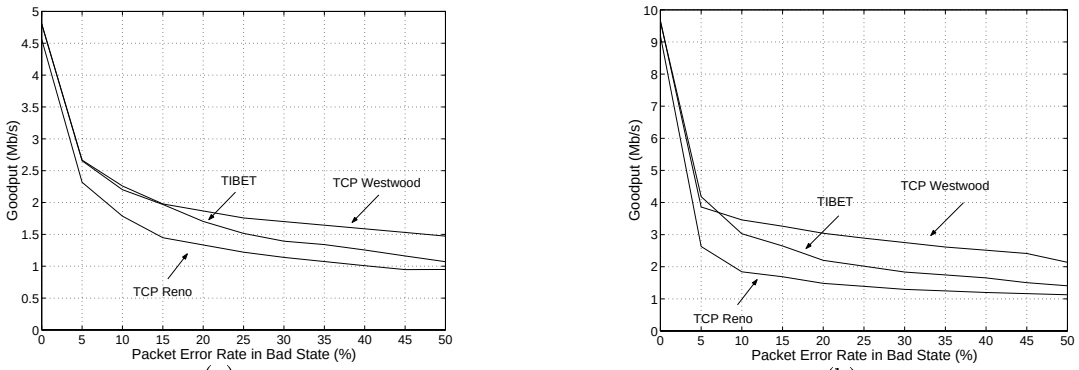
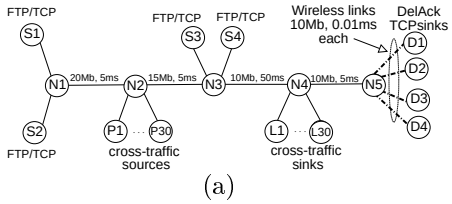


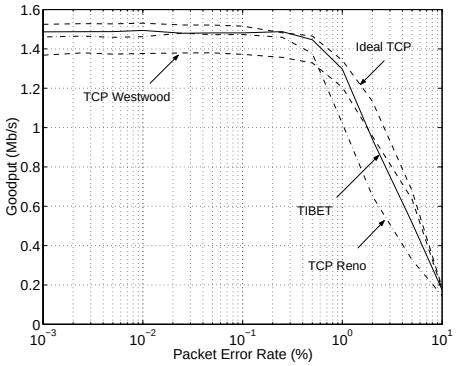
Figure 3.17: TIBET, TCP Westwood and TCP Reno goodput achieved over a link with capacity (a) 5 Mb/s (b) 10 Mb/s, affected by correlated losses, as a function of the packet error rate in the Bad state.

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 Figure 3.18(a) with 4 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 the ON period, each source transmits packets with 1500 byte size at constant bit rate equal to 0.5 Mbit/s; while in OFF period, the UDP sources transmit no packet.



(a)



(b)

Figure 3.18: (a) Mixed wired-wireless multi-hop network topology (b) Goodput achieved by TIBET, TCP Westwood, TCP Reno and Ideal TCP as a function of the packet error rate.

Figure 3.18(b) 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.

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 3.3.1:

Algorithm 3.3.5: TCP With Ideal Bandwidth Estimate($ErrorRecovery$)

if (received DUPACK = 3)

then $\begin{cases} ssthresh \leftarrow Exact\_Available\_Bw \cdot Exact\_RTT \\ \textbf{if } (cwnd > ssthresh) \\ \quad \textbf{then } \{ cwnd \leftarrow ssthresh \end{cases}$

Retransmit the first unacknowledged segment

if (Retransmission Timeout expires)

then $\begin{cases} ssthresh \leftarrow Exact\_Available\_Bw \cdot Exact\_RTT \\ cwnd \leftarrow 1 \cdot MSS \end{cases}$

Retransmit the first unacknowledged segment

The throughput achieved by this ideal scheme was obtained by simulation, and is shown by the dotted curves in Figures 3.16 and 18(b). The high goodput degradation measured, even with the Ideal TCP, at high error rates proves that such 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.

Chapter 4

TCP enhanced with Loss Differentiation

Loss Differentiation Algorithms (LDA), also known in the literature as Loss Predictors (LP), are used to provide TCP with an estimate of the cause of packet losses, to improve performance over heterogeneous networks including wired and wireless links. As we found that the LDA schemes proposed in literature exhibit poor performance in estimating the cause of packet losses, we propose enhancements to the Vegas, Non Congestion Packet Loss Detection (NCPLD) and Spike schemes, which achieve higher accuracy in all network scenarios.

4.1 Loss Differentiation Algorithms

The loss differentiation decision can be obtained by processing TCP state variables, such as congestion window (cwnd), slow start threshold (ssthresh) and Round Trip Time (RTT). Moreover, LDA can use additional parameters, such as the bandwidth estimation algorithms described in the previous Chapter. In conclusion, a generic LDA scheme can be seen as a Boolean function of TCP state variables and sender-side network measurements, that returns the network state when a packet loss occurs: either congested or not congested, as depicted in Figure 4.1.

Figure 4.1: Generic Scheme of a Loss Differentiation Algorithm.

4.1.1 Enhanced Vegas Loss Predictor

A first class of LDA schemes estimates the cause of packet losses based on rate estimates. In particular, TCP-Vegas [12] bases its decision on a loss predictor, $diff_V$, calculated as

$$diff_V = RTT_{min} \cdot (ER - AR) = cwnd \cdot \left(1 - \frac{RTT_{min}}{RTT}\right) \tag{4.1}$$

where cwnd is the congestion window, RTT_{min} is the minimum RTT sample measured during the TCP session, $cwnd/RTT$ is an estimate of the actual transmission rate (AR) of the TCP source, $cwnd/RTT_{min}$ is an estimate of the expected transmission rate (ER).

In [20] the authors propose to compare $diff_V$ to a single threshold, set to 1 segment: when $diff_V \leq 1$ losses are ascribed to transmission errors, while when $diff_V > 1$ losses are ascribed as due to congestion.

However, we found both by simulation and real Internet measurements that this predictor achieves very low accuracy in classifying losses due to random impairments on the wireless link. This is in line with the observations presented in [20].

To improve the performance of the Vegas predictor, we propose to use two parameters α and β [segments], when $diff_V \geq \beta$, the Vegas predictor assumes that the network is congested; when $diff_V \leq \alpha$, possible losses will be ascribed to transmission random errors. Finally, when $\alpha < diff_V < \beta$, the predictor assumes that the network state is the same as in the previous estimation.

Intuitively, if the expected rate is greater than the actual transmission rate, then the network is likely to experience congestion. On the contrary, if the expected rate is almost equal to the actual rate, the network is not congested and the loss is classified as due to random impairments on the wireless link.

We found that our proposed enhancement greatly improves the accuracy in loss differentiation achieved by the Vegas predictor, as we will show in Chapter 5. In the same Chapter we will also discuss the choice of the parameters α and β.

The behavior of the enhanced Vegas predictor can thus be described according to the following pseudocode:

Algorithm 4.1.1: VEGAS PREDICTOR($RTT, cwnd$)

$diff_V \leftarrow cwnd \cdot \left(1 - \frac{RTT_{min}}{RTT}\right)$

if $diff_V \leq \alpha$

then { **return** (Wireless loss)

else if $diff_V \geq \beta$

then { **return** (Congestion loss)

else if $\alpha < diff_V < \beta$

then { **return** (The network is assumed in the same state as the previous estimate)

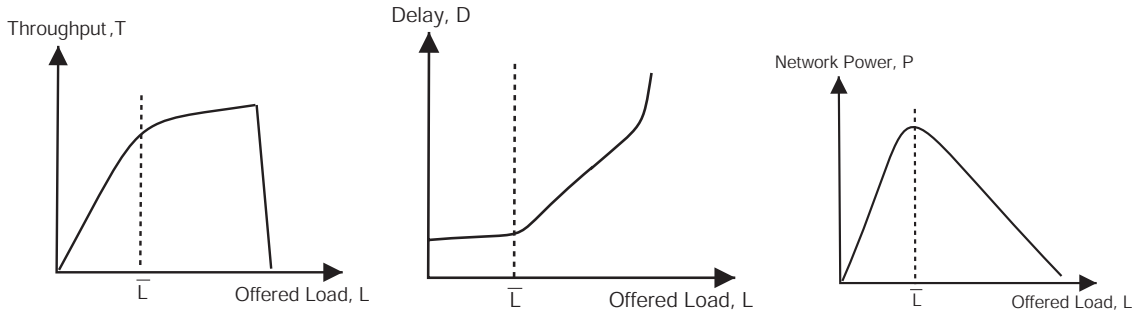
4.1.2 Enhanced Non Congestion Packet Loss Detection

A second class of LDA schemes uses delay measures to estimate the congestion status. Higher RTT values are supposed to be the effect of increased queuing delay over the network.

The Non Congestion Packet Loss Detection (NCPLD) scheme [17] estimates the cause of packet losses by trying to detect the knee-point in the load-throughput curve of the network, as defined in [44].

More specifically, when the load L offered to a network is increased, we initially observe a corresponding increase in the throughput-load curve, $T(L)$. However, when a load value $\overline{L}$ is reached, the network begins to be congested, and the gradient of the throughput-load function decreases, as shown in Figure 4.2(a); the value $\overline{L}$ for which such behavior is observed is called *knee-point*. In the same way, if we measure the delay $D(L)$ experienced in the network as a function of the offered load L, we observe that it is practically constant

for $L < \overline{L}$, and that its gradient increases significantly when L is greater than the knee-point $\overline{L}$ (see Figure 4.2(b)). Finally, if we consider the network power function, defined in [44] as $P(L) = \frac{T(L)}{D(L)}$, we observe that it is monotonically increasing for $L < \overline{L}$, decreasing for $L > \overline{L}$; hence the knee-point $\overline{L}$ represents the maximum for $P(L)$ (Figure 4.2(c)).



(a) (b) (c)

Figure 4.2: Throughput, Delay and Network Power as a function of the load offered to a network.

In NCPLD, the TCP sender estimates the total number of segments in flight over the path to the receiver ($TotalPipeSize$) as follows:

$$TotalPipeSize = \frac{1}{2} \cdot RTT_k \cdot \frac{FlightSize_k - FlightSize_{k-1}}{RTT_k - RTT_{k-1}} \tag{4.2}$$

where $FlightSize_k$ and RTT_k are respectively the flight-size and the round trip time measured at the reception of the $k-th$ acknowledgement. The NCPLD scheme has to estimate the bandwidth delay product. To this aim, as a key distinguish feature from the original NCPLD algorithm proposed in [17], we propose to use the TIBET algorithm we presented in the previous Chapter applied to the stream of received ACKs, with the purpose to obtain an accurate and stable estimation of the actual rate. These estimations are used to calculate the current value of the round trip delay at the knee-point, RTT_{kp}, calculated as:

$$RTT_{kp} = RTT_{min} + \frac{1}{2} \cdot \frac{Actual\_Rate \cdot RTT_{min}}{FlightSize_k} \tag{4.3}$$

Moreover, early simulations showed that the original NCPLD scheme [17] does not exhibit high accuracy, the main problem being the coarseness with which the TCP source measures the round trip delay. To solve this problem, we propose a modified version of the NCPLD scheme that approximates the difference between two consecutive RTT samples to the value of TCP clock granularity [29], when they assume the same value. This allows the NCPLD scheme to provide correct estimations of $TotalPipeSize$ even when consecutive samples of the round trip delay have apparently the same value.

In the following we report the pseudo-code of the proposed enhanced NCPLD algorithm, where RTT_k and $FlightSize_k$ are, respectively, the current RTT and $FlightSize$ values calculated at the receipt of th $k-th$ ACK.

Algorithm 4.1.2: NCPLD($BandwidthEstimate, RTT_k$)

if $RTT_k < RTT_{min}$

then $\{ RTT_{min} = RTT_k$

if ($FlightSize_k \neq FlightSize_{k-1}$ **and** $RTT_k \neq RTT_{k-1}$)

then
$$\begin{cases} TotalPipeSize \leftarrow \frac{1}{2} \cdot RTT_k \cdot \frac{FlightSize_k - FlightSize_{k-1}}{RTT_k - RTT_{k-1}} \\ BandwidthDelayProduct \leftarrow BandwidthEstimate \cdot RTT_{min} \\ DeltaDelay \leftarrow \frac{1}{2} \cdot RTT_k \cdot \frac{BandwidthDelayProduct}{TotalPipeSize} \\ RTT_{kp} \leftarrow RTT_{min} + DeltaDelay \end{cases}$$

if $RTT_k > RTT_{kp}$

then {**return** (Congestion loss)

else {**return** (Wireless loss)

4.1.3 Enhanced Spike Scheme

The *Spike scheme* is an algorithm originally designed for receiver hosts that use the UDP protocol [15]; recently, it has been modified in [18] to be implemented at the sender side of a TCP connection. The original Spike scheme measures the current RTT value and then computes the difference between the maximum (RTT_{max}) and the minimum (RTT_{min}) round trip time measured throughout the TCP session.

Based on these values, the Spike scheme computes two thresholds, $BspikeStart$ and $BspikeEnd$, calculated as follows:

$$\begin{aligned} BspikeStart &= RTT_{min} + \alpha \cdot (RTT_{max} - RTT_{min}) \\ BspikeEnd &= RTT_{min} + \beta \cdot (RTT_{max} - RTT_{min}) \end{aligned} \tag{4.4}$$

where typical values suggested for α and β are, respectively, 0.5 e 0.4, as suggested in [15].

The current value of the RTT is then compared to these two thresholds, to detect if the connection is experiencing congestion or not, according to the following rules:

- If the current value of the RTT is greater than $BspikeStart$, ($RTT > BspikeStart$), the Spike scheme assumes that the RTT is too high and a loss due congestion is imminent, as shown in Figure 4.3.

- If the current value of the RTT is less than $BspikeEnd$, ($RTT < BspikeEnd$), the connection is experiencing a low delay, and eventual packet losses are ascribed to random impairments on the wireless channel.

- Finally, if the RTT is comprised between $BspikeStart$ and $BspikeEnd$, ($BspikeStart \leq RTT \leq BspikeEnd$), the Spike scheme assumes that the network state is not changed from the previous estimate.

After a preliminary analysis, we found that the performance of the Spike scheme can be enhanced consistently by resetting the value of RTT_{max} after the first expiration of a retransmission timeout. By doing so, we avoid to use a biased value for RTT_{max} throughout the connection's lifetime. In fact, during the first slow-start phase, the TCP sender can inject into the network a consistent amount of traffic, and this behavior is evident when the network capacity is high or the network load is low. The value of RTT_{max} estimated in this phase is often too high to represent an estimate of the maximum delay that will be experienced by the connection in its steady-state condition. More precisely, we found that without our proposed enhancement, the value of RTT_{max} coincides with

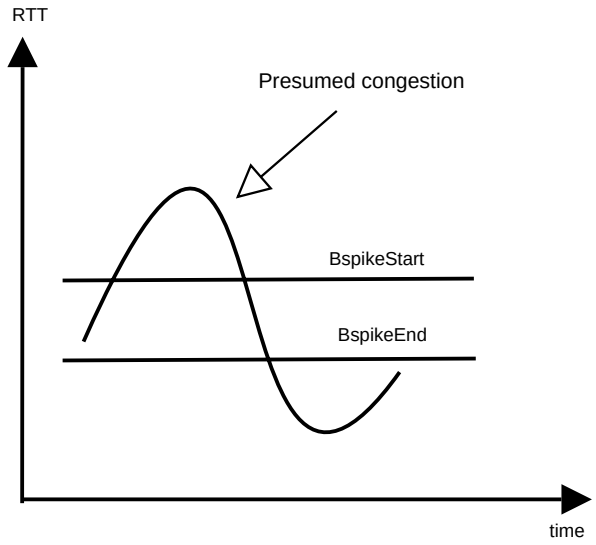


Figure 4.3: Heuristics used by the Spike scheme to estimate network state based on measures of the RTT

the RTT value measured during the first slow-start phase of the connection, and that the thresholds $BspikeStart$ and $BspikeEnd$ assume values that are much higher than those assumed by RTT during the steady-state phase of the connection. This phenomenon causes the Spike scheme to classify every loss as due to transmission errors, limiting its performance in heterogeneous networks.

The pseudo-code of the Spike scheme enhanced with the technique presented above is reported in the following:

Algorithm 4.1.3: SPIKE SCHEME(RTT)

if (First Retransmission Timeout Expires)
  then $\{ RTT_{max} \leftarrow 0$
if $RTT < RTT_{min}$
  then $\{ RTT_{min} \leftarrow RTT$
  else if $RTT > RTT_{max}$
  then $\{ RTT_{max} \leftarrow RTT$
$BspikeStart \leftarrow RTT_{min} + \alpha \cdot (RTT_{max} - RTT_{min})$
$BspikeEnd \leftarrow RTT_{min} + \beta \cdot (RTT_{max} - RTT_{min})$
if $RTT > BspikeStart$
  then $\{$**return** (Congestion loss)
  else if $RTT < BspikeEnd$
  then $\{$**return** (Wireless loss)
  else $\{$**return** (The network is assumed in the same state as the previous estimate)

4.1.4 Flip Flop Scheme

The *Flip Flop* scheme, proposed in [16], measures RTT samples to distinguish between congestion and random losses. In this scheme, using an adaptive Flip Flop filter [45], a parallel estimation of RTT is done on every new ACK received in NewReno. TCP usually uses an exponentially weighted moving average (EWMA) filter which is static. The Flip Flop filter uses two EWMA filters, one stable and the other agile. An agile filter is one which gives more weight to the most recently observed sample x_k unlike a stable filter that gives more weight to the current estimated value y_k. The two filters used in the Flip Flop scheme are reported in the following:

$$\begin{aligned} y_k^{agile} &= 1/10 \cdot y_{k-1}^{agile} + 9/10 \cdot x_k \\ y_k^{stable} &= 9/10 \cdot y_{k-1}^{stable} + 1/10 \cdot x_k \end{aligned} \tag{4.5}$$

The underlying principle is to employ an agile filter whenever possible but switch to

the stable one when the RTT samples vary drastically and become noisy. According to statistical quality control, control limits are defined around the current sample mean and when the samples exceed the control limits, the process is said to be out of control. To estimate the deviation, the filter uses a moving range which it estimates from the samples within the control limits. The control limit is defined as:

$$\overline{x} + 3 \cdot \frac{\overline{MR}}{d_2} \tag{4.6}$$

where $\overline{x}$ is the sample mean, $\overline{MR}$ is the Moving Range which is the average of the differences between adjacent RTT samples, $|x_i - x_{i-1}|$, and d_2 estimates the standard deviation of a given sample given its range. When the range is from a sample of two, as for MR, $d_2 \approx 1.128$ [46]. The basic tenet of this approach is that if the packets are suffering congestion losses, the observed RTTs will vary, while if packets are suffering random losses, the observed RTTs will not vary much. Using the Flip Flop filter, we define an upper control limit on RTT using equation (4.6). We then consider the much delayed packets, whose RTT exceeds the control limit, as *outliers*. More than η outliers in the last l samples are used as congestion indication, where η and l are tunable parameters.

Whenever an ACK is received, the Flip Flop scheme uses the current measured value of the RTT, s_rtt, to estimate its average est_rtt and its standard deviation $\overline{MR}$. To memorize the type of filtering used at the reception of the last l ACKs, a shift register of length l is used; whenever est_rtt is computed, the content of the register is shifted to the left and the least significant bit is set to 1 if the current RTT sample is an outlier. The number of bits whose value is set to 1 in the register represents the number of outliers that occurred in the last l ACKs: if this value is greater than η, the Flip Flop scheme assumes that the network is congested; otherwise, eventual packet losses are ascribed to transmission errors.

The pseudo-code of the Flip Flop scheme is reported in the following:

Algorithm 4.1.4: FLIP FLOP SCHEME(RTT, L, η)

Comment: "«" is the bitwise left shift operator

if $s\_rtt > est\_rtt + 3 \cdot \frac{\overline{MR}}{1.128}$

then
$$\begin{cases} vector \leftarrow vector << 1 \\ vector \leftarrow vector \textbf{ or } 1 \\ est\_rtt \leftarrow 9/10 \cdot est\_rtt + 1/10 \cdot s\_rtt \end{cases}$$

else
$$\begin{cases} vector \leftarrow vector << 1 \\ vector \leftarrow vector \textbf{ or } 0 \\ est\_rtt \leftarrow 1/10 \cdot est\_rtt + 9/10 \cdot s\_rtt \\ diff \leftarrow |s\_rtt - last\_rtt| \\ \overline{MR} \leftarrow 7/8 \cdot \overline{MR} + 1/8 \cdot diff \end{cases}$$

if First ACK is received (Initialization Phase)

then
$$\begin{cases} vector = 0 \\ est\_rtt \leftarrow s\_rtt \\ \overline{MR} \leftarrow \frac{est\_rtt}{2} \end{cases}$$

else $\{last\_rtt \leftarrow s\_rtt$

4.1.5 Constant Loss Differentiation Algorithms

To assess the accuracy of the Loss Differentiation algorithms presented above, establishing upper and lower bounds, we implemented the following additional schemes:

- Ideal-LDA: it knows the exact cause of last packet loss, yielding an upper bound to the accuracy.
- Constant-LDA: it produces a constant estimated state, no matter what the input conditions are. Therefore, two constant LDA schemes can be devised: the *Always Congested* and the *Always Wireless* scheme, the former assuming all losses are due to buffer overflow, the latter ascribing all losses to transmission errors over the wireless path. Note that the standard TCP version (TCP NewReno) currently implemented

in the Internet assumes that every loss is due to congestion, and can be therefore thought as equipped by an *Always Congested* predictor.

- Random-LDA: it produces an estimate of the network as a random result. As there is no a priori information about the distribution of congestion and wireless events, we set equiprobable decisions on output. This scheme yields a lower bound to the accuracy.

4.1.6 Implementation of Loss Differentiation Algorithms

The LDA schemes presented above have been implemented modifying the source code of the agent *Agent/TCP/Newreno* in Network Simulator (*ns* version 2). As introduced previously, network state has been modeled using a binary classification {congestion, not-congestion}, stored in a variable called *LDA_netstate_* implemented as a member of the class *NewrenoTcpAgent*. Within this class we implemented the function *void estimate_network(Packet*pkt)*, that updates the value of *LDA_netstate_* based on TCP state variables and estimates; this function is called whenever the TCP source receives an ACK. Figure 4.4 shows the structure of the modified TCP agent.

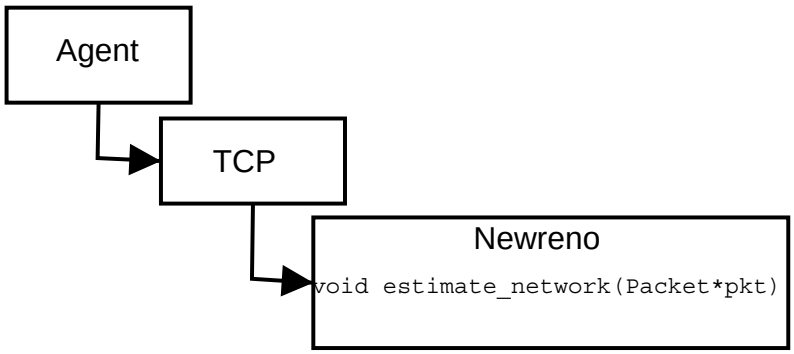


Figure 4.4: Modified TCP Source implementing Loss Differentiation.

4.2 TCP NewReno-LP: Enhanced Error Recovery based on Loss Prediction

Classical TCP implementations classify a segment as lost when 3 duplicate ACKs (DUPACK) are received or when a retransmission timeout expires (RTO expiration); then, the error recovery procedures described in Chapter 2 diminish the transmission rate by decreasing both the *congestion window* and the *slow start threshold*.

To improve TCP performance in heterogeneous networks, we propose to enhance TCP's error recovery mechanism using the loss differentiation schemes presented above. When a packet loss is caused by network congestion it is necessary to reduce TCP's transmission rate; on the other hand, when a packet loss is classified as due to random impairments on wireless links, the TCP source should not reduce its transmission rate.

Hence, we designed a novel error recovery scheme for TCP that aims at achieving both high goodput gain in the presence of wireless losses and fairness with concurrent TCP flows in the presence of network congestion.

The new TCP source that extends TCP NewReno using a Loss Predictor (LP) in its error recovery scheme is hence called TCP NewReno-LP.

4.2.1 TCP NewReno-LP: Algorithm

The accuracy of the Loss Predictor used to enhance TCP's error recovery mechanism is fundamental: when packet random error rate is low and most of the packet losses are due to congestion, LDA accuracy in ascribing losses is necessary to achieve fairness with concurrent TCP flows. On the other hand, when packet random error rate is high such as in wireless links, LP accuracy is necessary in order to achieve goodput gain. We will measure the accuracy of the proposed LP schemes in Chapter 5.

If the loss is classified as due to congestion, the TCP source reacts exactly as a classical TCP NewReno source, setting the slow start threshold ($ssthresh$) to half the current flight size. This allows TCP NewReno-LP to behave as fairly as the standard TCP protocol in congested network environments.

On the contrary, if the loss is classified as due to random bit corruption on the wireless channel, the *ssthresh* is first updated to the current flight size value.

Then, if the packet loss has been detected by the TCP source after the receipt of 3 duplicate ACKs, the TCP sender updates the *cwnd* to $ssthresh + 3$ Maximum Segment Sizes (MSS) and enters the fast retransmit phase as the standard TCP NewReno. This allows the source to achieve higher transmission rates upon the occurrence of wireless losses, if compared to the blind halving of the transmission rate performed by current TCP implementations.

If the packet loss has been detected by the TCP source after a retransmission timeout expiration, the congestion window is reset to 1 segment, thus enforcing a friendly behavior of the TCP source toward current TCP implementations.

In the following, we report the pseudo-code of the error recovery mechanism implemented in TCP NewReno-LP:

Algorithm 4.2.1: TCP NewReno-LP(*ErrorRecovery*)

if (received DUPACK > 2) **and** (TCP is not in Fast Recovery)

then
- **if** (LDA estimated a non-congested network state)
  - **then**
    - $ssthresh \leftarrow FlightSize$
    - $cwnd \leftarrow ssthresh$
  - **else**
    - $ssthresh \leftarrow min\{2 \cdot MSS, FlightSize/2\}$
    - $cwnd \leftarrow ssthresh$
- Retransmit the first unacknowledged segment

if (Retransmission Timeout expires)

then
- **if** (LDA estimates a non-congested network state)
  - **then**
    - $ssthresh \leftarrow FlightSize$
    - $cwnd \leftarrow 1 \cdot MSS$
  - **else**
    - $ssthresh \leftarrow min\{2 \cdot MSS, FlightSize/2\}$
    - $cwnd \leftarrow 1 \cdot MSS$
- Retransmit the first unacknowledged segment

To illustrate the behavior of TCP NewReno-LP, we consider a single connection transmitting over a 10Mb/s wireless link affected by random losses. The connection has a RTT equal to 100ms. Figure 4.5 shows the evolution of the congestion window and the slow start threshold of the TCP connection in this scenario.

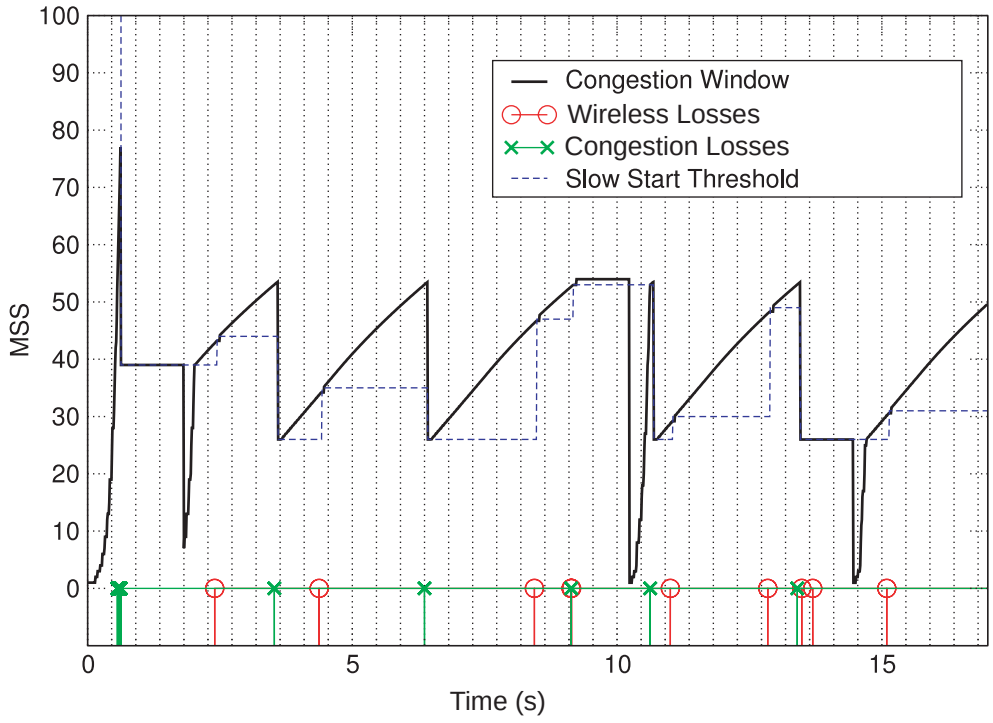


Figure 4.5: Evolution of *Congestion Window* and *Slow Start Threshold* of a TCP NewReno-LP connection over a 10 Mb/s link

4.2.2 TCP NewReno-LP: Implementation

The TCP NewReno-LP source has been implemented as an extension of the TCP NewReno agent; the enhanced error recovery mechanism described in the previous Section has been implemented in the procedures *dupack_action()* and *timeout(int tno)*. To estimate the nature of packet losses, the TCP NewReno-LP source uses the procedure *estimate_network(Packet * pkt)* already described in Section 4.1.6. The overall scheme of a TCP NewReno-LP source is depicted in Figure 4.6.

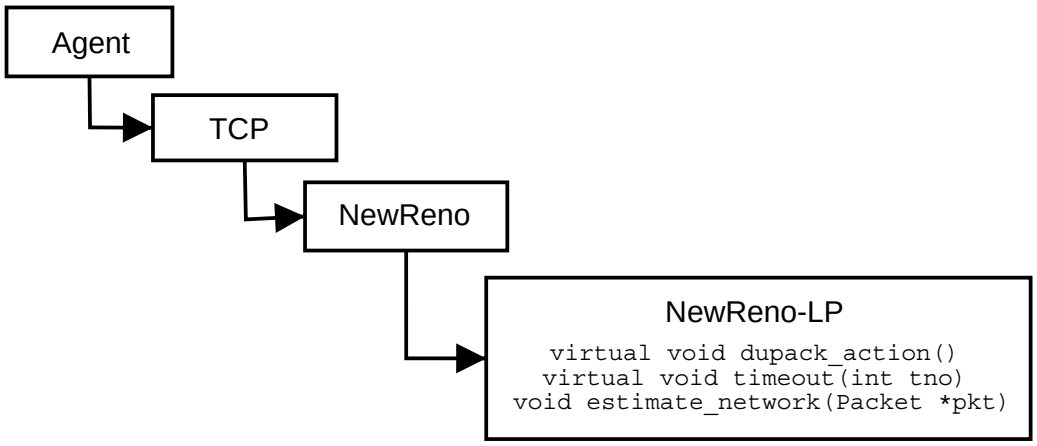


Figure 4.6: TCP NewReno-LP implementation in ns-2.

Chapter 5

Accuracy of Loss Differentiation Algorithms

The key feature for Loss Differentiation Algorithms is to be accurate in ascribing the cause of packet losses, as TCP error recovery based on loss differentiation reacts gently or aggressively depending on the LDA decision. To summarize, when packet random error rate is low and most of the packet losses are due to congestion, LDA accuracy in ascribing losses is necessary to achieve fairness with concurrent TCP flows. On the other hand, when packet random error rate is high such as in wireless links, LDA accuracy is necessary in order to achieve throughput gain.

In this Chapter we measure and compare the accuracy of the Loss Differentiation Algorithms we proposed previously. For this goal we define a novel metric for the accuracy, i.e. the ratio between the number of correct classifications and the total number of loss events, as detailed in the following. For every LDA scheme, we show the accuracy achieved in various network scenarios varying the packet error rate on the wireless link, the error model and the connections' round trip time. We also provide an upper bound on the performance of LDA schemes, ideally assuming perfect knowledge of the cause of packet losses. It is then shown that our proposed enhanced Vegas scheme approaches reasonably this bound. Based on these results, we propose to use the Vegas loss predictor to enhance the TCP NewReno error-recovery scheme.

5.1 Simulation Scenario

Simulations were carried out using the Network Simulator package (ns v.2 [37]). The accuracy of the LDA schemes outlined in the previous Chapter has been evaluated in several network scenarios. The simplest network topology, named *Single Link*, is described in Figure 5.1: a single TCP-NewReno source S, provided with LDA estimator, performs a bulk FTP transfer. The wired link, $N1 - N2$, has capacity and propagation delay set to $C = 10$ Mbit/s and $\tau = 50$ ms. The Maximum Segment Size (MSS) is set to 1500 bytes. All queues can store a number of packets equal to the bandwidth-delay product of the connection. The parameters of the wireless link $N2 - D$ are set to $C = 10$ Mbit/s and $\tau = 0.01$ ms.

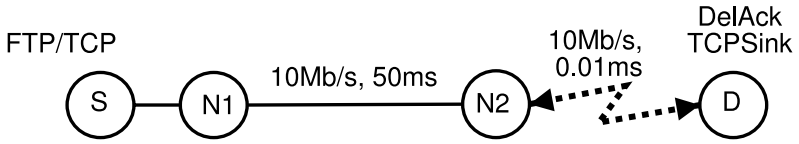


Figure 5.1: *Single Link* topology.

In our simulations, the wireless link may drop packets according to two different error models, without and with time-correlation. In the former case, packets are dropped independently, with Packet Error Rate (PER) ranging from 10^{-5} to 10^{-1}. In the latter case, the link $N2 - D$ was modeled, according to [43], with the two-state Markov chain described in Figure 5.2, where the channel is either in the Good or in the Bad state. In the *Good* state no packet loss occurs, while, in the *Bad* state, the packet error rate varies from 0 to 100% to take into account various levels of fading. The durations of the Good and Bad states are both exponentially distributed with average respectively equal to $T_{gg} = 1$ s and $T_{bb} = 50$ ms, in accordance with [43].

In the topology named *Congested Dumbbell*, commonly considered in the literature and shown in Figure 5.3, we simulate a simple hybrid environment: a single TCP source provided with LDA shares the bottleneck link $N1 - N2$, whose bandwidth is $C = 10$ Mbit/s and delay $\tau = 50$ ms, with 30 UDP sources having the same priority as the TCP

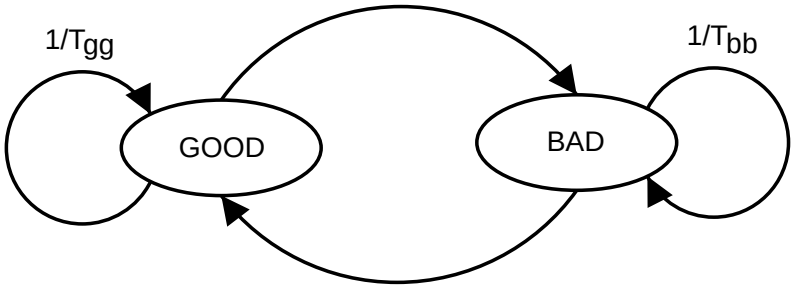


Figure 5.2: Markov Chain that models a channel affected by correlated losses.

source. Each UDP source switches between ON and OFF periods, whose lengths are both Pareto-distributed with shape parameter equal to 1.5 and mean set to 100 ms and 200 ms, respectively.

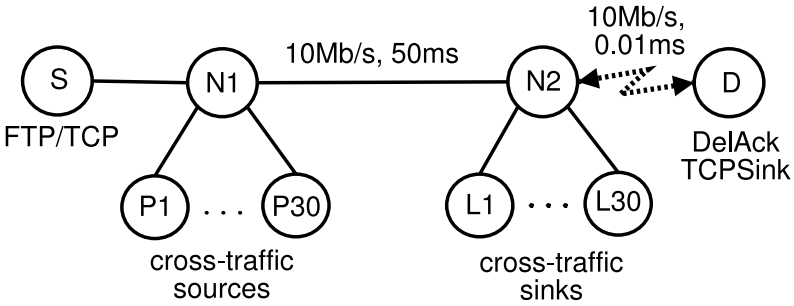


Figure 5.3: *Congested Dumbbell* topology.

During the ON period, each source transmits packets with 1500 byte size at constant bit rate equal to 0.5 Mbit/s. While in OFF period, the UDP sources don't transmit any packet. Such cross-traffic configuration leaves to the TCP source an available bandwidth that varies randomly during the simulation, with average equal to half the bottleneck capacity. The wireless link $N2 - D$ is affected by random losses according to the models described in the *Single-Link* topology.

Finally, in the *MultiHop* topology shown in Figure 5.4 a more complex scenario is set up. Four TCP sources enhanced with LDA access a multi-hop network from different entry nodes. The background traffic is modeled as described above, with 30 UDP sources

transmitting at a peak rate of 200 kb/s. Also in this case, the last hop of each TCP connection is affected by random transmission errors as for the Single Link topology.

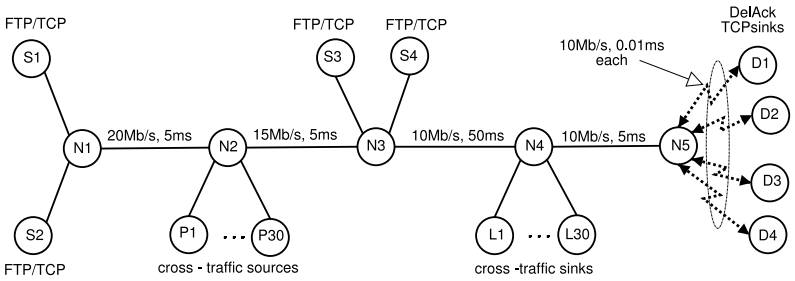


Figure 5.4: *MultiHop* topology.

5.2 Performance Metrics

Let LE be the number of loss events detected by the TCP source based on the reception of triple duplicate acknowledgements or retransmission timeout expirations. We define *wireless loss* (WL) a packet loss caused by the wireless noisy channel. Instead, a *congestion loss* (CL) is defined as a packet loss caused by network congestion. Let CL_D and WL_D be, respectively, the number of congestion and wireless losses correctly classified by the LDA scheme under investigation.

The accuracy A_C that a LDA scheme has in classifying losses due to network congestion is defined as the ratio between the number of such losses correctly detected and the total number of congestion losses:

$$A_C = \frac{CL_D}{CL} \tag{5.1}$$

In the same way, the accuracy A_W that a LDA scheme has in classifying wireless losses is defined as the ratio between the number of such losses correctly detected and the total number of wireless losses:

$$A_W = \frac{WL_D}{WL} \tag{5.2}$$

We propose to define the overall accuracy A_L of a LDA scheme as the ratio between the number of correct classifications and the total number of loss events:

$$A_L = \frac{CL_D + WL_D}{LE} \tag{5.3}$$

With such definition, A_L can be interpreted as a weighted average of A_C and A_W, with weights $r_C = \frac{CL}{LE}$ and $r_W = \frac{WL}{LE}$ respectively:

$$A_L = \sum_{i \in \{C,W\}} A_i \cdot r_i = A_C \cdot r_C + A_W \cdot r_W = \frac{CL_D + WL_D}{LE} \tag{5.4}$$

5.3 Accuracy of LDA Schemes

To evaluate the performance of the LDA schemes presented in the previous Chapter, we measured their accuracy embedding them in a classical TCP NewReno source that does not use this estimate within its error recovery scheme.

With the extensive simulation analysis we have performed, however, we have observed that TCP sources implementing error recovery mechanisms based on LDA estimates, like TCP NewReno-LP, achieve almost the same level of accuracy of such source in all the considered scenarios, as we will show in the next Chapter. This result is important as it allows to evaluate the accuracy of LDA schemes independently from TCP implementations and variations: once a LDA has proved sufficiently accurate, it can then be used to enhance a TCP source guaranteeing practically the same level of accuracy in loss differentiation.

We have considered various network scenarios with different patterns of packet losses. In the following, we will first consider the case in which all the losses are uncorrelated. Then, we will analyze the case in which losses are correlated and modeled according to the Markov chain described previously. Finally, we will show the impact of the Round Trip Time of the connection on the accuracy of LDA schemes.

5.3.1 Uncorrelated Losses

The accuracy of the LDA schemes analyzed in this work has been measured for different settings of the configuration parameters. Their performance is compared to that achieved by constant LDAs, that provide upper and lower bounds as well as the distribution of loss events in the considered network topologies as a function of the packet error rate (PER).

Accuracy of Constant LDAs

To measure the distribution of loss events as a function of the network topology, PER and Round Trip Time of the connection, we use the accuracy of the *Always Congested* and *Always Wireless* schemes. As such schemes always provide a constant estimate, it is correct to say that they classify perfectly loss events of one kind, while they have accuracy equal to zero in the classification of events of the other kind.

More specifically, the *Always Congested* scheme classifies correctly all congestion events, while the *Always Wireless* scheme has a 100% accuracy on classifying wireless losses: hence $A_C = 1$ and $A_W = 0$ for the *Always Congested* scheme; in the same way, $A_C = 0$ and $A_W = 1$ for the *Always Wireless* scheme. Finally, equation 5.4, applied to these 2 constant LDAs allows to calculate the values of r_C and r_W.

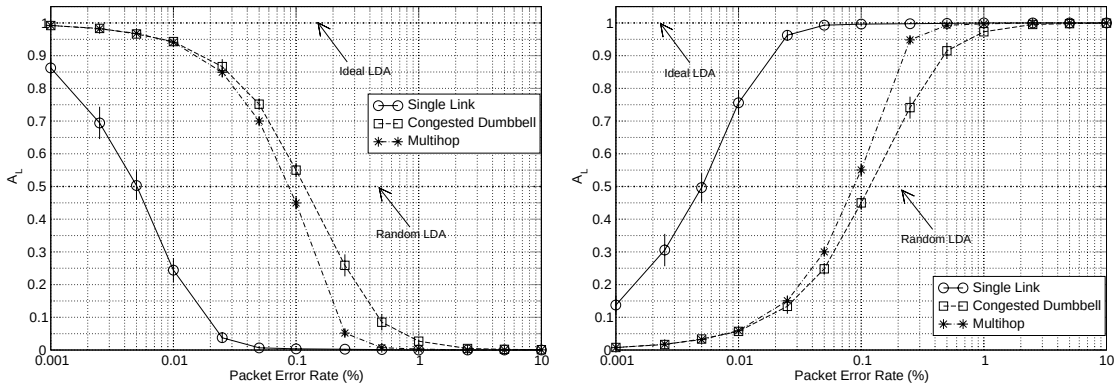
More precisely, indicating with A_{AC} and A_{AW} the accuracy of the *Always Congested* and *Always Wireless* schemes, respectively, we have:

$$A_{AC} = \frac{CL}{LE} \tag{5.5}$$

and in the same way, for the *Always Wireless* scheme:

$$A_{AW} = \frac{WL}{LE} \tag{5.6}$$

Hence, the accuracies of such constant LDAs coincide with the ratio between congestion (wireless) losses and total loss events: $A_{AC} = r_C$ and $A_{AW} = r_W$. Such measures are reported in Figure 5.5 for all the network topologies considered, together with the accuracy achieved by the *Ideal* and *Random* LDAs.



(a) Always Congested

(b) Always Wireless

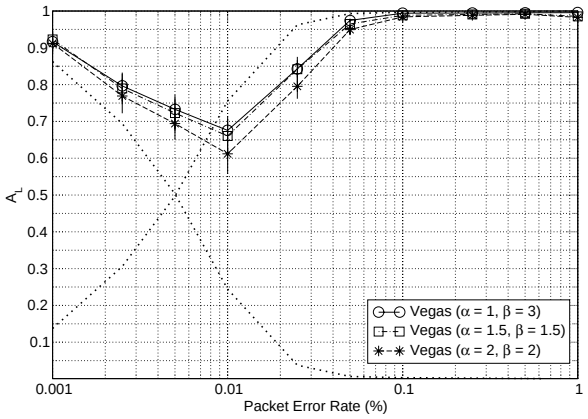
Figure 5.5: Accuracy of the *Always Congested* and *Always Wireless* LDAs as a function of the PER in the considered topologies.

We note that, as the PER increases, the accuracy of the *Always Congested* scheme decreases; this result is in line with the observation that classical TCP sources are not capable of achieving high throughput over links affected by significant packet losses. Hence, the majority of such losses is due to random impairments on the wireless link.

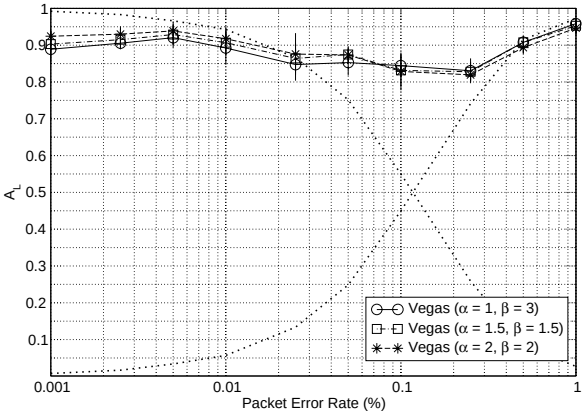
Enhanced Vegas Loss Predictor

As discussed in the previous Chapter, the Vegas predictor classifies packet losses as due to congestion or transmission errors based on two thresholds, α and β. Hence, we gauged the sensibility of the Vegas Predictor measuring its performance for different settings of these two parameters, and the results are shown in Figure 5.6.

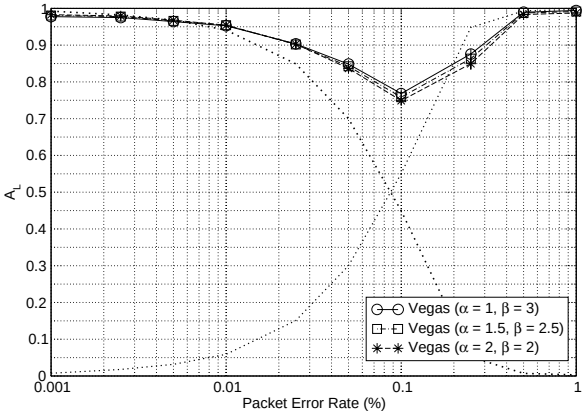
Figures 5.6(a), 5.6(b) and 5.6(c) show the accuracy of the Vegas predictor for different values of α and β, in the 3 topologies described above, as a function of the PER on the wireless link. We also performed an extensive simulation campaign, with more complex network topologies involving multiple hops, and the best performance was obtained for $\alpha = 1$ and $\beta = 3$. However, we note that the Vegas predictor is less sensible to the parameters tuning than the other schemes considered in this work; moreover, this predictor achieves good accuracy in discriminating both congestion and wireless losses.



(a) Single Link



(b) Congested Dumbbell



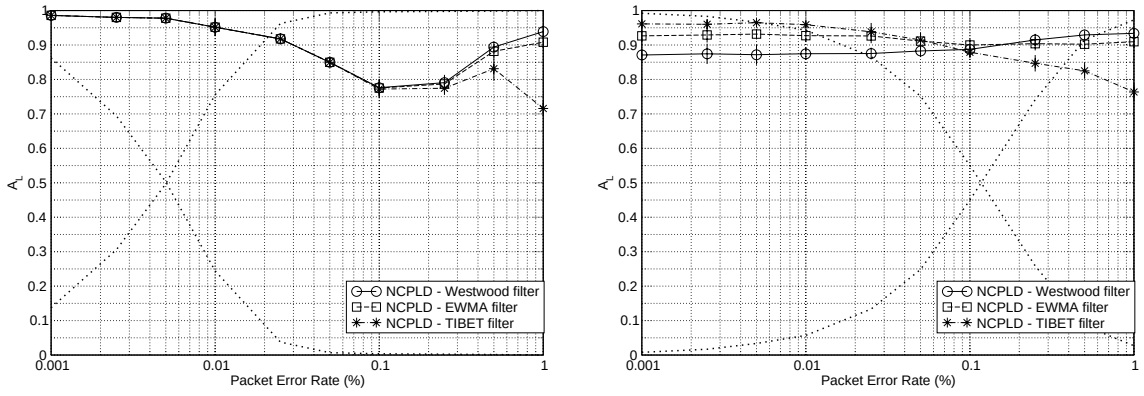
(c) MultiHop

Figure 5.6: Accuracy of the enhanced *Vegas* predictor for different settings of α and β, as a function of the PER, in the *Single Link*, *Congested Dumbbell* and *MultiHop* topologies. The dotted lines represent the accuracy achieved by the Always Congested and Always Wireless schemes.

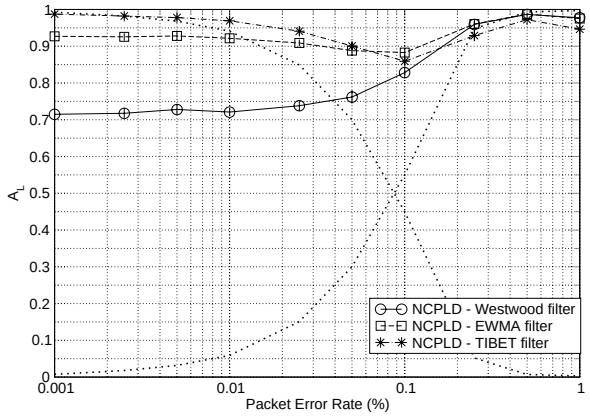
Enhanced NCPLD

The accuracy of the NCPLD scheme enhanced with the use of TIBET has been measured in the 3 network topologies considered in this chapter.

Figures 5.7(a), 5.7(b) and 5.7(c) show the accuracy achieved by the enhanced NCPLD scheme as a function of the packet error rate on the wireless link. To underline the impact of the bandwidth estimator embedded in NCPLD on the accuracy of such scheme, we considered various estimation algorithms. We note that TIBET allows NCPLD to achieve



(a) Single Link (b) Congested Dumbbell



(c) MultiHop

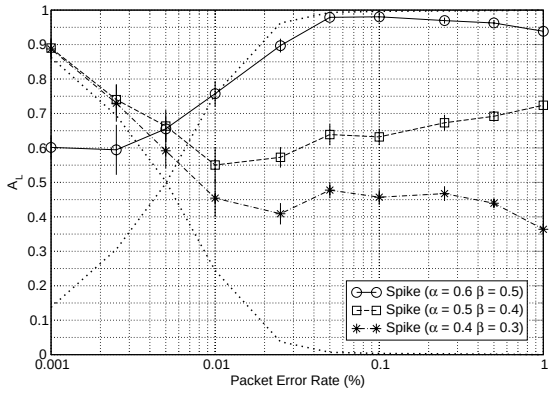
Figure 5.7: Accuracy of the enhanced *NCPLD* scheme as a function of the PER for different bandwidth estimators in the *Single Link*, *Congested Dumbbell* and *MultiHop* topologies. The dotted lines represent the accuracy achieved by the Always Congested and Always Wireless schemes.

the best performance, as it performs an efficient bandwidth estimate in all the considered network topologies. On the contrary, the Westwood filter or a simple Exponentially Weighted Moving Average (EWMA) scheme achieve worse performance, thus diminishing the accuracy of NPLD in classifying congestion losses.

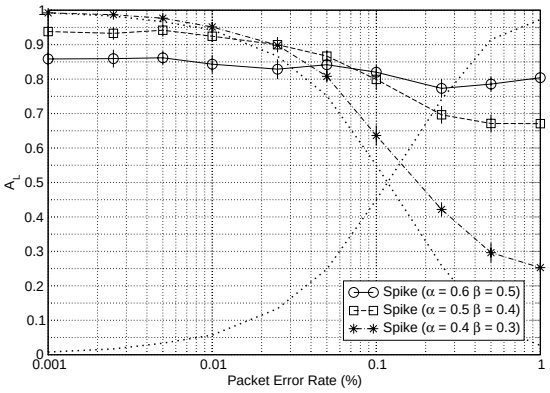
Enhanced Spike Scheme

The Spike scheme enhanced as described in the previous Chapter has been tested with different settings of the parameters α and β, to measure the sensibility of its accuracy to

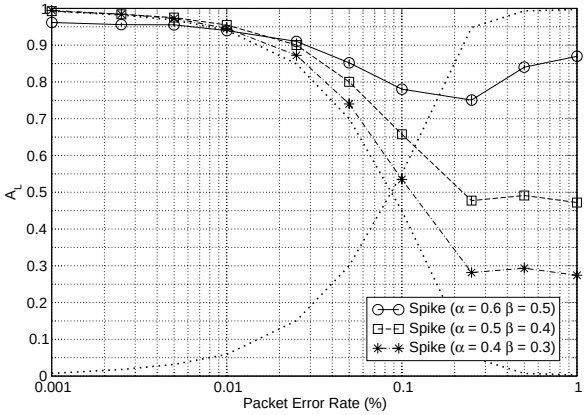
such settings. Figures 5.8(a) and 5.8(b) show the accuracy of the Spike scheme in the topologies *Single Link* and *Congested Dumbbell* as a function of the PER in the presence of uncorrelated losses.



(a) Single Link



(b) Congested Dumbbell



(c) MultiHop

Figure 5.8: Accuracy of the enhanced Spike scheme for different values of α and β as a function of the PER on the wireless link in the *Single Link*, *Congested Dumbbell* and *MultiHop* topologies. The dotted lines represent the accuracy achieved by the Always Congested and Always Wireless schemes.

It can be observed that the accuracy of the Spike scheme is quite sensible to the settings of α and β; more in detail, when α and β are less than 0.5, the Spike scheme is very accurate in classifying congestion losses. On the other hand, when such parameters are greater than 0.5, the Spike scheme is more accurate in classifying losses due to transmission errors on the wireless link.

Similar results have been obtained in the *MultiHop* topology, and are shown in Figure 5.8(c).

Flip Flop Scheme

The Flip Flop scheme has been configured to take into account only the last $L = 8$ filterings performed on incoming ACKs, as suggested in [16].

The accuracy of this scheme has been evaluated for $\eta = 2$, 4 and 6, to test its sensibility to the parameters setting. In [16] the authors affirm that the Flip Flop scheme achieves a good accuracy in classifying congestion losses when η assumes low values; this result has been observed in our simulations for the considered values of {L, η}.

However, we observed that when η approaches L, the Flip Flop scheme is more accurate in discriminating wireless losses, but less accurate in individuating network congestion. Figures 5.9(a), 5.9(b) and 5.9(c) show the accuracy of the Flip Flop scheme in the *Single Link*, *Congested Dumbbell* and *MultiHop* topologies, respectively, for the considered parameters settings.

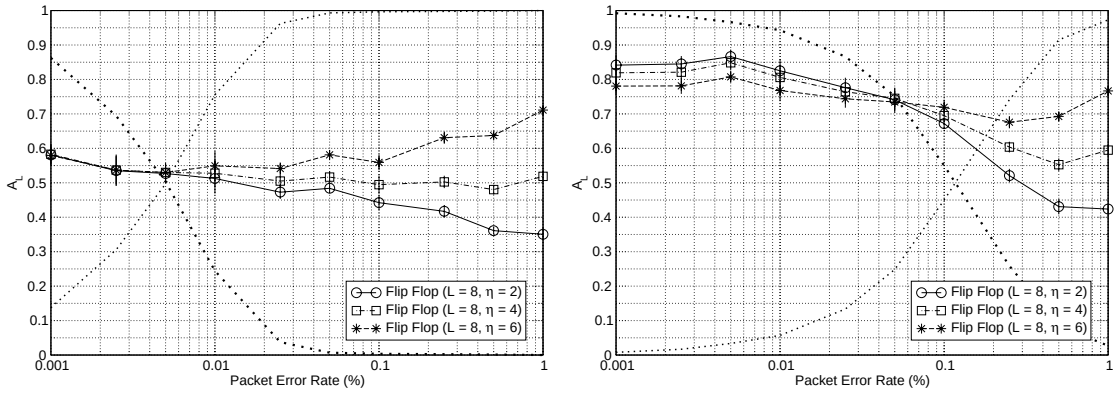
Note that the best performance has been observed for$L = 8$ and $\eta = 6$; this is substantially in accordance with [16], thus backing up the results obtained in our simulations.

5.3.2 High Packet Error Rates

When the packet error rate is high, it is likely that multiple packet losses occur in the same window of data. In this case, the fast recovery procedure cannot prevent the expiration of retransmission timeouts [27], and usually the congestion window assumes low values for the whole connection's lifetime.

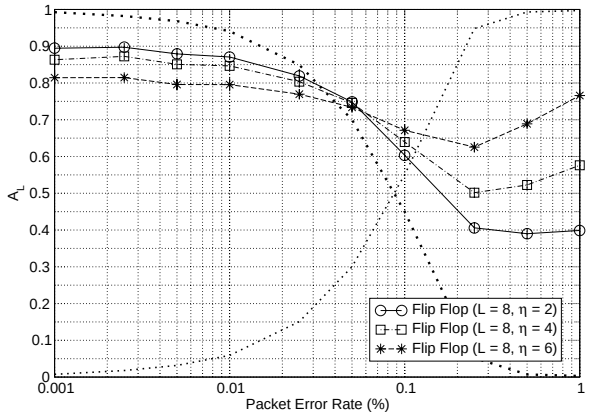
We observed that even in this situation the accuracy of the enhanced loss differentiation schemes presented in the previous Chapter is still satisfactory, with the exception of the NCPLD scheme. This scheme, in fact, is quite affected by such network conditions, as it turns out that the values estimated for *TotalPipeSize* are quite different from the total number of segments in flight over the path to the receiver, thus worsening its performance.

Figures 5.10(a), 5.10(b) and 5.10(c) compare the accuracy of the LDA schemes pre-



(a) Single Link

(b) Congested Dumbbell



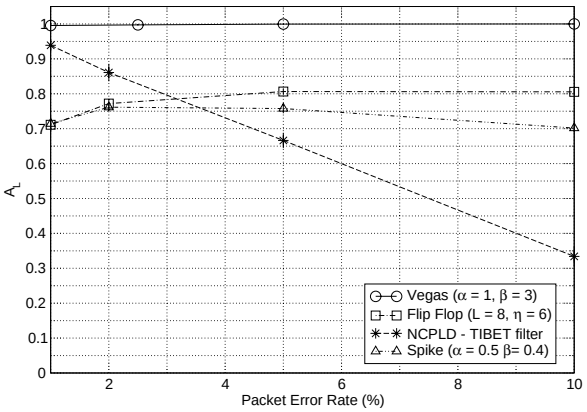
(c) MultiHop

Figure 5.9: Accuracy of the *Flip Flop* scheme for different settings of the parameters L and η as a function of the PER on the wireless link in the *Single Link*, *Congested Dumbbell* and *MultiHop* topologies. The dotted lines represent the accuracy achieved by the Always Congested and Always Wireless schemes.

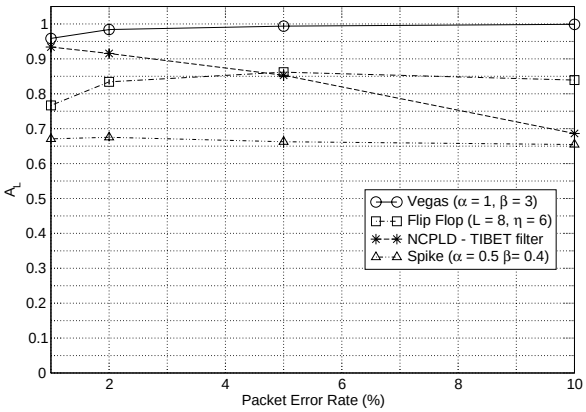
sented before in the three considered topologies as a function of the PER, with $PER > 1\%$.

We notice that the Vegas predictor, the Spike and the Flip Flop schemes achieve an accuracy that is practically independent of the PER, while that of the NCPLD scheme diminishes for increasing error rates.

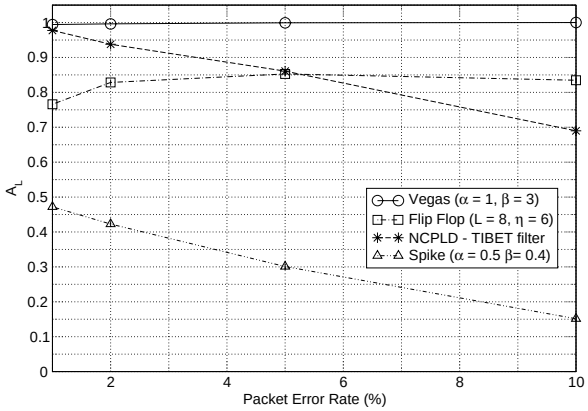
Moreover, the Vegas predictor always achieves a very high accuracy, practically coincident with that of an Ideal scheme. Based on this observation we can affirm that the Vegas predictor is the best suited even in these demanding network conditions.



(a) Single Link



(b) Congested Dumbbell



(c) MultiHop

Figure 5.10: Accuracy of the Vegas, NCPLD, Spike, and Flip Flop schemes with high PER values in the *Single Link*, *Congested Dumbbell* and *MultiHop* topologies.

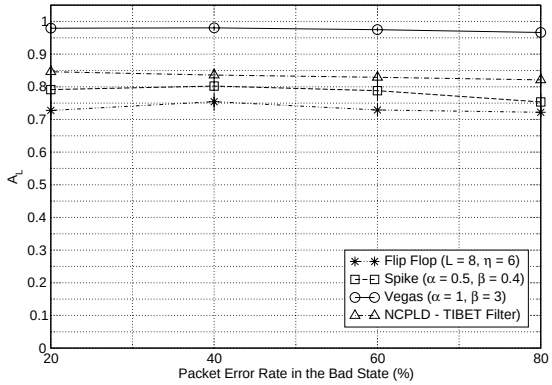
5.3.3 Correlated Losses

We measured the accuracy of the LDA schemes in the presence of correlated errors on the wireless link, modeled according to the Markov model depicted in Figure 5.2. To simulate various levels of fading, we varied the error rate in the *Bad* state, while in the *Good* state no packet loss occurs, in accordance with [47].

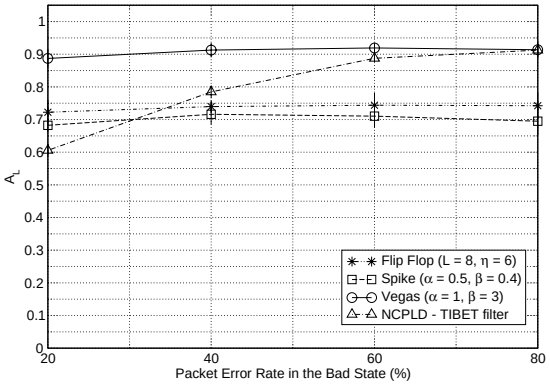
We observed that, in the presence of correlated errors, the accuracy of all the LDA schemes increases with respect to the case in which errors are uncorrelated. Note that with this error model, it is likely that multiple losses occur in the same window of transmitted data, thus causing the expiration of retransmission timeouts.

Hence, when packet losses are correlated, LDA schemes can contribute with their high accuracy to improve the performance of TCP connections exploiting loss differentiation.

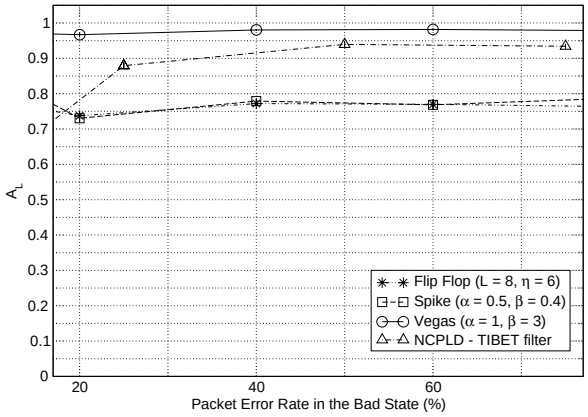
Figures 5.11(a), 5.11(b) and 5.11(c) show the accuracy achieved by the considered LDA schemes as a function of the packet error rate in the *Bad* state in the *Single Link*, *Congested Dumbbell* and *MultiHop* topologies, respectively.



(a) Single Link



(b) Congested Dumbbell



(c) MultiHop

Figure 5.11: Accuracy of the NCPLD, Spike, Flip Flop and Vegas schemes with correlated losses on the wireless link in the *Single Link*, *Congested Dumbbell* and *MultiHop* topologies.

We notice that the Vegas predictor always achieves a greater accuracy than all the other schemes. More specifically, in all the considered scenarios, the Vegas predictor always achieved an accuracy greater than 90% (see Figure 5.11(b)); however, its accuracy practically overlaps that of an Ideal predictor in all the different scenarios. As correlated losses are likely to occur in real wireless networks, we believe that these results show how

the Vegas predictor can be particularly appealing for such networks.

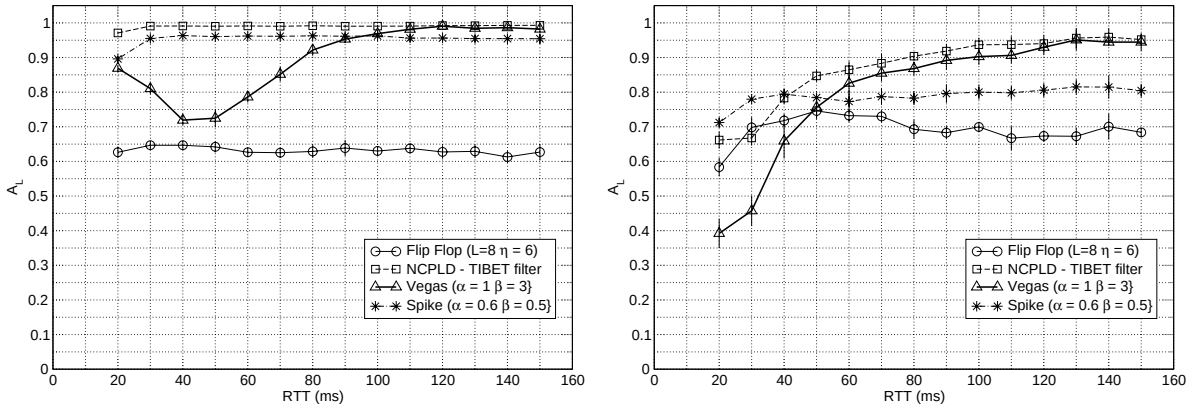
5.3.4 Impact of the Round Trip Time

The accuracy of the LDA schemes has been evaluated in the two network topologies considered above, varying the propagation delay on the wired link traversed by the TCP connection, thus increasing its RTT.

The accuracy achieved by the Spike, Flip Flop and NCPLD schemes is practically independent from the RTT. On the contrary, we observed that the Vegas Predictor achieves good performance when the bandwidth-delay product is sufficiently high. When the bandwidth-delay product is low, instead, we found that the value of $diff_V$ is very frequently comprised between α and β. Hence, as in this case the Vegas scheme assumes the network is in the same situation of the preceding estimate, it is likely that the loss prediction refers to a network condition quite far in time and uncorrelated with the current one.

However, we observe that when the bandwidth-delay product of the connection is low, there is no much room left for performance improvement, as a congestion window of few segments suffices to transmit continuously, exploiting all the available bandwidth.

Figures 5.12(a) and 5.12(b) show the accuracy achieved by the four LDAs in the *Single Link* and *Congested Dumbbell* topologies as a function of the RTT, with a wireless link affected by uncorrelated losses ($PER = 0.5\%$). Notice that all the schemes achieve an accuracy in packet loss classification greater than both the Constant and Random LDAs.



(a) Single Link

(b) Congested Dumbbell

Figure 5.12: Accuracy of the NCPLD, Spike, Flip Flop and Vegas schemes as a function of the RTT of the connection, with uncorrelated losses ($PER = 0.5\%$) in the *Single Link* and *Congested Dumbbell* topologies.

Chapter 6

Performance of Enhanced TCP Sources

As we discussed in the previous Chapter, the Vegas Predictor proved the most accurate in classifying the nature of packet losses; therefore we decided to implement such predictor to enhance the error recovery mechanism of TCP NewReno-LP sources.

In this Chapter we measure TCP NewReno-LP performance, namely the achieved *goodput*, *fairness* and *overhead*, and compare it with other TCP versions, in several simulated network scenarios comprising wireless links, ad hoc networks and high-capacity channels. We considered two different types of connections: the long-lived TCP connections, typical of FTP file transfers, and short-lived connections, typical of HTTP connections.

To provide a bound to the performance of every possible TCP based on Loss Differentiation we also measure the performance of a TCP NewReno-LP source implementing *Ideal LDA*. In the following we present, and discuss, the results obtained by simulation.

6.1 Simulation Scenario

In accordance with existing literature [48], the first network topology we considered to measure the performance of TCP NewReno-LP is the one shown in Figure 6.1: a single TCP connection performs FTP bulk transfers between the source node S and the destination node D. This topology is a generalization of the *Single Link* network presented in the previous Chapter, where:

- the link $S - N$, with capacity and propagation delay respectively equal to C_{SN} and τ_{SN}, represents the wired part of the network, and it is eventually subject to congestion events;

- the link $N - D$, with capacity and propagation delay respectively equal to C_{ND} and τ_{ND}, represents the wireless part of the connection and it is characterized by random transmission errors.

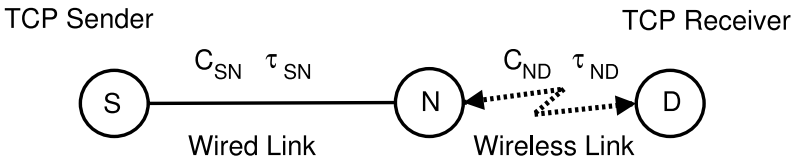


Figure 6.1: Network topology used to measure the performance of TCP NewReno-LP.

Again, we considered two cross-traffic configurations: in the first one the TCP connection performs data transfers without cross traffic on the wired link; in the second case, the TCP source shares the link $S-N$ with $n = 30$ UDP sources having the same priority as the TCP source. Each UDP source switches between ON and OFF periods, whose durations are both Pareto-distributed with shape parameter equal to 1.5 and mean set to $T_{on} = 100$ ms and $T_{off} = 200$ ms, respectively.

During the OFF period, the UDP sources do not transmit any packet. While in the ON period, each source transmits packets with 1500 byte size at a constant bit rate equal to r Mbit/s. The value of r has been chosen to assure that the average overall cross-traffic rate is equal to half the capacity of the wired link, C_{SN}, according to the following equation:

$$r = \frac{C_{SN}}{2n} \cdot \left(1 + \frac{T_{off}}{T_{on}}\right) \tag{6.1}$$

Table 6.1 shows the values used for all the network parameters in the simulations.

Parameter	Values
C_{SN}	2 Mb/s, 5 Mb/s, 10 Mb/s
τ_{SN}	variable between 10 ms and 75 ms
C_{ND}	10 Mb/s
τ_{ND}	0.01 ms
Wireless Losses	correlated/uncorrelated
Cross Traffic	with/without cross-traffic
Packet dimension	1500 byte

Table 6.1: Network Parameters used in the Topology of Figure 6.1

6.2 Metrics used to measure TCP performance

According to existing literature, we considered the following common metrics to evaluate TCP performance:

- *Accuracy of the Vegas Predictor*: in the previous Chapter we showed how the Vegas Predictor achieves the highest accuracy in classifying the cause of packet losses in various network scenarios. As we said, we measured the accuracy of this loss predictor embedding it in a classical TCP NewReno source that does not use this estimate within its error recovery scheme. As TCP NewReno-LP has a modified error recovery mechanism, it is necessary to measure the accuracy of the Vegas Predictor embedded in this modified TCP source.

- *Goodput*: for a TCP connection, it is possible to calculate its goodput based on the following equation:

$$Goodput = \frac{\text{Number of Transmitted bits} - \text{Number of Retransmitted bits}}{\text{connection's duration}} \text{ [bit/s]} \tag{6.2}$$

  In this Chapter we compare the goodput of TCP NewReno-LP with other TCP versions in various network scenarios. It is known [7] that classical TCP implementations like TCP NewReno perform poorly both when links are affected by high

packet losses and when the round trip time of the connection increases. For these reasons, we first measure the goodput achieved by the various TCP sources as a function of the packet error rate on the wireless channel; then, we measure the impact of the RTT on the performance of TCP.

- *Friendliness and Fairness*: we measure how the proposed TCP source is able to share fairly network resources both in homogeneous and mixed scenarios; the term $fairness$ relates to the performance of a set of TCP connections implementing the same algorithms, while the term $friendliness$ relates to the performance of a set of connections using different TCP flavors.

- *Overhead*: in [49] the overhead is defined as the ratio between the amount of retransmitted data and the amount of ACKed data. In a wireless environment, it is necessary to evaluate the ability of a TCP source to avoid useless retransmissions, thus achieving an efficient use of energy resources.

6.3 Performance of TCP NewReno-LP

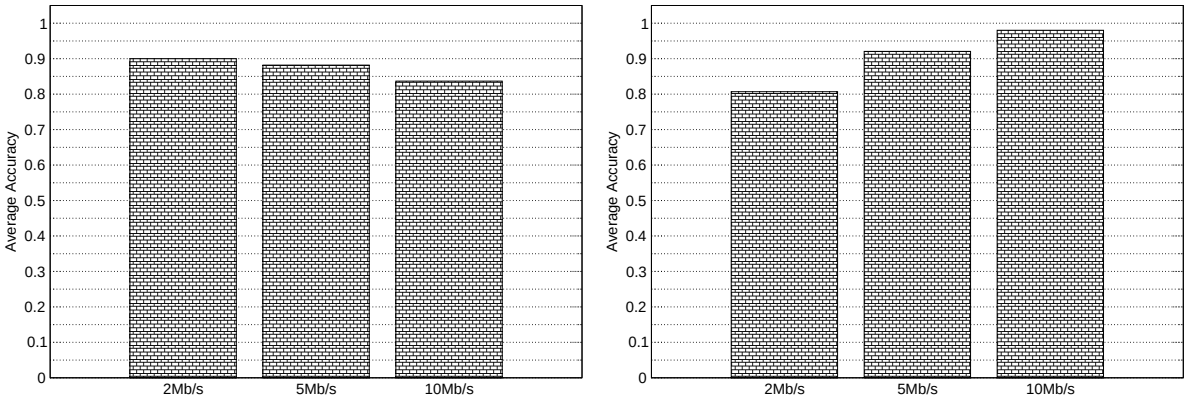
In this Section we show the performance achieved by TCP NewReno-LP in the network topology described above, with various parameters settings. All the values of accuracy, goodput, friendliness and overhead have been calculated over multiple simulations to achieve very narrow 97.5% confidence intervals [50].

6.3.1 Accuracy of the Vegas Predictor in TCP NewReno-LP

TCP NewReno-LP adopts a new error recovery mechanism based on the loss differentiation performed by the Vegas Predictor. Hence, it is necessary to measure the impact of this modification on the accuracy achieved by the loss discrimination procedure.

We observed that the accuracy of the Vegas predictor embedded in a TCP NewReno-LP source is very high. Figure 6.2(a) shows that this predictor achieves an average accuracy greater than 80% in the presence of uncorrelated errors on the wireless link, and for all the values of C_{SN} that we considered in our simulations.

When transmission errors are correlated, the accuracy achieved by the Vegas Predictor is even higher: Figure 6.2(b) shows that the accuracy in this case is very close to 100%.



(a) Uncorrelated Errors (b) Correlated Errors

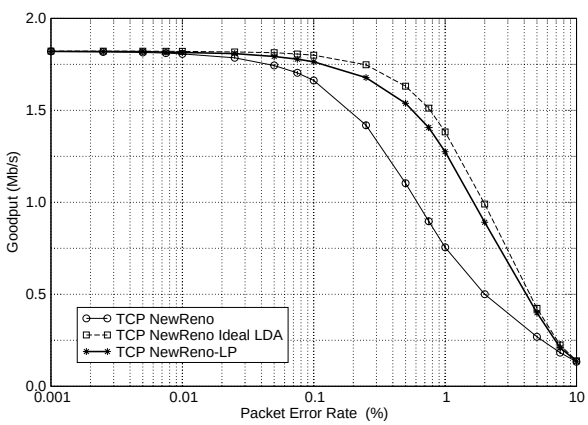
Figure 6.2: Average accuracy achieved by the Vegas Predictor with both uncorrelated and correlated errors on the wireless link and for different values of C_{SN} and $\tau_{SN} = 50$ ms.

6.3.2 Uncorrelated Losses

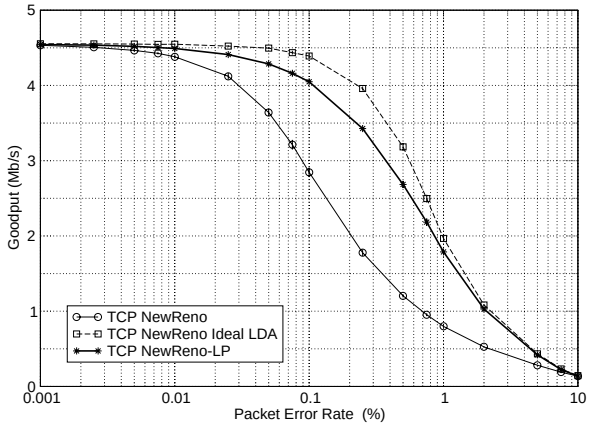
It is known that uncorrelated losses degrade the performance of classical TCP sources more than correlated ones [7]. We observed that TCP NewReno-LP allows to increase the goodput achieved during a connection if compared to standard TCP NewReno. Figures 6.3(a), 6.3(b) and 6.3(c) show the goodput achieved by TCP NewReno-LP enhanced with both the Vegas Predictor and Ideal LDA (to upper bound the achievable performance), as a function of the Packet Error Rate (PER) and for $C_{SN} = 2$, 5 and 10 Mb/s. The performance of TCP NewReno is also reported for comparison.

We underline that TCP NewReno-LP achieves higher performance than TCP NewReno; moreover, the performance achieved by TCP NewReno-LP enhanced with the Vegas Predictor is quite close to that achieved by the Ideal scheme, thus showing that no much space is left for further gain.

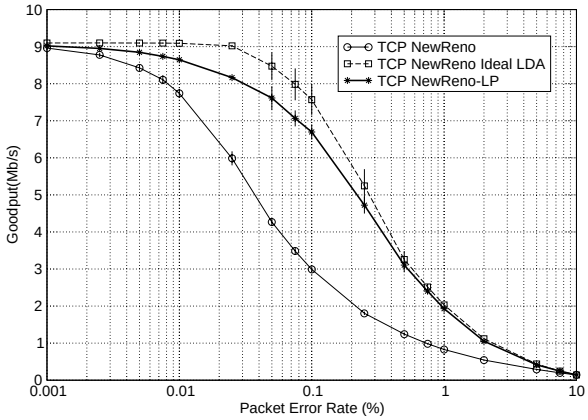
The goodput gain achieved by TCP NewReno-LP is more evident when the bandwidth-delay product of the connection increases. This is due to the very high accuracy in packet loss classification achieved by the Vegas Predictor, that proved to be always greater than



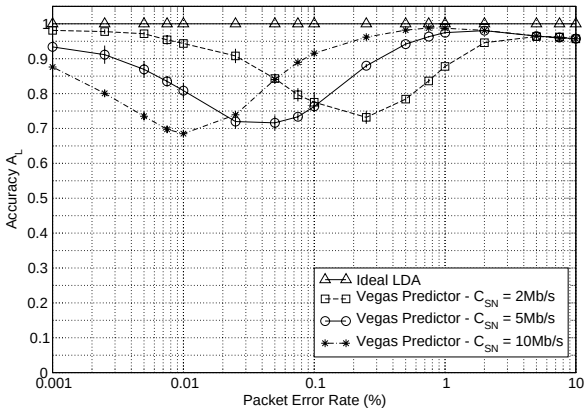
(a) C_{SN} =2Mb/s



(b) C_{SN} =5Mb/s



(c) C_{SN} =10Mb/s



(d) Accuracy of the Vegas Predictor

Figure 6.3: 6.3(a), 6.3(b), 6.3(c): goodput achieved by TCP NewReno-LP for C_{SN} = 2,5 and 10 Mb/s, respectively, as a function of the error rate (uncorrelated losses); 6.3(d): accuracy of the Vegas Predictor for the same values of C_{SN}; τ_{SN} = 50 ms.

70% for all the considered values of the PER and C_{SN}. To underline so, Figure 6.3(d) shows the accuracy of the Vegas Predictor as a function of the PER for the same values of C_{SN} considered above. We note that the Vegas Predictor approaches the Ideal LDA scheme for high PER values in all the considered scenarios, in accordance with the results shown in the previous Chapter.

In all the figures we observe that the goodput achieved by TCP NewReno-LP practically overlaps that achieved by TCP NewReno when the packet error rate is close to zero; this behavior is due to the accuracy with which the Vegas Predictor embedded in TCP NewReno-LP classifies congestion losses. This result is very important as it shows that

TCP NewReno-LP is able to achieve high goodput gain in the presence of consistent PER values, while maintaining the same level of aggressiveness of classical TCP NewReno on wired networks characterized by low packet error rates.

6.3.3 Correlated Losses

TCP NewReno-LP achieves a remarkable goodput gain over classical TCP NewReno sources in the presence of correlated losses. Even in this case, we observed that the Vegas Predictor embedded in TCP NewReno-LP achieves very high accuracy, practically approaching the performance achieved by a TCP source enhanced with an Ideal scheme.

Figures 6.4(a), 6.4(b) and 6.4(c) show the goodput achieved by the considered TCP versions when the wireless link is affected by such losses, and for different values of the capacity C_{SN}. We observe that the goodput achieved by TCP NewReno-LP is always higher than that achieved by TCP NewReno, and that the goodput gain is, in some cases, even greater than 100%.

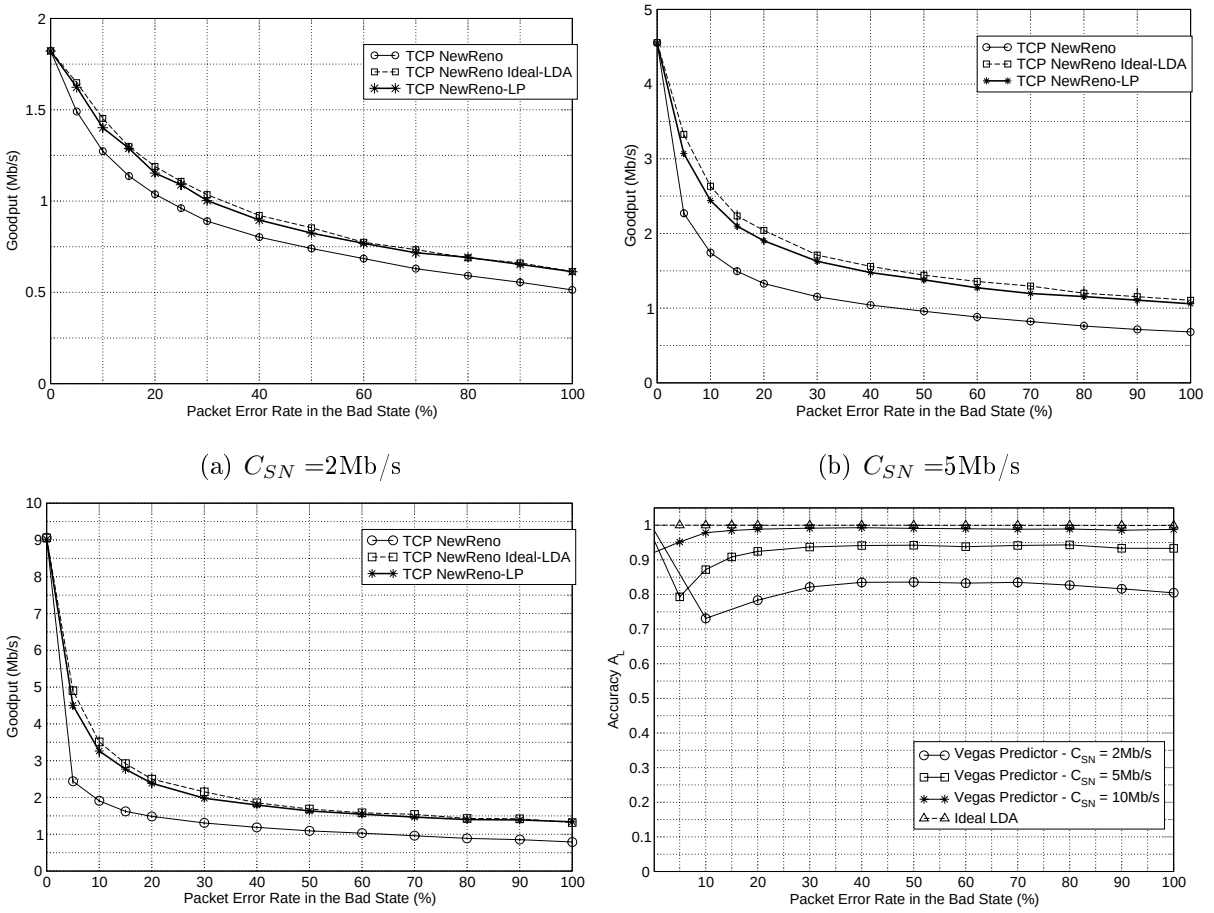
Note that, as expected, the performance of TCP NewReno-LP is close to that achieved by a TCP source enhanced with an Ideal predictor: the accuracy of the Vegas Predictor in these scenarios is, in fact, very close to 100% for the whole range of the error rates in the *Bad* state considered, as reported in Figure 6.4(d).

6.3.4 Impact of the Round Trip Time

Packet losses are not the only cause of TCP throughput degradation. Many studies proposed in the literature [51] have pointed out that TCP performance also degrades when the Round Trip Time of the connection increases. TCP NewReno-LP allows to alleviate this degradation and obtains better performance.

Figures 6.5(a), 6.5(b) and 6.5(c) show the goodput achieved by TCP NewReno-LP as a function of the RTT of the connection resulting from varying the propagation delay on the wired link $S - N$, with C_{SN} = 2,5 and 10 Mb/s respectively, in the presence of uncorrelated errors and with a constant Packet Error Rate equal to 0.5%.

As the capacity increases, the goodput gain achieved by TCP NewReno-LP is more



(a) C_{SN} =2Mb/s

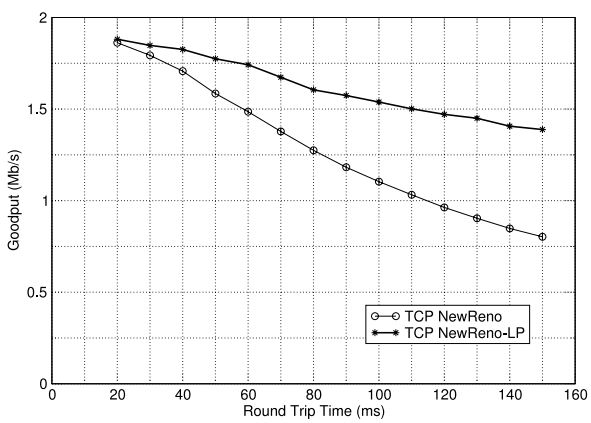
(b) C_{SN} =5Mb/s

(c) C_{SN} =10Mb/s

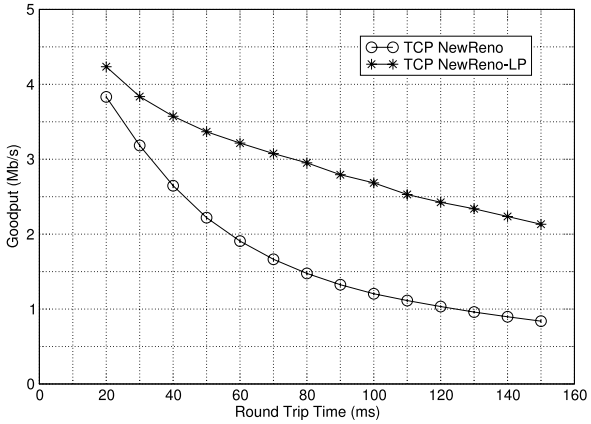
(d) Accuracy of the Vegas Predictor

Figure 6.4: 6.4(a), 6.4(b), 6.4(c): goodput achieved by TCP NewReno-LP for C_{SN} = 2,5 and 10 Mb/s, respectively, as a function of the error rate in the Bad state over a wireless link affected by correlated losses; 6.4(d): accuracy of the Vegas Predictor for the same values of C_{SN}; τ_{SN} = 50 ms.

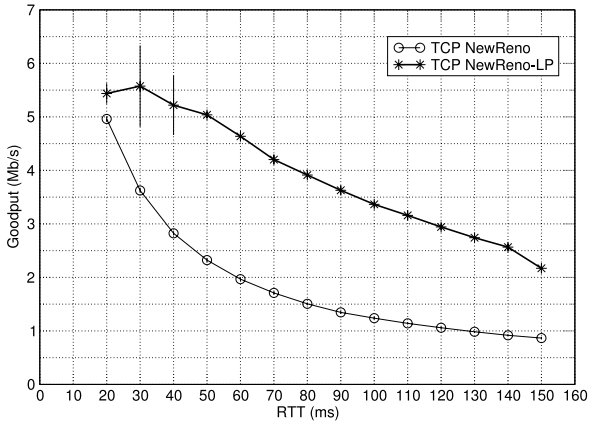
evident. Moreover, we observe that when the accuracy of the Vegas Predictor embedded in TCP NewReno-LP is higher, this TCP source achieves better performance. For low values of the bandwidth-delay product, the Vegas Predictor is less efficient in classifying the nature of packet losses, as shown in Figure 6.5(d) and in accordance with the results reported in the previous Chapter. In this situation, the goodput achieved by TCP NewReno-LP is slightly higher than that of TCP NewReno. On the contrary, for higher bandwidth-delay products, the accuracy of the Vegas Predictor increases and TCP NewReno-LP achieves high goodput gain if compared to TCP NewReno.



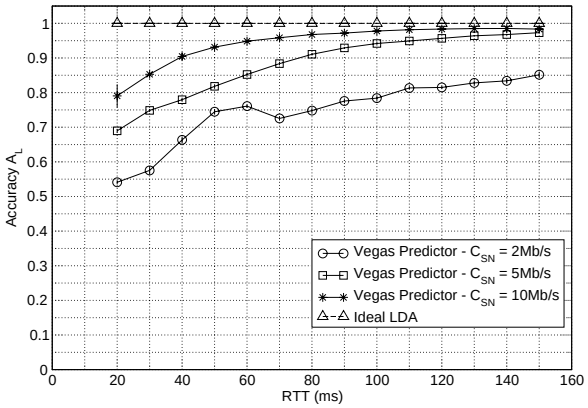
(a) C_{SN} =2Mb/s



(b) C_{SN} =5Mb/s



(c) C_{SN} =10Mb/s



(d) Accuracy of the Vegas Predictor

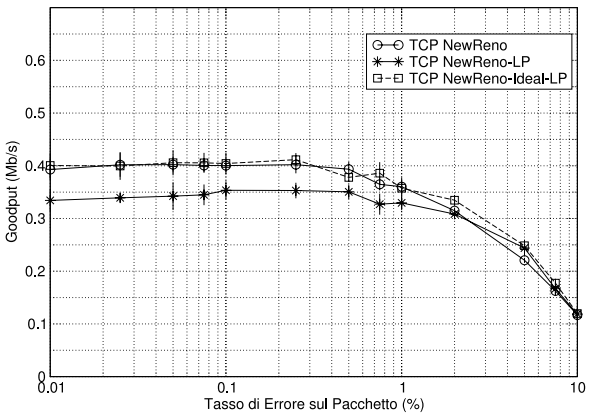
Figure 6.5: 6.5(a), 6.5(b), 6.5(c): goodput achieved by TCP NewReno-LP as a function of the RTT of the connection for C_{SN} = 2,5 and 10 Mb/s, respectively; 6.5(d): accuracy of the Vegas Predictor for the same values of C_{SN}; transmission errors on the wireless channel are uncorrelated and the PER is equal to 0.5%.

6.3.5 Performance in the presence of Cross-Traffic

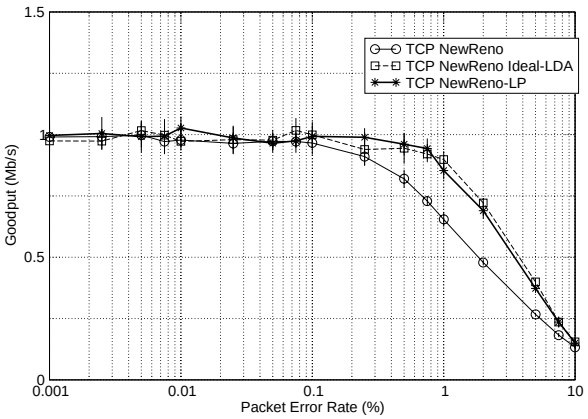
When cross-traffic is activated on the wired link $S - N$, the number of congestion losses experienced by TCP increases consistently. Moreover, as the TCP connection is now sharing the bottleneck capacity with other traffic sources, its bandwidth-delay product diminishes. This impacts on the accuracy of the Vegas Predictor, that performs better with connections having high bandwidth-delay products.

However, TCP NewReno-LP shows a consistent goodput gain with respect to classical

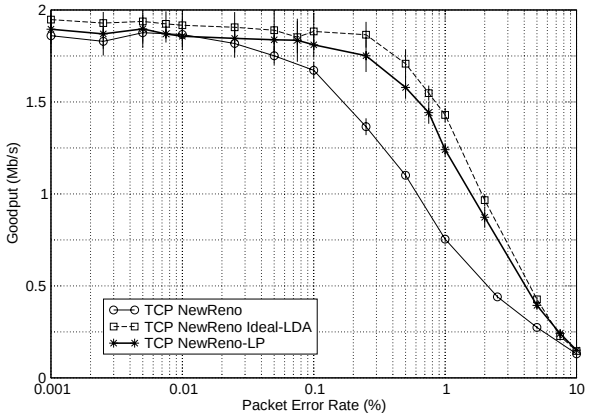
TCP NewReno. Figures 6.6(a), 6.6(b) and 6.6(c) show the goodput achieved by TCP NewReno-LP as a function of the PER in the presence of the $n = 30$ Pareto ON/OFF sources transmitting at the rate r, as defined in Section 6.1.



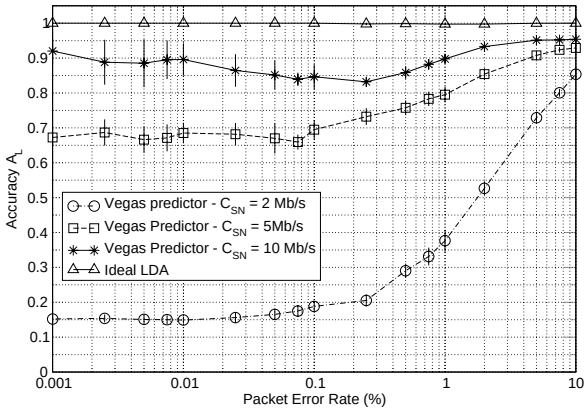
(a) C_{SN} =2Mb/s



(b) C_{SN} =5Mb/s



(c) C_{SN} =10Mb/s



(d) Accuracy of the Vegas Predictor

Figure 6.6: 6.6(a), 6.6(b), 6.6(c): goodput achieved by TCP NewReno-LP in the presence of cross-traffic as a function of the PER; 6.6(d): accuracy of the Vegas Predictor for different values of C_{SN}; packet losses are uncorrelated; $\tau_{SN} = 50$ ms.

We note that TCP NewReno-LP enhanced with the Vegas Predictor achieves practically the same performance as a TCP based on an Ideal predictor when the bandwidth-delay product is high (i.e. for high values of C_{SN}); on the contrary, when the bandwidth-delay product is lower, the goodput achieved by TCP NewReno-LP can, in some cases, be slightly lower than that of TCP NewReno.

This observation is in line with the accuracy in packet loss classification achieved by

the Vegas Predictor in the presence of cross-traffic, as shown in Figure 6.6(d). We note that the accuracy of this predictor is low when $C_{SN} = 2$ Mb/s: in this case the goodput achieved by TCP NewReno-LP is quite lower than the goodput that could be obtained with an Ideal estimator (Figure 6.6(a)).

On the contrary, for higher values of C_{SN}, the accuracy of the Vegas Predictor is sufficiently high and the performance of TCP NewReno-LP approaches that achieved with an Ideal loss differentiator (Figures 6.6(b) and 6.6(c)).

Note that, as the capacity of wireless access networks is continuously growing in these last years, such behavior of TCP NewReno-LP makes it suitable for the next generation networks.

6.3.6 Short-Lived TCP Connections

It is known [52, 53] that the major part of the Internet traffic is constituted by short-lived TCP connections that use the HTTP protocol. A recent study showed that approximately 90% of the TCP connections involve the transmission of less than 10 kbyte of data.

Hence, we studied the performance of TCP NewReno-LP with short-lived TCP connections. We considered, in line with the literature [52], a typical HTTP connection involving the transfer of a 10 kbyte file over a 5 Mb/s link affected by a 5% random packet loss, with a 100 ms Round Trip Time. We simulated 500 transfers and measured the duration of each file transfer.

The average time to complete the transfer was 0.79 s for TCP NewReno-LP and 0.81 s for TCP NewReno. Hence, also for short file transfers, TCP NewReno-LP achieves a slight improvement over the current TCP version.

However, as we showed in the previous Sections, the goodput gain of TCP NewReno-LP is much higher when FTP connections are involved.

6.3.7 Friendliness and Fairness

So far we have shown that the TCP NewReno-LP scheme estimates accurately the cause of packet losses and that achieves higher goodput than existing TCP versions over wireless

links with both uncorrelated and correlated losses.

Following the methodology proposed in [14], we evaluated fairness and friendliness of TCP NewReno-LP in a variety of network scenarios and we compared them by those achieved by TCP Westwood. The term *fairness* relates to the performance of a set of TCP connections implementing the same algorithms, while the term *friendliness* relates to the performance of a set of connections using different TCP flavors.

This Section shows how the proposed scheme is able to share fairly and friendly network resources in mixed scenarios where the sources use different TCPs.

To this purpose, in accordance with the literature [48] we considered the network topology shown in Figure 6.7.

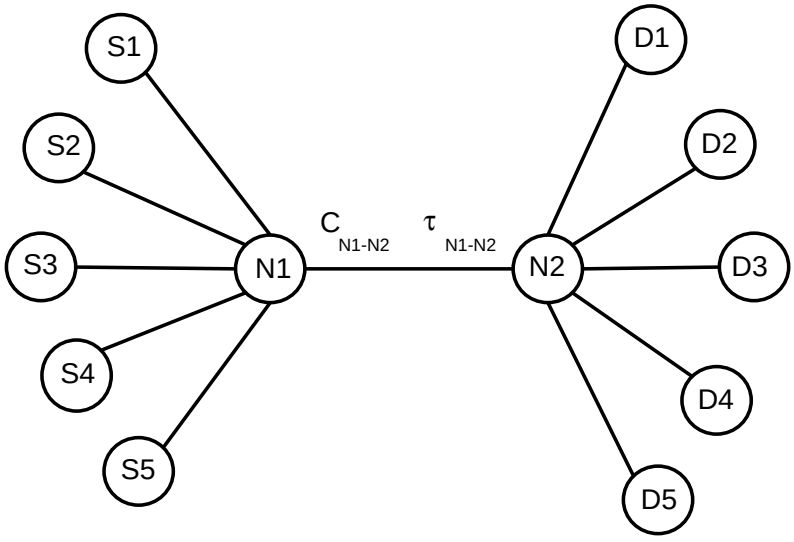


Figure 6.7: Network topology used to measure TCP's friendliness and fairness.

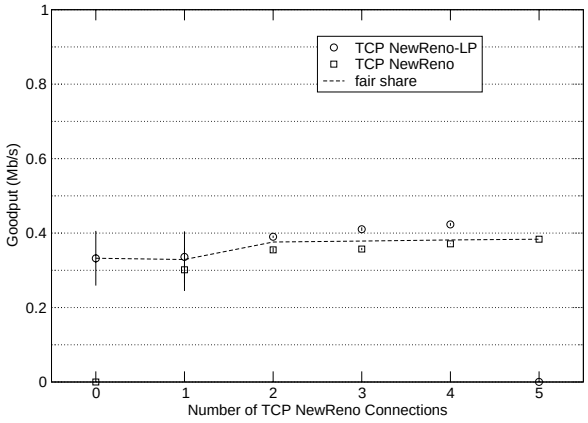
In this scenario, N TCP NewReno-LP and K TCP NewReno sources perform FTP file transfers simultaneously over the link $N1 - N2$ having capacity $C_{N_1N_2}$ and propagation delay $\tau = 50$ms; all the links are error-free.

We considered the case in which $N + K = 5$, and we measured the average goodput achieved by TCP NewReno and TCP NewReno-LP sources. These values are compared to the *fair share*, defined as the average of the goodput achieved by all the TCP sources regardless of their nature.

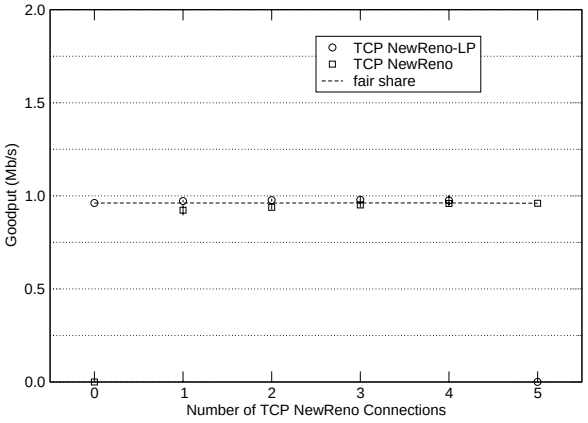
We observed that TCP NewReno-LP achieves high friendliness towards TCP NewReno

for every value of N, and this is more evident when the accuracy of the LDA scheme embedded in this TCP source is higher.

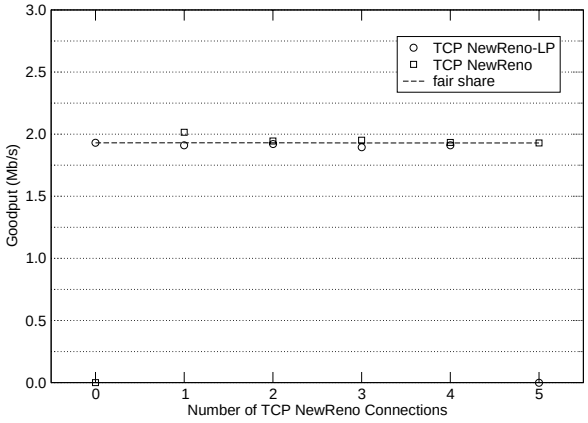
Figures 6.8(a), 6.8(b) and 6.8(c) show the average goodput achieved in the topology of Figure 6.7. We note that the number of concurrent TCP NewReno connections does not influence the goodput achieved by TCP NewReno-LP. This result is in line with the behavior of TCP NewReno-LP at low error rates already observed in Figures 6.3 and 6.4, that show how the goodput achieved by TCP NewReno-LP is very close to that of TCP NewReno when the wireless link is not affected by packet losses. We can conclude that TCP NewReno-LP achieves an high level of friendliness towards classical TCP sources.



(a) $C_{N_1N_2}$ =2Mb/s



(b) $C_{N_1N_2}$ =5Mb/s



(c) $C_{N_1N_2}$ =10Mb/s

Figure 6.8: Friendliness achieved by TCP NewReno-LP with a variable number of concurrent TCP NewReno connections in the topology illustrated in Figure 6.7, for different values of $C_{N_1N_2}$.

To measure the level of fairness achieved by TCP NewReno-LP we considered the same scenario described above, first with 5 TCP NewReno-LP connections and then with 5 TCP NewReno sources sharing a 10 Mbit/s link with RTT equal to 100 ms. In this scenarios, congestion is the only cause of packet losses. The Jain's fairness index [54] of 5 TCP NewReno-LP connections was equal to 0.9987, and that achieved by 5 TCP NewReno sources was equal to 0.9995. These results confirm that TCP NewReno-LP achieves the same level of fairness of TCP NewReno. Similar results have been obtained for other values of the bottleneck capacity.

We also extended our simulation campaign to more complex scenarios with a varying number of competing connections. The results obtained confirm that TCP NewReno-LP achieves an high level of fairness and friendliness towards TCP NewReno, thus allowing its smooth introduction into the Internet.

6.3.8 Overhead

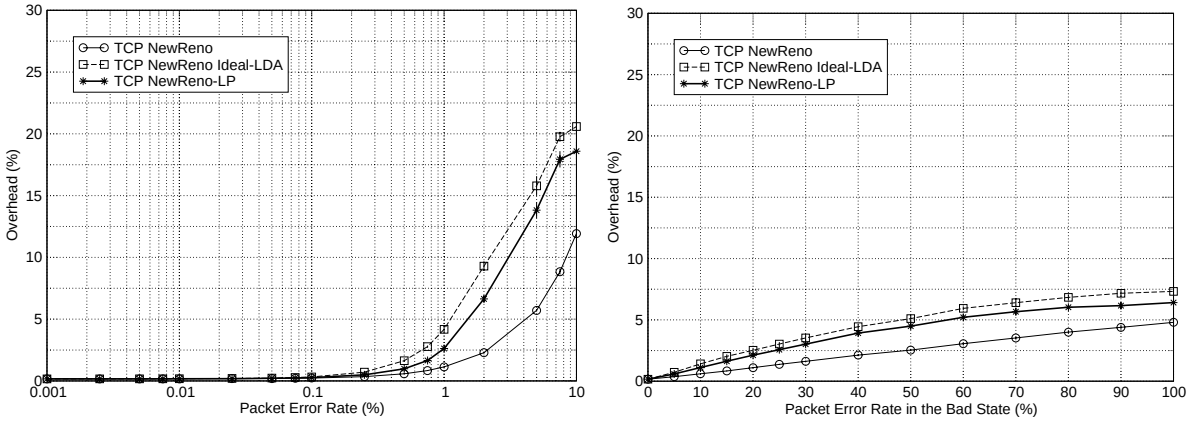
When packet losses are due to random impairments on the wireless link, TCP's retransmissions are necessary to enhance the error recovery mechanism achieving high goodput gain with respect to classical TCP sources. However, an aggressive retransmission strategy can cause both network congestion and low energy efficiency of the transmission protocol.

TCP NewReno-LP proves to introduce in the network an almost negligible traffic overhead when the packet error rate is low; for higher PER values, however, the overhead of this TCP source increases, as it is shown in Figure 6.9(a) where uncorrelated packet losses are considered.

When packet losses are correlated, the overhead introduced by TCP NewReno-LP is considerably lower, as it is shown in Figure 6.9(b). Note that similar results have been obtained for all the considered values of the bottleneck capacity C_{SN}.

6.3.9 Comparison with TCP performing Bandwidth Estimation

In Chapter 3 we reviewed various TCP enhancements based on bandwidth estimation recently proposed in the literature, proposing TIBET, a new scheme that achieves unbiased



(a) Uncorrelated Errors

(b) Correlated Errors

Figure 6.9: Overhead introduced by TCP NewReno-LP in the network of Figure 6.1 with $C_{SN} = 5$ Mb/s and $\tau_{SN} = 50$ ms, with uncorrelated and correlated packet losses

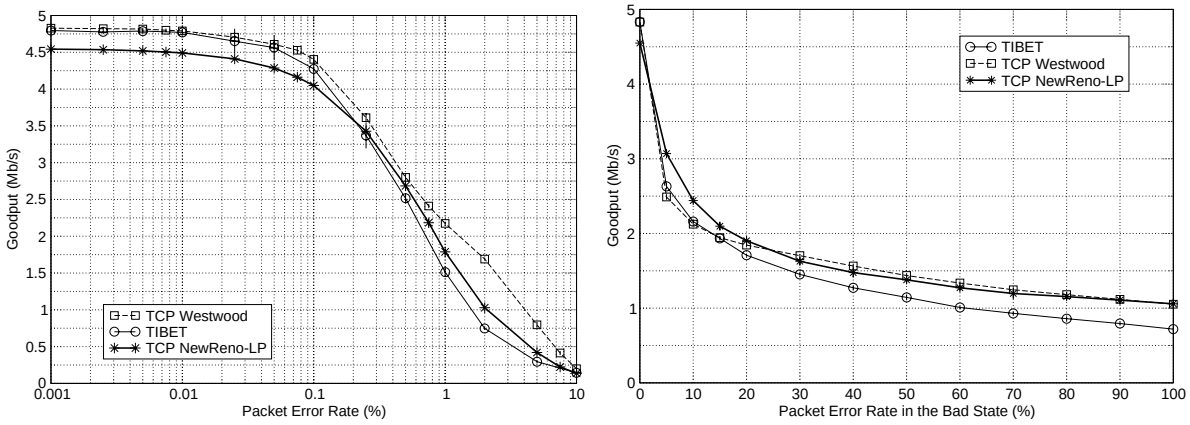
and stable estimates.

We observe that, according to existing literature [16], these bandwidth estimation algorithms can be viewed as performing an *implicit* loss classification based on the rate at which the sender receives the ACK stream.

Hence, in this Section we compare the performance achieved by TCP sources enhanced with bandwidth estimation (*implicit* loss classification) to that of TCP NewReno-LP, that performs an *explicit* loss classification.

In the presence of uncorrelated losses, the goodput achieved by TIBET and TCP Westwood is comparable to that achieved by TCP NewReno-LP. For low values of the packet error rate, TCP sources performing implicit loss classification achieve slightly higher goodput, as shown in Figures 6.10(a) and 6.10(b), where the performance of the three modified TCP sources is measured in the network scenario depicted in Figure 6.1 with $C_{SN} = 5$ Mb/s, with both correlated and uncorrelated losses on the wireless link.

Note that TCP NewReno-LP achieves better performance than TIBET, especially when packet losses are correlated. As we already observed in Chapter 3, TCP Westwood is very aggressive, and its goodput gain is achieved at the expenses of a lower friendliness



(a) Uncorrelated Errors

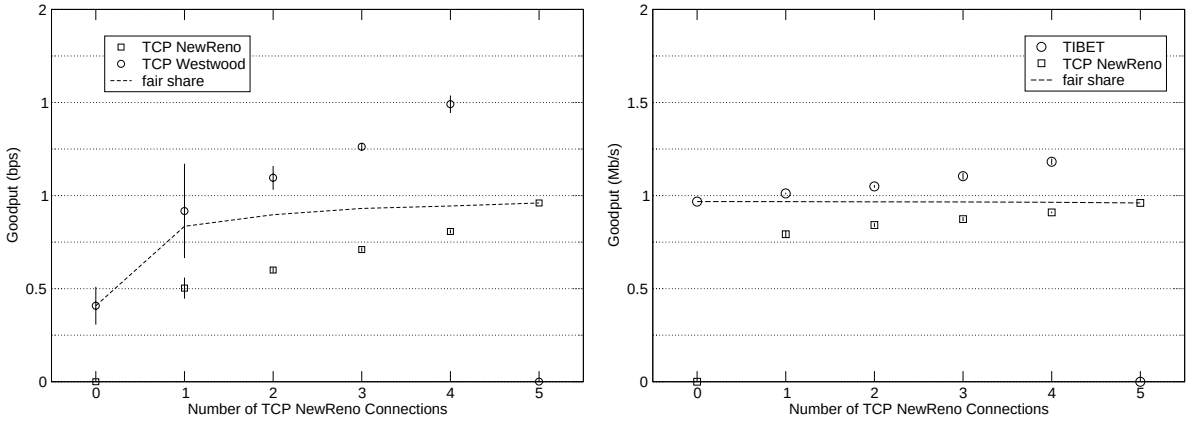
(b) Correlated Errors

Figure 6.10: Goodput achieved by TCP NewReno-LP, TIBET and TCP Westwood in the topology of Figure 6.1 without cross traffic, with $C_{SN} = 5$ Mb/s and $\tau_{SN} = 50$ ms, with (a) uncorrelated and (b) correlated losses on the wireless link.

towards other TCP versions. However, we observed that TCP NewReno-LP can outperform both TIBET and TCP Westwood in several network scenarios and in real Internet measurements, as we will show in the next Chapter devoted to the presentation of the test beds used to gain insight into the implementations of the proposed enhanced TCP sources.

Moreover, TCP NewReno-LP achieves greater friendliness than both TIBET and TCP Westwood in several network topologies. Figures 6.11(a) and 6.11(b) show the average goodput achieves by TCP Westwood and TIBET in the network topology depicted in Figure 6.7, with a varying number of competing TCP NewReno connections. Recall that, as already discussed in the previous Sections, TCP NewReno-LP achieves a goodput that is practically coincident with the fair-share.

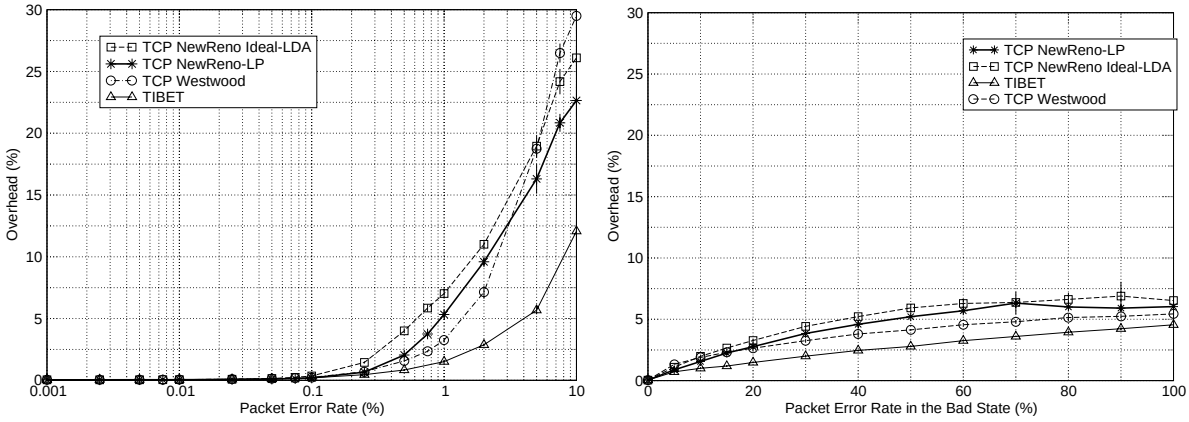
If we consider the energy efficiency of the proposed TCP protocols, TIBET and TCP Westwood introduce a slightly lower overhead in network utilization than TCP NewReno-LP. Figure 6.12(a) shows the overhead introduced by the three TCP versions under examination in the topology illustrated in Figure 6.1 with $C_{SN} = 5$ Mb/s and $\tau_{SN} = 50$ ms, in the presence of uncorrelated errors on the wireless channel. Note that the goodput gain in the three modified TCP sources is obtained at the expenses of a slight increase



(a) TCP Westwood (b) TIBET

Figure 6.11: Friendliness of TCP Westwood and TIBET in the topology depicted in Figure 6.3.7 with $C_{12} = 5$ Mb/s and $\tau_{SN} = 50$ ms.

in the overhead injected into the network. Finally, it is interesting to notice that when losses are correlated, the differences in energy consumption of the three protocols are less evident than in the case of uncorrelated errors, as Figure 6.12(b) demonstrates.



(a) Uncorrelated Errors (b) Correlated Errors

Figure 6.12: Overhead introduced in the network of Figure 6.1 with $C_{SN} = 5$ Mb/s and $\tau_{SN} = 50$ ms, for different error models on the wireless link.

6.4 Effect of Link Layer ARQ: the 802.11 case

The scenario shown in Figure 6.13 has been used to test the performance of the TCP sources examined in this Chapter 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.

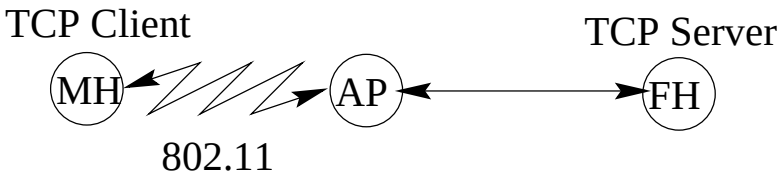
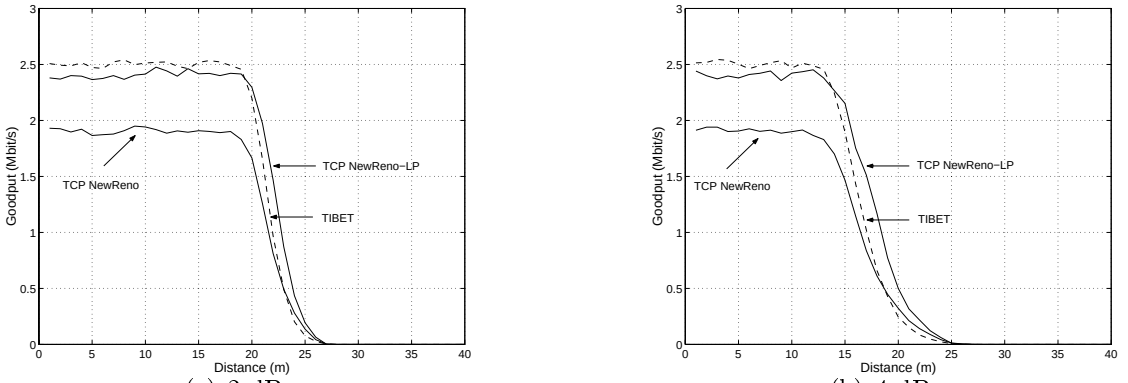


Figure 6.13: Mixed wired-wireless 802.11 network topology.

The wireless channel includes a power loss due to shadowing, modeled as a log-normal random variable with standard deviation set equal to 2 and 4 dB in the simulations to take into account different levels of fading, 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. The round trip time of the connection is equal to 100 ms, and 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 [55]. 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 Figure 6.14, show a great improvement achieved by TCP NewReno-LP and TIBET over TCP NewReno. Note that TCP NewReno-LP achieves a goodput gain in the order of 100% with respect to the other two TCP sources especially for intermediate distances between the mobile host and the access point.

Similar results have been obtained varying the round trip time of the connection and the capacity of the wireless link.



(a) 2 dB (b) 4 dB

Figure 6.14: Goodput achieved by TCP NewReno-LP, TIBET and TCP NewReno in the topology of Figure 6.13 with shadowing equal to (a) 2 dB (b) 4dB.

6.5 TCP Performance with Ad Hoc Routing Protocols

Mobile ad hoc networks have gained a lot of attention lately as a means of providing continuous network connectivity to mobile computing devices regardless of physical location. Recently a large amount of research has focused on the routing protocols needed in such an environment. In this Section we investigate the effects that different ad hoc routing protocols have on TCP performance.

According to [56], our network model consists of 30 nodes in a 1500x300 meter flat, rectangular area. The nodes move according to the random waypoint mobility model described in [57], where the authors showed how to implement such model to construct more efficient and reliable simulations for mobile ad hoc networks.

In the random waypoint model each node x picks a random destination and speed in the rectangular area and then travels to the destination in a straight line. Once node x arrives at its destination, it pauses, picks another destination, and continues onward. We used a pause time of $0s$ so that each node is in constant motion throughout the simulation. All nodes communicate with 802.11 technology, and the wireless links are characterized by the same levels of shadowing and path loss described in the previous Section.

All our simulation results are based on the average throughput of 50 scenarios. In each scenario, two nodes are randomly chosen: one acts as a client and performs a long-lived

FTP file transfer with the other node, that acts like a server. The speed of each node is uniformly distributed in an interval of $0.9 \cdot v$ and $1.1 \cdot v$, for some mean speed v, and such mean speed is varied from 1 to 8 m/s.

We considered the most notable ad hoc routing algorithms proposed in the literature, namely Ad hoc On demand Distance Vector (AODV) [58], Optimized Link State Routing (OLSR) [59], and Destination-Sequenced Distance Vector (DSDV).

Figure 6.15 reports the average goodput achieved by TCP NewReno, TCP NewReno-LP and TIBET as a function of the average nodes speed.

AODV proved the most performant algorithm for practically the whole set of nodes speeds. Note also that the performance degradation in high speed scenarios is not so relevant as with the other two ad hoc routing protocols. These results indicate that, at least in this network scenario, AODV is the most suitable routing algorithm to be used when TCP connections are established across the nodes of the ad hoc network.

We also underline that, for low nodes speeds, OLSR allowed to achieve higher performance. Hence, when this condition is verified, OLSR represents a possible alternate choice as an efficient ad hoc routing algorithm.

Finally, DSDV performed poorly with respect to the other two routing protocols, thus resulting less appealing for a possible application to a real network scenario.

Note that, in all these scenarios, the three TCP sources practically achieved the same performance. This is mainly due to the demanding network conditions, where frequent disconnections are experienced by the TCP connections. In this case, almost all losses are indicated by timeout expirations and the congestion window of the different TCP sources always assumes low values. In these network conditions, performance improvement can be achieved very difficultly without changing completely the standard TCP behavior.

6.6 High Speed Links

Currently, with the increasing network capacity offered by the WDM optical technology, the TCP protocol is faced to new challenges.

In a steady-state environment, in fact, it is known that with a packet loss rate p, the

current Standard TCP's average congestion window is roughly $\frac{1.2}{\sqrt{p}}$ segments. This places a serious constraint on the congestion windows that can be achieved by TCP in realistic environments. For example, for a standard TCP NewReno connection with 1500-byte packets and a 100 ms round-trip time, achieving a steady-state throughput of 10 Gb/s would require an average congestion window of 83333 segments, and a packet drop rate of at most one congestion event every 5000000000 packets (or equivalently, at most one congestion event every 1 hour and 2/3). The average packet drop rate of at most $2 \cdot 10^{-10}$ needed for full link utilization in this environment corresponds to a bit error rate of at most $2 \cdot 10^{-14}$, and this is an unrealistic requirement for current networks.

To address this fundamental limitation of TCP and of the TCP response function (i.e. the function mapping the steady-state packet drop rate to TCP's average sending rate in packets per round-trip time), the so-called High Speed TCP option has been recently proposed in the literature.

However, in lossy environments, even this technique is unable to provide a satisfactory utilization of network resources. On the other hand, TCP NewReno-LP allows to sustain high speed transfers even at high network losses.

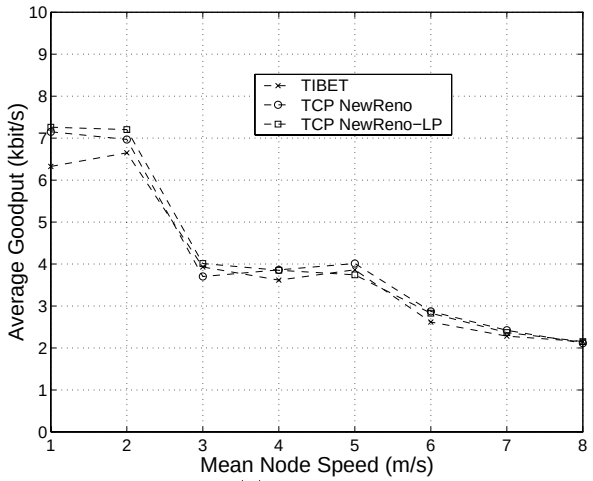
Figure 6.16 shows the goodput achieved by TCP NewReno, TCP NewReno-LP and TIBET transmitting over a 10 Gigabit/s link affected by random packet losses. The round trip time of the connection is equal to 100 ms.

When the TCP sources do not use the HSTCP option (Figure 6.16(a)) TCP NewReno-LP outperforms all the other TCP variants for the whole range of packet error rates. The goodput gain is particularly relevant when packet losses are not consistent, that is, in the network scenario that is more likely to occur in a connection that uses such high speed links.

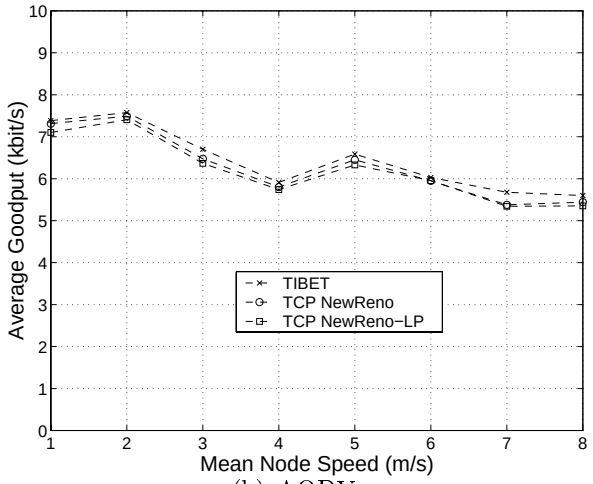
On the other hand, when the HSTCP option is turned on (Figure 6.16(b)) the performance of classical TCP NewReno and TCP NewReno-LP are comparable for very low packet error rates. However, when the PER slightly increases, TCP NewReno-LP still outperforms the other TCP versions. Note that TCP NewReno-LP's performance slightly increases when the HSTCP option is used; however, the main factor that determines its

performance is its enhanced loss recovery based on loss differentiation.

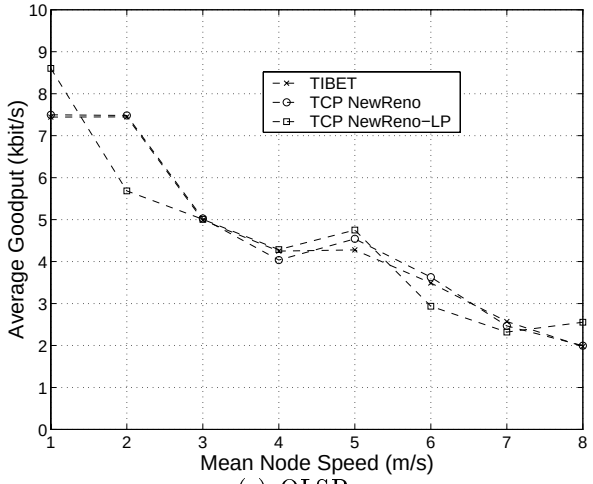
The same experiment was conducted with a link having the same configuration described above and a capacity equal to 5 Gigabit/s. The results are shown in Figure 6.17 and confirm the observations presented above.



(a) DSDV

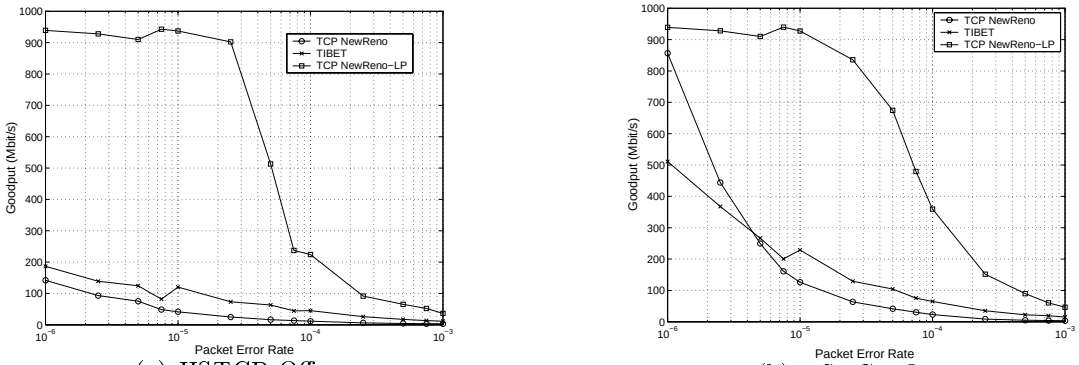


(b) AODV



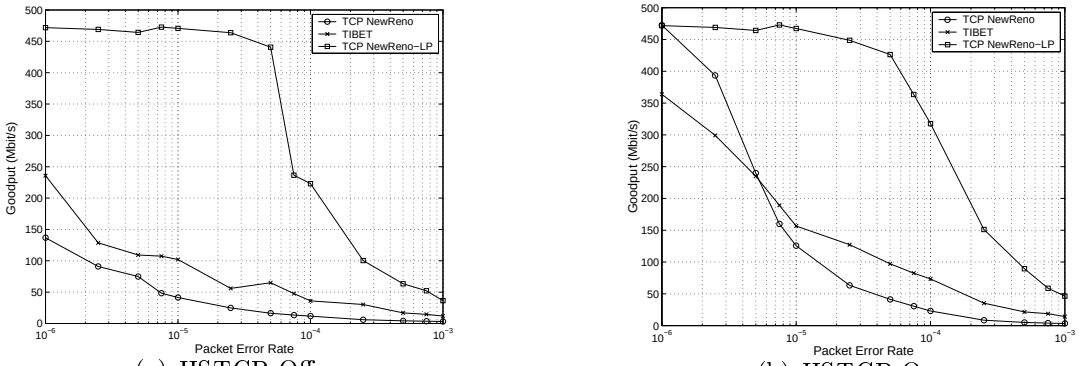
(c) OLSR

Figure 6.15: Goodput achieved by TCP NewReno-LP, TIBET and TCP NewReno with different ad hoc routing protocols as a function of the mean speed of mobile hosts.



(a) HSTCP Off (b) HSTCP On

Figure 6.16: Goodput achieved by TCP NewReno-LP, TIBET and TCP NewReno on a 10 Gigabit/s link as a function of the packet drop rate with (a) HSTCP option turned off (b) HSTCP option turned on.



(a) HSTCP Off (b) HSTCP On

Figure 6.17: Goodput achieved by TCP NewReno-LP, TIBET and TCP NewReno on a 5 Gigabit/s link as a function of the packet drop rate with (a) HSTCP option turned off (b) HSTCP option turned on.

Chapter 7

Implementation and Test Bed

To get more details on the TCP NewReno-LP implementation we have built three test beds. The first one implements an heterogeneous network with a wired part and a wireless link affected by random losses; the second one implements an ad hoc network and the third one models a Cellular IP network with mobile hosts that experience temporary disconnections as they move from one area to the other of the network.

7.1 Heterogeneous Network

The first test bed we considered is shown in Figure 7.1: it consists of a PC server, a client and a PC router, all connected by 10 Mb/s LAN cables. The PC router emulates a wireless link with the desired delay and packet loss rate using the NIST Net software [60], thus allowing to control and tune the features of the wireless link.

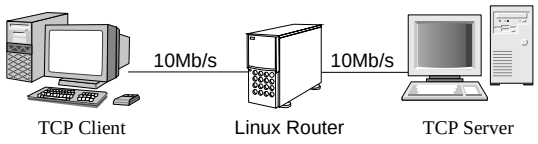


Figure 7.1: Test bed Topology for TCP performance evaluation.

In the PC server, besides TCP NewReno that is the current TCP implementation in the Linux kernel version 2.2-20, we have implemented TCP NewReno-LP, TIBET

and TCP Westwood. The choice to implement the TCP variants detailed above in the Linux kernel version 2.2-20 was motivated by the observation that this version is fully compliant with the standard TCP implementation as recommended in [23, 27]. Successive versions of the Linux kernel, starting from 2.4, introduced improved features as the Rate-Halving algorithm, a modified congestion control engine and the so-called undo procedures that are not yet considered standard and can have a deep impact on TCP performance, thus masking the advantages introduced by bandwidth estimation and loss differentiation techniques.

7.1.1 Uncorrelated Losses

We ran the test bed and we measured the goodputs achieved by the four TCP versions. Figure 7.2 compares the steady-state goodput achieved by TCP NewReno-LP, 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.

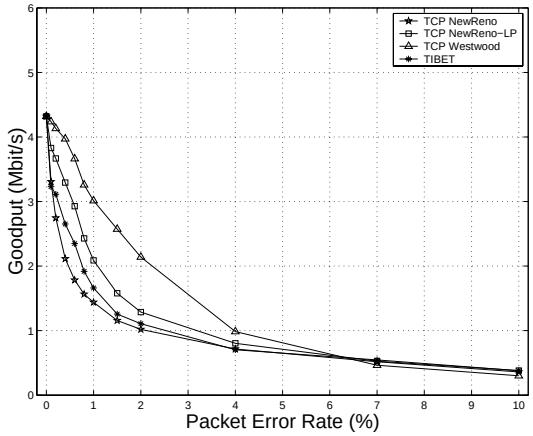
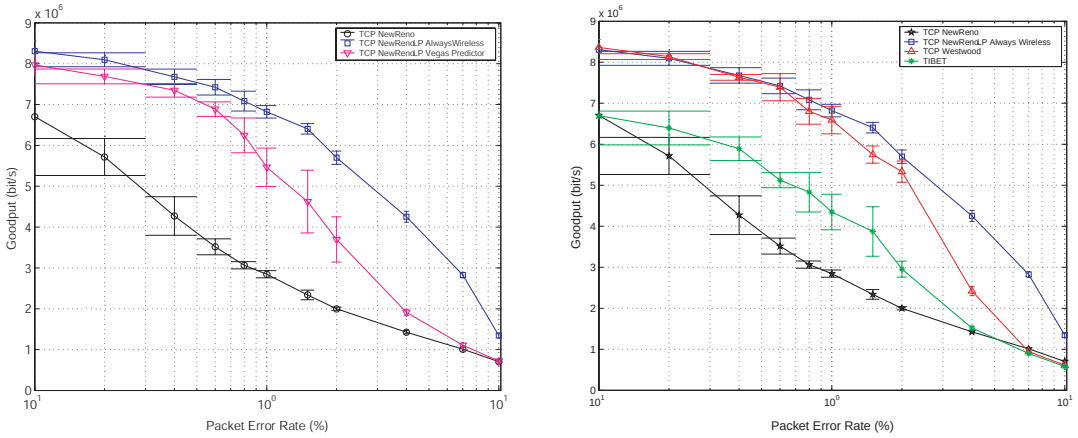


Figure 7.2: Goodput achieved by TCP NewReno-LP, TIBET, TCP Westwood and TCP NewReno in the Test Bed.

The measures on this real scenario validate the results obtained by simulation in the previous Chapters and provide a further support on the advantages of TCP NewReno-LP over TCP NewReno. Figure 7.2 also shows the improvement achieved by TCP NewReno-LP over TIBET, more evident for PER values in the 1% to 4% range.

Note that in this scenario, as well as in many measures presented in this Chapter, TCP Westwood obtained a higher goodput than any other TCP version. This behavior is due to its overestimate of the available bandwidth, that leads to aggressive behavior and unfair sharing of network resources, as we showed in Chapters 3 and 6, and as we discussed in detail in [1].

To further test the performance of these TCP implementations we considered the same test bed configuration, performing FTP file transfers involving both medium-sized (50 Mbyte) and long-sized (140 Mbyte) files. The Round Trip Time of the connection was set, using NISTNet, to 50 ms and 100 ms, to analyze the impact of the RTT on the performance achieved by these sources.



(a) (b)

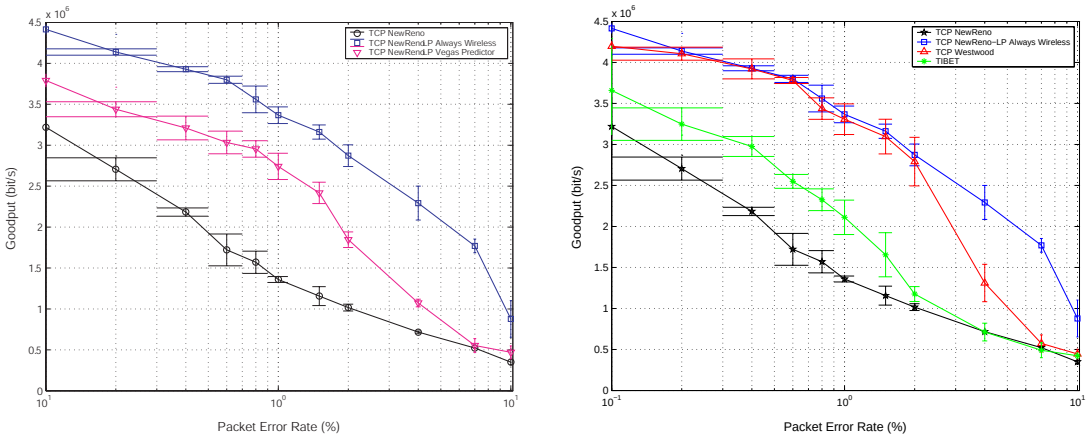
Figure 7.3: Goodput measured as a function of the Packet Error Rate, $RTT = 50$ ms, medium file transfer.

Figures 7.3(a) and 7.3(b) show the goodput achieved by the considered TCP sources performing a medium-sized file transfer with $RTT = 50$ ms, as a function of the packet error rate on the wireless link. In both the Figures we have reported the goodput achieved by TCP NewReno and TCP NewReno-LP enhanced with an Ideal predictor, to provide, respectively, a lower and an upper bound to the performance that can be achieved.

Note that TCP NewReno-LP enhanced with the Vegas Predictor allows to obtain high goodput gain in all the considered scenarios and for all packet losses. Its perfor-

mance is better than that achieved by TIBET, and only TCP Westwood achieves higher performance, at the price of being unfair towards existing TCP versions.

To gauge the impact of the RTT we considered the same scenario described above, setting the RTT of the TCP connections equal to 100 ms. Figures 7.4(a) and 7.4(a) show the measured goodput as a function of the PER on the wireless link.



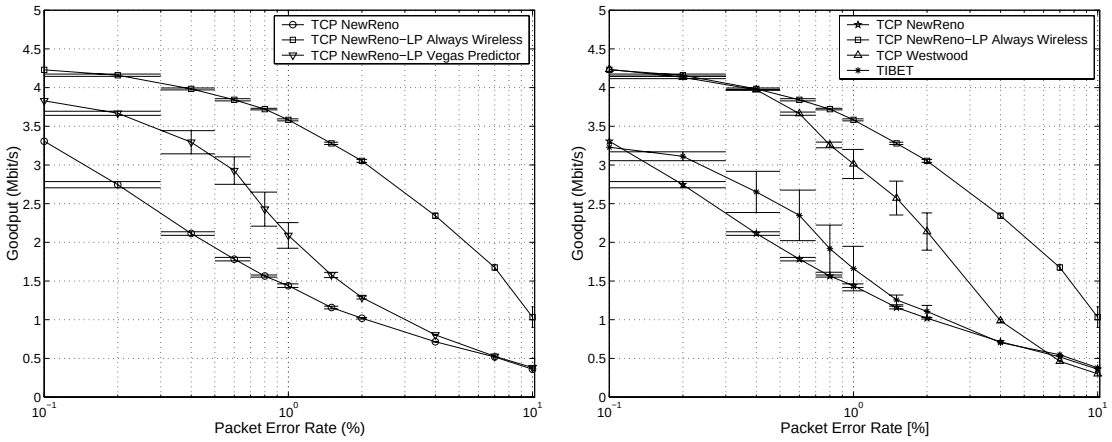
(a) (b)

Figure 7.4: Goodput measured as a function of the Packet Error Rate, $RTT = 100$ ms, medium file transfer.

Evidently, the overall performance of the TCP sources diminishes, as packet losses have a greater impact on the performance of connections having higher bandwidth-delay product, as pointed out in [7]. However, even in this case the utilization of TCP NewReno-LP allows to mitigate this negative effect, achieving high performance than classical TCP sources.

Finally, we considered long file transfers, to better characterize the steady-state behavior of the considered TCP sources. The results, involving connections having $RTT = 100$ ms, are reported in Figures 7.5(a) and 7.5(b).

These results confirm that in all the considered scenarios, TCP NewReno-LP has an edge of advantage over other TCP versions.



(a) (b)

Figure 7.5: Goodput measured as a function of the Packet Error Rate, $RTT = 100$ ms, long file transfer.

7.1.2 Correlated Losses

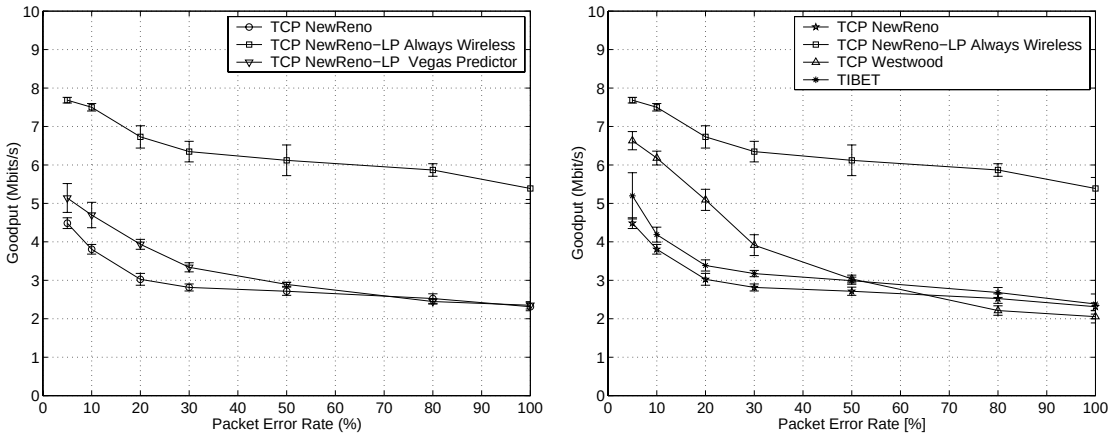
We then considered the same two-state Markov model described in Chapter 5 to model correlated losses, and we measured the goodput achieved by TCP sources as a function of the packet error rate in the Bad state, to take into account various levels of fading. The results are reported in Figures 7.6(a) and 7.6(b).

These results confirm the improved performance achieved by TCP NewReno-LP even in this network scenario that models very closely real wireless link conditions.

7.1.3 Friendliness and Fairness

The TCP sources implemented in our test bed have been tested in an heterogeneous environment comprising connections running different TCP versions, and their level of friendliness and fairness has been measured.

To this purpose, we considered the same mixed scenario of Section 6.3.7, where 5 TCP connections using either the TCP source under investigation (namely TCP NewReno-LP, TIBET and TCP Westwood) or TCP NewReno share a 10 Mb/s link. By simulation we measured, for each connection, the goodput. The average goodputs of TCP NewReno-LP



(a) (b)

Figure 7.6: Goodput Achieved by various TCP versions in the presence of correlated losses.

and TCP NewReno connections are shown in Figure 7.7.

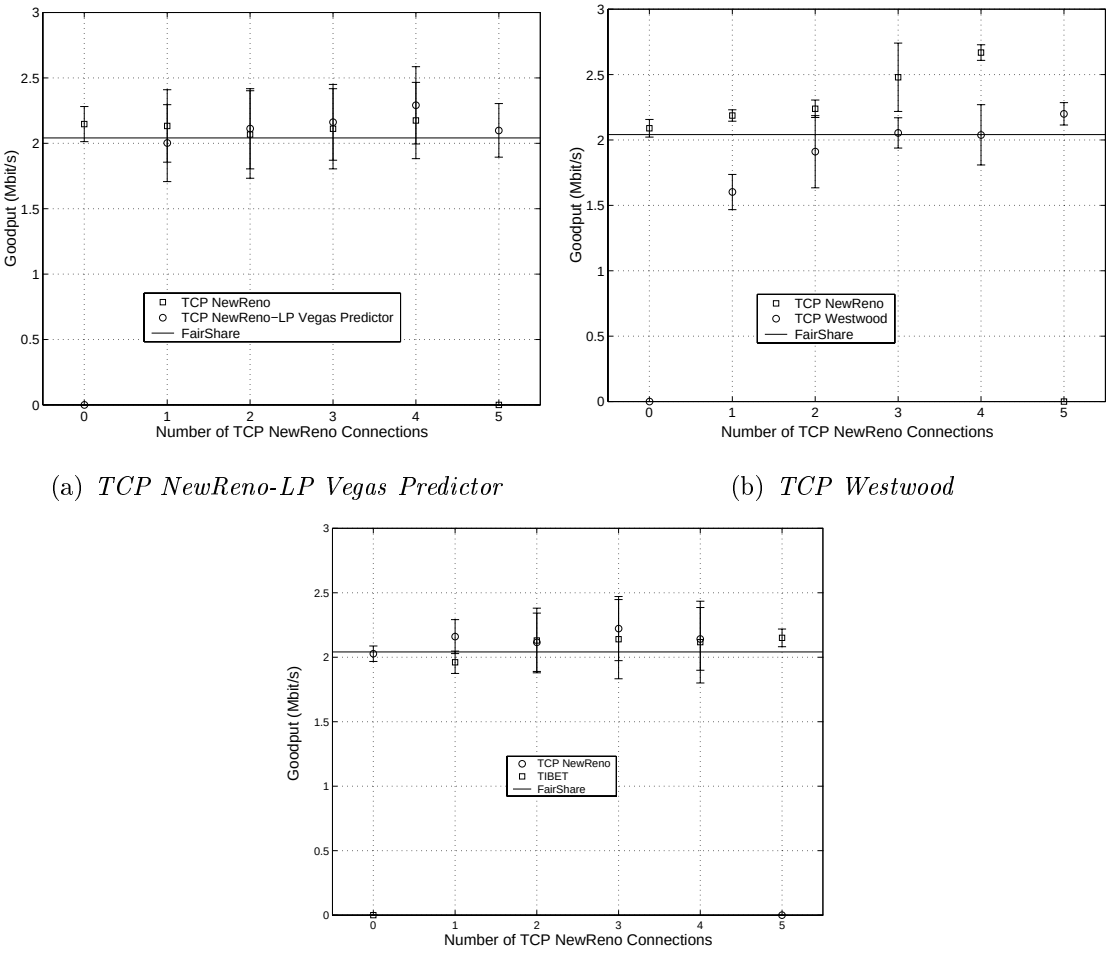
We notice that TCP NewReno-LP shares friendly network resources with classical TCP NewReno sources, thus allowing its smooth introduction in the Internet; the same result is achieved by TIBET, as shown in Figure 7.7(c). On the other hand, TCP Westwood shows an unfriendly behavior that prevents its deployment in actual networks.

Note that, in an homogeneous scenario, TCP NewReno-LP achieves the same level of fairness as TCP NewReno. This result is evident in Figure 7.7(a), where the average goodput achieved by 5 TCP NewReno-LP connections (the values on the extreme left) is practically coincident with that achieved by 5 TCP NewReno sources (extreme right).

7.1.4 Short-Lived TCP Connections

We studied the performance of TCP NewReno-LP also with short-lived connections, typical of HTTP transactions. More specifically, we considered the Web page of our laboratory (Advanced Network Technologies Laboratory, ANTLab), depicted in Figure 7.8.

We then performed an HTTP request to this page, using the test bed showed in Figure 7.1. Using NistNet we set the connection's RTT equal to 50 ms, a bottleneck link capacity equal to 10 Mbit/s and a Packet Error Rate that varies in the 0% to 12% range.



(a) *TCP NewReno-LP Vegas Predictor*

(b) *TCP Westwood*

(c) *TIBET*

Figure 7.7: Friendliness of TCP NewReno-LP, TIBET and TCP Westwood towards classical TCP NewReno.

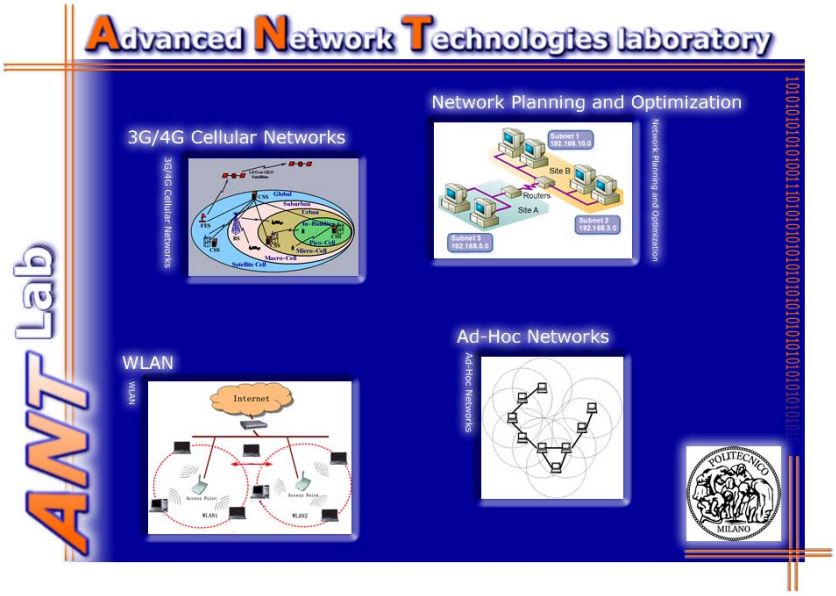


Figure 7.8: Web Page used to measure TCP performance with short-lived connections. The page contains 50 objects having average dimension equal to 90 Kbyte.

The download time of the Web page was measured and averaged over 50 independent trials. The results are reported in Figure 7.9 for the TCP versions considered in this Chapter.

As we already observed in Section 6.3.6, the performance of TCP NewReno-LP are in line with the other TCP versions. Note that, however, in some scenarios TCP NewReno-LP achieves a slight improvement over standard TCP NewReno.

7.2 Ad-hoc Network

To further analyze the behavior of the proposed TCP enhancements in a realistic wireless environment, we considered a test bed implementing an ad hoc network of n nodes that formed a linear chain containing $n-1$ wireless hops (a *String Network*, commonly considered in the literature [56]). The nodes used the 802.11 MAC protocol for medium access. Then, a one-way TCP data transfer was performed between the two nodes at the ends of the linear chain, and the TCP goodput was measured between these nodes.

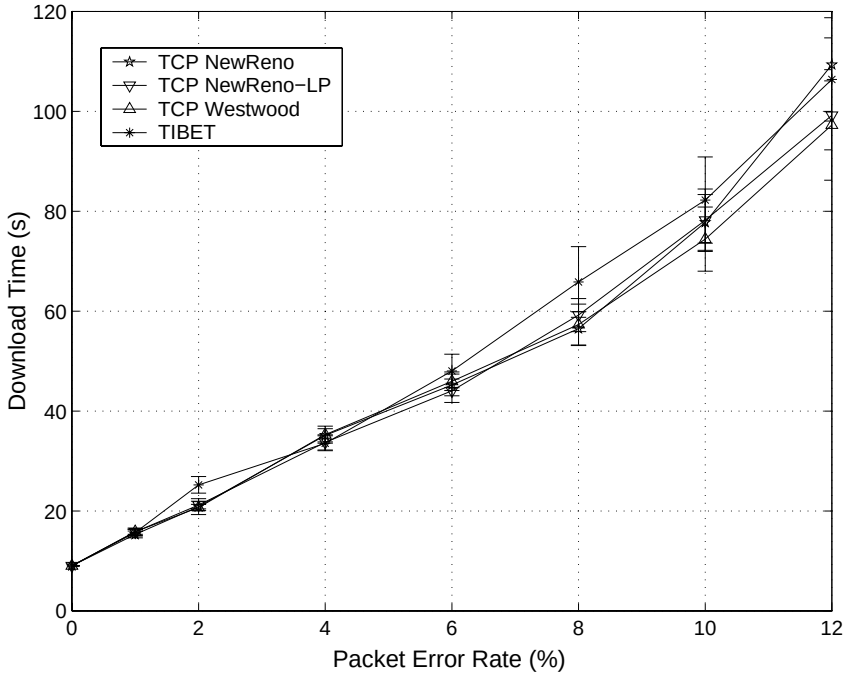


Figure 7.9: Average Download Time with short-lived connections.

The mobile hosts were placed to avoid that two non-adjacent terminals could reach each other with their transmissions. For this reason we used the software NetStumbler [61] to find the correct locations to place the terminals. NetStumbler is a software that allows to analyse the parameters of a Wireless LAN 802.11, giving accurate information concerning the quality of the radio link, namely the signal strength, the noise level and the Signal-to-Noise ratio (SNR).

Figure 7.10 shows a 3D rendering of the test location, the ANTLab laboratory in Milan, as well as the disposition of the terminals.

The same location is shown in Figure 7.11 in a bird-view way, putting in evidence the approximate radio coverage areas of the terminals.

Finally, Figure 7.12 shows the output of the *traceroute* command, that measures the delays experienced in the 5 hop ad hoc network.

On the first node of the string we installed the Linux kernel implementing the advanced TCP versions proposed in this work. Then the last node of the ad hoc string network performed a long-lived FTP file transfer with the server.

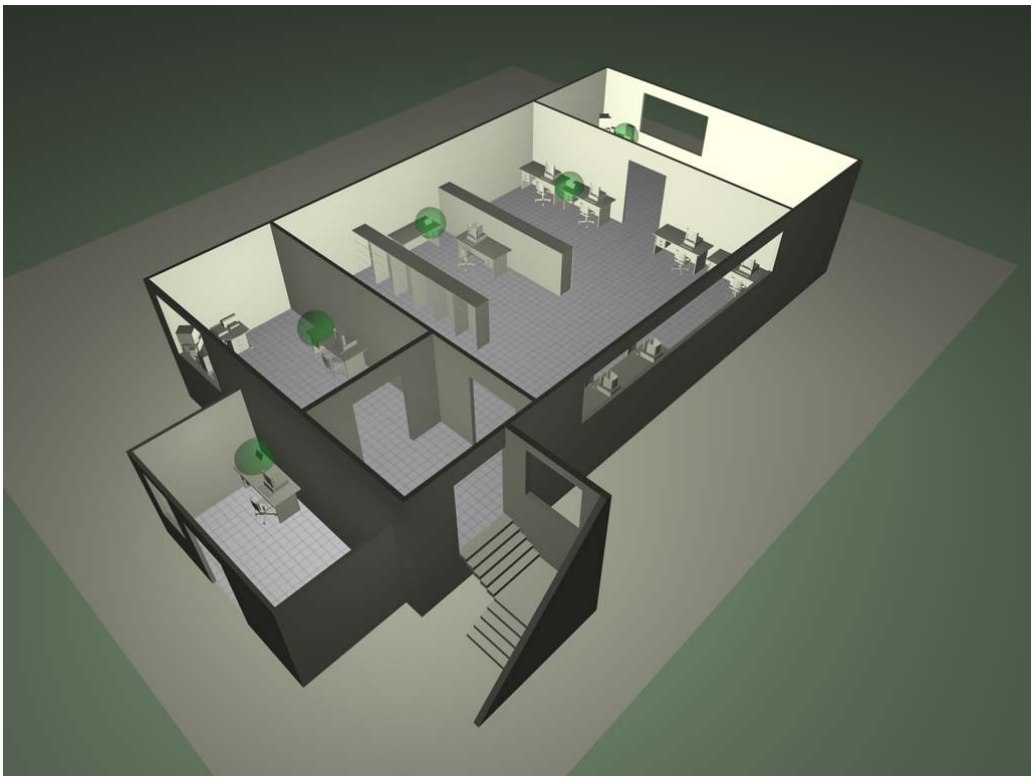


Figure 7.10: 3D-Rendering of the ANTLab laboratory used for the ad hoc network. The green circles show the position of the mobile terminals.

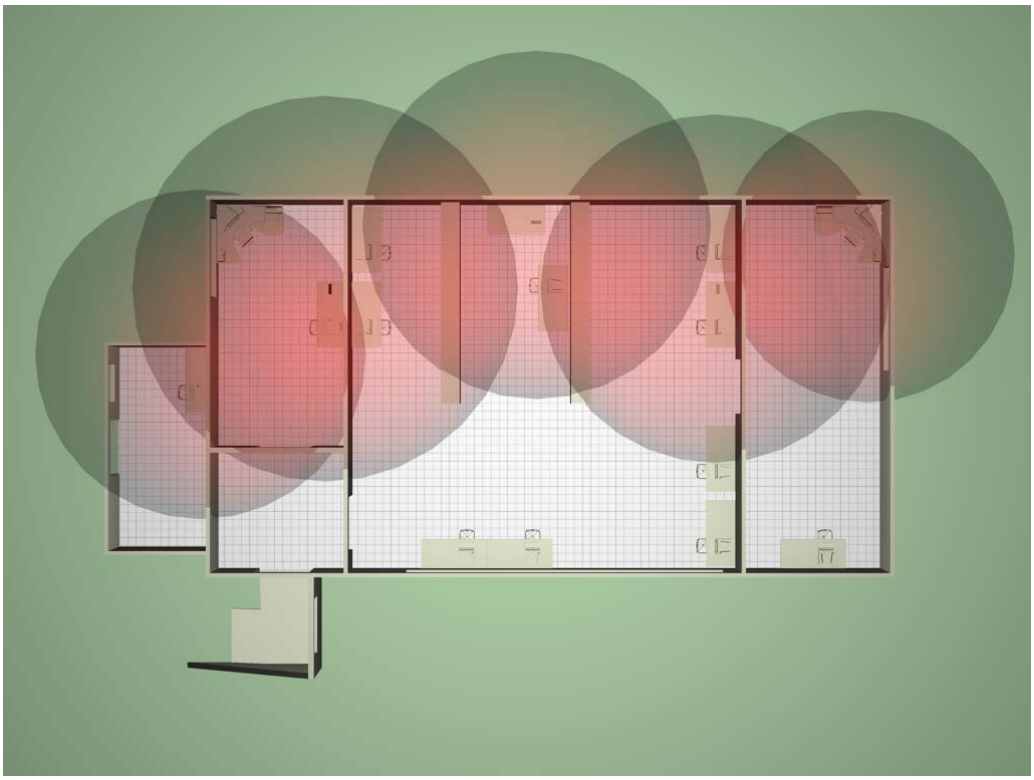


Figure 7.11: Bird-view of the ad hoc test bed, with the approximate radio coverage areas.

```
mobile_host1:~# traceroute -n 192.168.40.2
traceroute to 192.168.40.2 (192.168.40.2), 30 hops max, 38 byte packets

 1  192.168.10.2 (192.168.10.2)  1.477 ms  1.344 ms  1.322 ms
 2  192.168.20.2 (192.168.20.2)  3.335 ms  2.519 ms  2.485 ms
 3  192.168.30.2 (192.168.30.2)  4.579 ms  4.180 ms  3.937 ms
 4  192.168.40.2 (192.168.40.2)  6.243 ms  5.834 ms  5.445 ms
```

Figure 7.12: Output of the *traceroute* command showing the delays experienced in the Ad-Hoc string network.

We started with a 4-hop network (using all the 5 nodes) and progressively we eliminated the last nodes, one at the time, moving the client host closer to the server.

Figure 7.13 presents the measured TCP throughput as a function of the number of hops, averaged over ten runs.

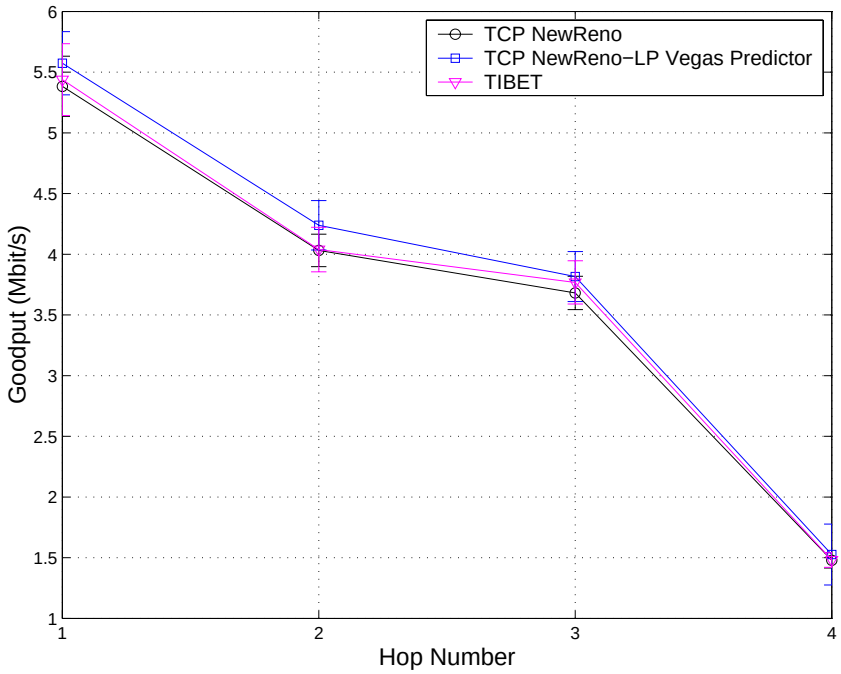


Figure 7.13: Goodput achieved in the Ad-hoc multihop network as a function of the number of hops.

Observe that the throughput decreases rapidly when the number of hops increases. This trend is similar to that reported in [56]. However, note that TCP NewReno-LP achieves a slight improvement with respect to the standard TCP NewReno version.

7.3 Cellular IP Network

Cellular IP [62] represents a new mobile host protocol that is optimized to provide access to a Mobile IP enabled Internet [63], in support of fast moving wireless hosts.

Cellular IP inherits cellular systems principles for mobility management, passive connectivity and handoff control, but is designed based on the IP paradigm. Three com-

ponents are defined in a Cellular IP network: the *gateway*, the *base stations* and the *mobile hosts*. Mobile hosts are connected to base stations, which are logically organized in a tree structure, whose head is represented by the gateway. Base stations serve as wireless access points but at the same time they route IP packets and integrate cellular control functionalities traditionally found in Mobile Switching Centers and Base Station Controllers. The Cellular IP network is connected to the Internet via a *gateway* router. Mobility between gateways (i.e., Cellular IP access networks) is managed by Mobile IP while mobility within access networks is handled by Cellular IP.

Figure 7.14 shows the architecture of our test bed, composed by three base stations ($BS1 - BS3$), the gateway (GW) and a single mobile host ($MH1$).

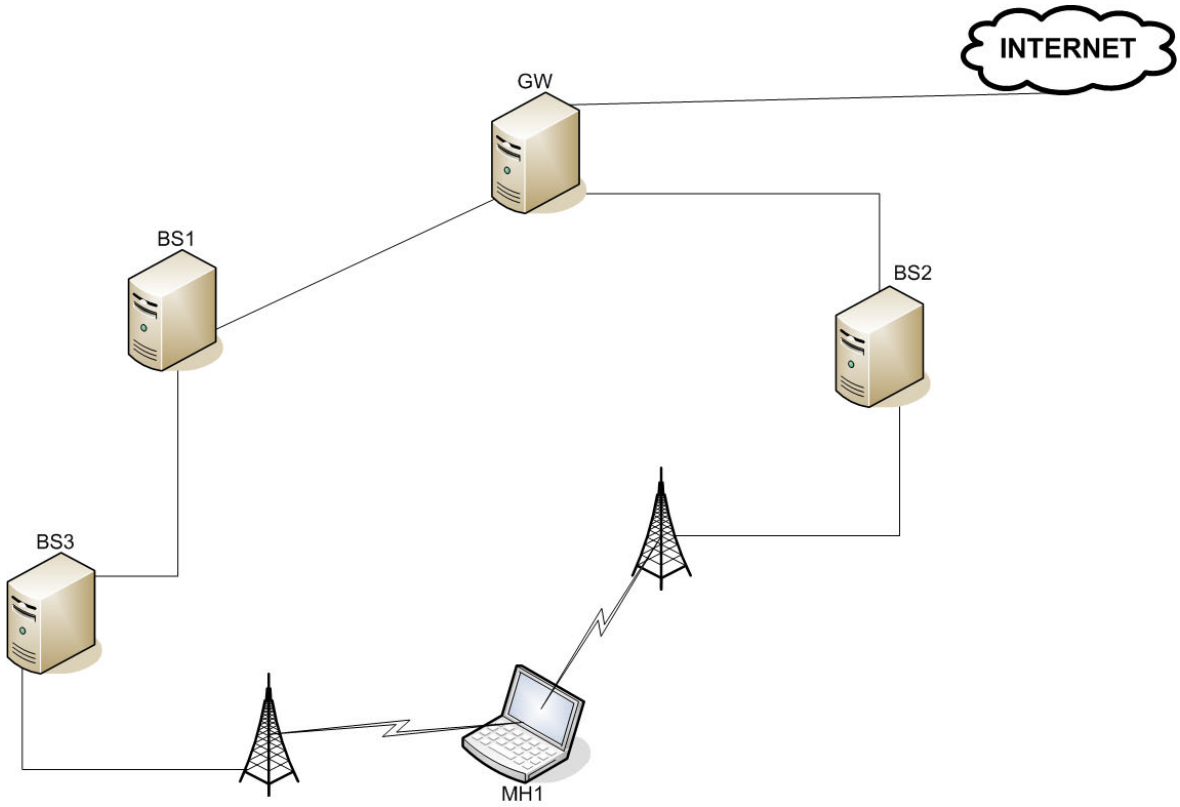


Figure 7.14: Cellular IP test bed topology.

$MH1$ moves between $BS2$ and $BS3$ and performs a bulk FTP file transfer with a PC Server, external to the Cellular IP network, that implements TCP NewReno, TCP

NewReno-LP, TIBET and TCP Westwood. The duration of the connection has been limited to 150 seconds, and its RTT is equal to 100 ms. The duration of the handoff experienced by $MH1$ has been set equal to 100 ms, since at each handoff $MH1$ needs to inform the gateway of its new association, and GW has to update its routing tables accordingly.

Then we measured the amount of data transferred during the connection's lifetime as a function of the frequency with which $MH1$ experiences the handoffs. The results are shown in Figure 7.15 for TCP NewReno, TCP NewReno-LP, TIBET and TCP Westwood.

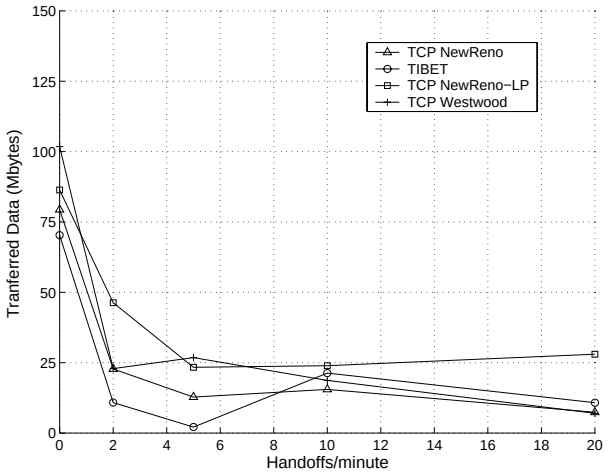


Figure 7.15: Goodput achieved by various TCP versions as a function of the handoff rate in a Cellular IP network.

Note that, even in this scenario, TCP NewReno-LP always performs better than the other TCP sources, even for high handoff rates, since it can distinguish with sufficient accuracy between losses due to congestion and losses due to handoffs. Note that TCP NewReno-LP can achieve a performance up to 100% higher than all the other TCP versions when the handoffs frequency increases, thus making it suitable for an utilization in a campus area network supporting IP mobility.

Chapter 8

Conclusions

In this work we have discussed, and analyzed, issues related to the use of enhanced bandwidth estimation and loss differentiation algorithms for TCP congestion control. These algorithms, differently from that used in TCP NewReno, 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 NewReno over wired links. We have discovered that a key to achieving both performance improvement and a fair behavior is to base the TCP operation on unbiased and accurate bandwidth estimates and loss differentiations.

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 NewReno. To overcome this problem we first proposed a new algorithm, TIBET, that performs an unbiased and accurate bandwidth estimation; based on TIBET, we then proposed enhancements to the Vegas, NCPLD, and Spike schemes, which achieve higher accuracy in all the considered network scenarios.

These algorithms have then been used to enhance the TCP error recovery mechanism, and it has been found that significant improvement is obtained both in simulated scenarios and in real networks.

We also defined ideal schemes that assume the exact value of the bandwidth and the exact cause of packet losses, and that provide, for all possible schemes based on these estimation approaches, an upper bound to the throughput.

Although there is still room for improvement in TCP performance, the above mentioned bounds show that the proposed TCP sources enhanced with bandwidth estimation and loss differentiation well approach these ideal bounds.

Bibliography

[1] A. Capone, L. Fratta, and F. Martignon. Bandwidth Estimation Schemes for TCP over Wireless Networks. *IEEE Transactions on Mobile Computing*, 3(2):129–143, 2004.

[2] S. Bregni, D. Caratti, and F. Martignon. Enhanced Loss Differentiation Algorithms for Use in TCP Sources over Heterogeneous Wireless Networks. In *Proceedings of GLOBECOM'03*, San Francisco, 1-5 Dec. 2003.

[3] A. Capone and F. Martignon. TCP with Bandwidth Estimation over Wireless Networks. In *Proceedings of VTC fall 2002*, September 2002.

[4] F. Martignon. Improving TCP Performance over Wireless Networks using Loss Differentiation Algorithms. In *IEEE Conference Mobile Wireless Ccommunication Networks MWCN 04*, October 2004.

[5] A. Capone and F. Martignon. Bandwidth Estimates in the TCP Congestion Control Scheme. In *Proceedings of Tyrrhenian International Workshop on Digital Communications (IWDC 2001)*, Taormina, Italy, Sep.2001.

[6] M. Mathis, J. Semke, and J. Mahdavi. The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm. *ACM Computer Communications Review*, 27(3), 1997.

[7] T. V. Lakshman and U. Madhow. The performance of TCP/IP for networks with high bandwidth-delay products and random loss. *IEEE/ACM Transactions on Networking*, 5(3):336–350, 1997.

[8] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz. A Comparison of Mechanisms for Improving TCP Performance over Wireless Links. In *IEEE/ACM Transactions on Networking*, volume 5(6), pages 759–769, December 1997.

[9] S. Keshav. A control-theoretic approach to flow control. In *Proceedings of ACM SIGCOMM'91*, pages 3–15, September 1991.

[10] J. C. Hoe. Improving the Start-up Behavior of a Congestion Control Scheme for TCP. *ACM SIGCOMM Computer Communications Review*, 26(4):270–280, October 1996.

[11] M. Allman and V. Paxson. On Estimating End-to-End Network Path Properties. In *Proceedings of ACM SIGCOMM'99*, 1999.

[12] L.S. Brakmo and L.L. Peterson. TCP Vegas: End-to-End Congestion Avoidance on a Global Internet. *IEEE Journal on Selected Areas in Communications*, 13(8):1465–1480, October 1995.

[13] C. Casetti, M. Gerla, S. Mascolo, M. Y. Sanadidi, and R. Wang. TCP Westwood: End-to-End Congestion Control for Wired/Wireless Networks. In *Wireless Networks Journal*, volume 8, pages 467–479, 2002.

[14] R. Wang, M. Valla, M.Y. Sanadidi, and M. Gerla. Adaptive Bandwidth Share Estimation in TCP Westwood. In *Proceedings of Globecom'02*, 2002.

[15] S. Cen, P. Cosman, and G. Voelker. End-to-End Differentiation of Congestion and Wireless Losses. In *Proceedings of ACM Multimedia Computing and Networking 2002*, volume 4673, pages 1–15, San Jose, CA, January 2002.

[16] D. Barman and I. Matta. Effectiveness of Loss Labeling in Improving TCP Performance in Wired/Wireless Networks. In *Proceedings of the Tenth IEEE International Conference on Network Protocols*, November 2002.

[17] N. K. G. Samaraweera. Non-congestion packet loss detection for TCP error recovery using wireless links. In *Proceedings of IEEE Communications*, volume 146, 1999.

[18] G. Yang, R. Wang, M. Y. Sanadidi, and Mario Gerla. Performance of TCPW BR in Next Generation Wireless and Satellite Networks. Technical report, UCLA CS #200223, 2002.

[19] S. Biaz and N. H. Vaidya. Distinguishing Congestion Losses from Wireless Transmission Losses: A Negative Result. *Seventh International Conference on Computer Communications and Networks (IC3N), New Orleans*, 1998.

[20] Cheng Peng Fu and S. C. Liew. TCP Veno: TCP enhancement for transmission over wireless access networks. In *IEEE Journal on Selected Areas in Communications*, volume 21(2), pages 216–228, Feb. 2003.

[21] K. Xu, Y. Tian, and N. Ansari. TCP-Jersey for Wireless IP Communications. In *IEEE Journal on Selected Areas in Communication*, volume 22(4), May 2004.

[22] Eric Hsiao-Kuang Wu and Mei-Zhen Chen. JTCP: Jitter-Based TCP for Heterogeneous Wireless Networks. In *IEEE Journal on Selected Areas in Communication*, volume 22(4), May 2004.

[23] M.Allman, V.Paxson, and W.Stevens. TCP Congestion Control. *RFC 2581*, April 1999.

[24] J. Mogul and S. Deering. Path MTU Discovery. *IETF RFC 1191*, November 1990.

[25] V. Jacobson. Congestion avoidance and control. *ACM Computer Communications Review*, 18(4):314–329, 1988.

[26] V. Paxson. End-to-End Internet Packet Dynamics. *IEEE/ACM Transactions on Networking*, 7(3):272–292, 1997.

[27] S. Floyd and T. R. Henderson. The NewReno Modifications to TCP's Fast Recovery Algorithm. *IETF RFC 2582*, 26(4), April 1999.

[28] K.Fall and S.Floyd. Simulation-based Comparisons of Tahoe, Reno and SACK TCP. *Computer Communication Review*, July 1996.

[29] V. Paxson and M. Allman. Computing TCP's Retransmission Timer. *RFC 2988*, November 2000.

[30] J. C. Mogul. Observing TCP Dynamics in Real Networks. In *Proceedings of ACM SIGCOMM'92 Symposium on Communications Architectures and Protocols*, pages 305–317.

[31] L. Zhang, S. Shenker, and D. Clark. Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic. In *Proceedings of SIGCOMM'91 Symposium on Communications Architectures and Protocols*, pages 133–147, Zurich, September 1991.

[32] J. Mo, V. Anantharam, R. J. La, and J. Walrand. Analysis and Comparison of TCP Reno and Vegas. In *Proceedings of ACM GLOBECOMM'99*, 1999.

[33] C. Dovrolis, P. Ramanathan, and D. Moore. What do packet dispersion techniques measure? . In *Infocom 2001*, volume 2, pages 905–914, 2001.

[34] C. Dovrolis and M. Jain. End-to-End Available Bandwidth: Measurement Methodology, Dynamics, and Relation with TCP Throughput. In *Sigcomm 2002*, August 2002.

[35] S. Mascolo, C. Casetti, M. Gerla, S.S. Lee, and M. Sanadidi. TCP Westwood: Congestion Control with Faster Recovery. Technical report, UCLA CS Technical Report #200017, 2000.

[36] R. Wang, M. Valla, M.Y. Sanadidi, B. Ng, and M. Gerla. Efficiency/Friendliness Tradeoffs in TCP Westwood. In *IEEE Symposium on Computers and Communications*, Taormina, Italy, July 2002.

[37] ns-2 network simulator (ver.2).LBL. URL: http://www.isi.edu/nsnam.

[38] I. Stoica, S. Shenker, and H. Zhang. Core-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocations in High Speed Networks. In *Proceedings of ACM SIGCOMM'98*, Vancouver, Canada, September 1998.

[39] L. A. Grieco, S. Mascolo, and R. Ferorelli. Additive-Increase/Adaptive-Decrease Congestion Control: a Mathematical Model and Its Experimental Validation. In *IEEE International Symposium on Computer and Communications, Giardini Naxos, Italy*, July 1-4, 2002.

[40] S. Floyd, M. Mathis, J. Mahdavi, and A. Romanow. TCP Selective Acknowledgement Option. *RFC 2018*, April 1996.

[41] M. Zorzi, A. Chockalingam, and R. R. Rao. Throughput Analysis of TCP on Channels with Memory. In *IEEE Journal on Selected Areas in Communications*, volume 18, pages 1289–1300, July 2000.

[42] A. Chockalingam and M. Zorzi. Wireless TCP Performance with Link Layer FEC/ARQ. In *Proceedings of IEEE International Conference on Communications, ICC'99*, volume 2, pages 1212–1216, June 1999.

[43] A. A. Abouzeid, S. Roy, and M. Azizoglu. Stochastic Modeling of TCP over Lossy Links. In *Proceedings of INFOCOM 2000*, Tel Aviv, Israel, March 2000.

[44] D. Bertsekas and R. Gallagher. *Data Networks*. Prentice-Hall, Englewood Cliffs, NJ, 1987.

[45] M. Kim and B. Noble. Mobile Network Estimation. In *Proceedings of the Seventh Annual International Conference on Mobile Computing and Networking*, pages 298–309, July 2001.

[46] D. C. Montgomery. *Introduction to Statistical Quality Control*. John Wiley & Sons, New York, Englewood Cliffs, NJ, 1985.

[47] P. Bhagwat, P. Bhattacharya, A. Krishna, and S. K. Tripathi. Using channel state dependent packet scheduling to improve throughput over wireless channels. *Wireless Networks*, 3(1), 1997.

[48] S. Floyd and V. Paxson. Difficulties in Simulating the Internet. In *IEEE/ACM Transactions on Networking*, volume 9, pages 392–403, August, 2001.

[49] V. Tsaoussidis, H. Badr, X. Ge, and K. Pentikousis. Energy / Throughput Tradeoffs of TCP Error Control Strategies. In *Proceedings of the 5th IEEE Symposium on Computers and Communications (ISCC)*, 2000.

[50] K. Pawlikowski, H.-D.Joshua Jeong, and J.-S.Ruth Lee. On Credibility of Simulation Studies of Telecommunication Networks. *IEEE Communications Magazine*, pages 132–139, Jan. 2002.

[51] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Throughput: A Simple Model and its Empirical Validation. In *Proceedings of ACM SIGCOMM '98*, 1998.

[52] N. Cardwell, S. Savage, and T. Anderson. Modeling TCP Latency. In *Proceedings of INFOCOM 2000*, pages 1742–1751, 2000.

[53] M. Mellia, I. Stoica, and H. Zhang. TCP Model for Short Lived Flows. *IEEE Communications Letters*, 6(2):85 – 88, February 2002.

[54] R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling.* Wiley, New York, 1991.

[55] *Monarch Project.* Available at http://www.monarch.cs.rice.edu/.

[56] G. Holland and N. Vaidya. Analysis of TCP Performance over Mobile Ad Hoc Networks. *Wireless Networks*, 8:275–288, March - May 2002.

[57] W. Navidi and T. Camp. Stationary Distributions for the Random Waypoint Mobility Model. *IEEE Transactions on Mobile Computing*, 3(1):99–108, January - March 2004.

[58] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc On-Demand Distance Vector (AODV) Routing. *RFC 3561*, July 2003.

[59] T. Clausen and P. Jacquet. Optimized Link State Routing Protocol (OLSR). *RFC 3626*, October 2003.

[60] *NIST Net Home Page.* Available at http://snad.ncsl.nist.gov/itg/nistnet/.

[61] NetStumbler. Available at: http://www.netstumbler.com/.

[62] A. T. Campbell, J. Gomez, S. Kim, Z. Turanyi, C.-Y. Wan, , and A. Valko. Design, Implementation and Evaluation of Cellular IP. *IEEE Personal Communications*, 7(4):42–49, 1999.

[63] C. Perkins. IP Mobility Support for IPv4. *IETF RFC3344*, August 2002.