INTERNATIONAL JOURNAL OF SATELLITE COMMUNICATIONS AND NETWORKING
Int. J. Satell. Commun. Network. 2004; **22**:547–566 (DOI: 10.1002/sat.799)

TCP Hybla: a TCP enhancement for heterogeneous networks

Carlo Caini*,† and Rosario Firrincieli

DEIS/ARCES, Bologna University, Viale Risorgimento 2, 40136, Bologna, Italy

SUMMARY

In heterogeneous networks, TCP connections that incorporate a terrestrial or satellite radio link are greatly disadvantaged with respect to entirely wired connections, because of their longer round trip times (RTTs). To cope with this problem, a new TCP proposal, the TCP Hybla, is presented and discussed in the paper. It stems from an analytical evaluation of the congestion window dynamics in the TCP standard versions (Tahoe, Reno, NewReno), which suggests the necessary modifications to remove the performance dependence on RTT. TCP Hybla performance is firstly evaluated in the case of an ideal channel, with good correlation between analytical and simulation data. Then, more realistic situations, which require the adoption of a benchmark network topology and a careful ns-2 simulation set-up, are examined. In particular, TCP Hybla performance is compared with that achievable by TCP standard in the presence of congestion and link losses, either separately or jointly considered. In all the examined cases, the superiority of TCP Hybla is evident, as it greatly reduces the severe penalization suffered by wireless, and especially satellite, TCP connections. Finally, it is worth noting that TCP Hybla does not infringe the end to end semantics of TCP and is compatible with other promising enhancements. Copyright © 2004 John Wiley & Sons, Ltd.

KEY WORDS: TCP/IP; congestion control; wireless communications

1. INTRODUCTION

The current versions of the TCP protocol (Tahoe, Reno, NewReno) suffer performance problems in connections characterized by relatively high link error rates and long propagation delays, such as those that encompass terrestrial and satellite radio links. As the TCP protocol was basically designed to recover only from congestion situations, losses of TCP segments due to errors in the transmission link are erroneously ascribed to congestion problems, causing a totally inappropriate activation of the congestion avoidance mechanisms [1, 2]. This problem can be addressed either through proper modifications of TCP protocol or, rather simplistically, by enforcing the reliability of lower layers, when possible [3]. However, it must be stressed that even in the case of an errorless channel, radio connections would be still penalized by long propagation delays. This is due to the TCP window-based transmission algorithm which, being triggered by acknowledgment (ACK) arrivals, depends on network delays. Long round trip times (RTTs), reduce the congestion window (cwnd) growth rate and result in a significant

*Correspondence to: Carlo Caini, DEIS/ARCES, Bologna University, Viale Risorgimento 2, 40136, Bologna, Italy.
†E-mail: ccaini@deis.unibo.it

throughput degradation, as well as an unfair sharing of the available bandwidth resources between wired connections and connections that incorporate at least one radio link (usually characterized by much longer RTTs especially when satellite legs are involved [3]). In this case, the performance degradation is a direct consequence of the intrinsic behaviour of the TCP protocol and cannot be resolved without introducing proper modifications of the TCP congestion control algorithm. This is the aim of the TCP Hybla proposal presented in this paper.

In recent years, several solutions have been proposed to improve TCP performance both over wired and wireless links. Some suggestions focus on limited modifications of standard procedures and/or on tuning TCP parameters, whereas others envisage the introduction of alternative TCP procedures.

With reference to the former category, in Reference [4], the advantages of a larger slow start initial window (SS-IW) are pointed out, while in Reference [5] modifications of the SS threshold and packet spacing are envisaged to prevent early buffer overflow. Along the same lines, in Reference [6] two minor modifications to the fast retransmit algorithm are suggested, as in Reference [7] the adoption of the SACK (Selective ACK) option, to quickly recover from multiple losses.

The latter category of proposals is characterized by the introduction of major TCP modifications. TCP Vegas [8] aims at preventing the congestion losses by exploiting a dynamic estimation of the available bandwidth. Although positive results may be achieved in wired networks, the maximum cwnd increase rate is left unaltered with respect to the standard, leaving basically unresolved the performance penalization suffered by long RTT connections. TCP-Peach [9] is based on the replacement of slow start and fast recovery algorithms with "sudden start" and "rapid recovery" procedures, which rely on the introduction of 'dummy' segments to probe the bandwidth availability of the network. TCP-Peach requires all the routers along the connection to implement some priority mechanism at the IP layer, to discard dummy segments in presence of congestion. Another interesting proposal is the TCP Westwood [10], which introduces a modification of the fast recovery algorithm called "faster recovery". In contrast with the TCP standard, which halves the congestion window after three duplicate ACKs and 'blindly' fixes the slow start threshold to it, TCP Westwood attempts to select a slow start threshold more consistent with the actual available bandwidth. Although it may offer several advantages in presence of link errors, it does not address the TCP bias against connections with long RTTs. Finally, other proposals rely either on splitting the end to end connections to isolate the wireless legs [11], or on modification of the lower layers (TCP spoofing) [12]. Although promising, they intrinsically lead to infringement of the end to end semantics of TCP, which presents several disadvantages on security and privacy issues [9]. In conclusion, although a lot of interesting contributions have been presented in literature, a comprehensive solution to the performance disparity due to different RTTs is far from being reached.

The TCP Hybla protocol presented in this paper aims at providing and assembling a set of techniques to solve the aforementioned RTT disparity problem. A key element is the modification of the standard rules for the congestion window increase, obtained by following the indications of an analytical study [13] of the congestion window dynamics. The enhancements introduced can be considered as an extension of the 'constant-rate' additive increase policy, presented by Floyd in Reference [14] and more recently discussed in Reference [15]. Besides, it will be shown that, in presence of losses due to congestion or link errors, the proposed modifications of the cwnd growth policy highly benefit from both the adoption of the SACK option and the use of timestamps, which by contrast provide only a limited gain in satellite connections when associated with TCP standard. Further important advantages are also obtained by implementing packet spacing techniques, which, by removing the transmission burstiness, reduce the probability of buffer overflow at intermediate routers and allows TCP Hybla to avoid the limitations introduced in Reference [15] on the original constant-rate algorithm.

Numerical results, obtained both analytically and/or by means of the ns-2 simulator [16], show that a large improvement of long RTT connections performance can be effectively achieved, without penalizing the overall network performance and, as the Hybla proposal requires only end point modifications, without infringing the end to end semantics of the TCP protocol.

The paper is organized as follows. In Section 2, the congestion control and the loss recovery algorithms of TCP standard protocol are analysed. In Section 3, the TCP Hybla proposal is introduced and a preliminary performance evaluation on an ideal channel is given. In Section 4 the benchmark network topology and the simulation set-up are specified. In Section 5, simulation results referring to congestion and link losses are presented and discussed, while in Section 6 the fairness and friendliness of TCP Hybla and TCP Newreno are investigated. Finally, some issues related to the RTT are discussed in Section 7 and conclusions are drawn in Section 8.

2. TCP ALGORITHM (TAHOE, RENO AND NEWRENO)

When a new connection is established, the sender probes for bandwidth availability by gradually increasing the congestion window value W. In the slow start phase, starting from an initial value W_0, typically equal to one or two maximum segment size (MSS), the congestion window is increased by MSS bytes per non-duplicate ACK received. When the congestion window reaches the value of the slow start threshold (ssthresh), the source switches to the congestion avoidance (CA) phase, during which the congestion window is increased by MSS/W bytes per new ACK received [17]. In a real channel this rise continues until either the size of receiver buffer (advertised window) is reached, or a segment is lost. In this case, action is taken following a recovery procedure that depends on the specific TCP version adopted [18].

2.1. *Slow start and congestion avoidance algorithms*

By expressing the value W of the congestion window in MSS units, the standard cwnd update rules previously described are given by

$$W_{i+1} = \begin{cases} W_i + 1, & \text{SS} \\ W_i + 1/W_i, & \text{CA} \end{cases} \tag{1}$$

where the index i denotes the reception of the ith ACK. Finally, note that the actual transmission of TCP segments occurs in relation to a window size given by min(cwnd, advertised window), where the advertised window corresponds to the receiver buffer size [19].

Now, to carry out an analytical investigation of the impact of RTT on TCP congestion control, it is convenient to temporarily consider an ideal channel (i.e. without losses), and to adopt a continuous model [20, 21] to describe the congestion window evolution in time. To this end, let us first note that in the SS phase the cwnd update rule (1) results in a discrete exponential increase with RTT, as the congestion window is doubled at every RTT, while in the

CA phase it leads to an approximately linear increase with time, as the growth is of about MSS bytes per RTT. Then, denoting by $W(t)$ the cwnd expressed in segments, and by t_γ the time at which the ssthresh value, γ, is reached, we have

$$W(t) = \begin{cases} 2^{t/\mathrm{RTT}}, & 0 \leqslant t < t_\gamma, \quad \mathrm{SS} \\ \dfrac{t - t_\gamma}{\mathrm{RTT}} + \gamma, & t \geqslant t_\gamma, \quad \mathrm{CA} \end{cases} \tag{2}$$

where $t_\gamma = \mathrm{RTT} \log_2 \gamma$. From (2) it is evident that the lower the RTT, the higher the congestion window increase rate.

The value of the congestion window in bytes, $W_{\mathrm{B}}(t)$, can be obtained by multiplying $W(t)$ by the sender MSS (under the assumption that there are always enough bits to fill whole MSS segments). Figure 1 shows $W_{\mathrm{B}}(t)$ for three different RTTs, having assumed in the example $\mathrm{MSS} = 1024$ bytes and $\gamma = 32$ (this relatively low value for realistic links is chosen only for the sake of clearness in the graphic representation). Note that, as a consequence of different RTTs, the standard congestion control algorithm causes different congestion window growth rates and different values for t_γ. By inspection of the figure, it is apparent the severe penalization that affects the slower connections, whose cwnd in practice increases at a longer time scale.

For performance comparison, it is useful to calculate the amount of data transmitted by a standard TCP source from the transmission start, $T_d(t)$. To this end, let us first introduce the expression of the segment transmission rate, i.e. the amount of segments transmitted per second,

$$B(t) = W(t)/\mathrm{RTT} \tag{3}$$

From (3) it is straightforward to calculate $T_d(t)$ as

$$T_d(t) = \int_0^t B(\tau)\,\mathrm{d}\tau = \begin{cases} \dfrac{2^{t/\mathrm{RTT}} - 1}{\ln(2)}, & 0 \leqslant t < t_\gamma, \quad \mathrm{SS} \\ \dfrac{\gamma - 1}{\ln(2)} + \dfrac{(t - t_\gamma)^2}{2\mathrm{RTT}^2} + \dfrac{\gamma(t - t_\gamma)}{\mathrm{RTT}}, & t \geqslant t_\gamma, \quad \mathrm{CA} \end{cases} \tag{4}$$

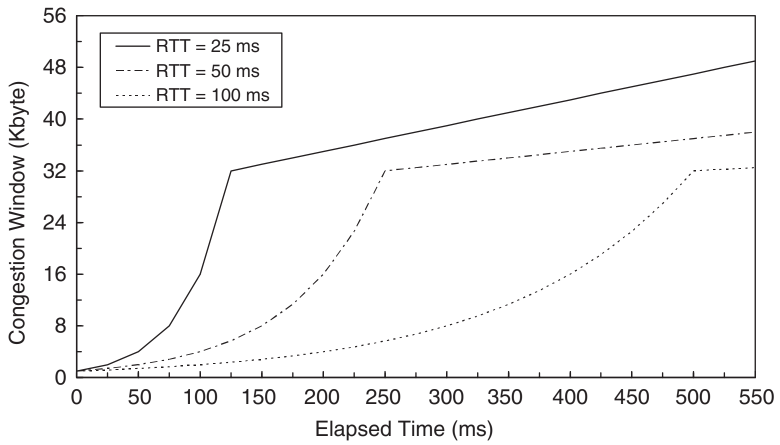

Figure 1. TCP standard: congestion window evolution in time for several RTT values.

Some curves obtained from (4) will be presented later. Now, for the following discussion, it is essential to note that because of the presence of RTT in $B(t)$, the amount of data transmitted in the connections with longer RTTs would be lower *even if* the congestion window evolution in time $W(t)$ was the same. As a consequence, in order to make the instantaneous transmission rate, $B(t)$, invariant with RTT, it is necessary not to compensate, but to *overcompensate* the different congestion window evolutions in time resulting from different RTTs, as explained in the TCP Hybla section.

2.2. Loss recovery mechanisms

The various versions of TCP essentially differ on the loss recovery mechanism they implement to tackle packet losses present in real channels [17, 22]. Here we will limit ourselves to recall the main features of the TCP NewReno *recovery phase* [18], which is basically maintained by TCP Hybla, but the mandatory implementation of a couple of additional features.

Let us start assuming that the segment n is dropped. As a consequence, a new duplicate acknowledgement (dupACK) for the segment $n-1$ is generated by the receiver every time a new segment arrives. The reception of the third dupACK triggers in the sender the fast retransmit and the fast recovery procedures, starting the recovery phase. First, the segment n is retransmitted, then the ssthresh is approximately updated[§] to the half of the cwnd value before the loss detection (*old_cwnd*), and the cwnd is reduced to *ssthresh* plus 3 MSS. After that, each additional dupACK increments the cwnd by MSS and triggers the transmission of a new segment if *cwnd* exceeds the *old_cwnd*. The reception of the first non-duplicate ACK may determine two consequences. If the ACK is 'partial', i.e. if it confirms only a part of the segments that were in flight when the loss was detected, the recovery phase is not terminated, the first unacknowledged segment is retransmitted, the cwnd is deflated by the amount of the data acknowledged, and a new segment is sent if allowed by the new cwnd value. Moreover, the first partial ACK arrival during the recovery phase resets the retransmission timer. If, however, the ACK confirms all the segments in flight when the loss was detected, the recovery phase ends, the cwnd is deflated to the ssthresh value, and the sender restarts transmission in congestion avoidance phase.

If multiple losses occur in the same window, NewReno remains in the recovery phase until either all of the lost packets have been recovered or a retransmission time out (RTO) expires. As NewReno is able to recover only one packet per RTT [18], the time spent in the recovery phase can become excessive [23] for long RTTs penalizing the wireless connections throughput.

3. TCP HYBLA ALGORITHM

3.1. Slow start and congestion avoidance algorithms

The basic idea of the TCP Hybla is to obtain for long RTT connections (e.g. satellite and wireless) the same instantaneous transmission rate, $B(t)$, of a comparatively fast reference TCP connection (e.g. wired). Equation (3) suggests that this goal can be achieved in two steps: first, by making $W(t)$ independent of RTT through a time scale modification, then by compensating the effect of the division by RTT. Both the steps can be better described after introducing

[§]More accurately, to no more than max(FlightSize/2,2MSS), where FlightSize is the number of segments sent but not acknowledged yet.

a normalized round trip time, ρ, defined as

$$\rho = \mathrm{RTT}/\mathrm{RTT}_0 \tag{5}$$

where RTT_0 is the round trip time of the reference connection to which we aim to equalize our performance. The former step is performed by multiplying by ρ the time (or the time elapsed from the reaching of the *ssthresh*), achieving a $W(t)$ independent of RTT. The latter, by multiplying by ρ the congestion window resulting at the first step (including the original ssthresh, γ), to have $B(t)$ independent of RTT. In conclusion we have

$$W^{\mathrm{H}}(t) = \begin{cases} \rho 2^{\rho t/\mathrm{RTT}}, & 0 \leqslant t < t_{\gamma,0}, \quad \mathrm{SS} \\ \rho\left[\rho \dfrac{t - t_{\gamma,0}}{\mathrm{RTT}} + \gamma\right], & t \geqslant t_{\gamma,0}, \quad \mathrm{CA} \end{cases} \tag{6}$$

where the superscript H identifies the Hybla proposal. As a consequence of the modification introduced in the second step, the switching time $t_{\gamma,0}$, defined as the time at which the congestion window reaches the value $\rho\gamma$, is the same for every RTT, being $t_{\gamma,0} = \mathrm{RTT}_0 \log_2 \gamma$. The congestion window evolution in time given by (6) is presented in Figure 2, for the same RTT values already considered in Figure 1, and having chosen $\mathrm{RTT}_0 = 25$ ms, for comparison purposes.

From (6), the segment transmission rate $B^{\mathrm{H}}(t) = W^{\mathrm{H}}(t)/\mathrm{RTT}$ of a TCP Hybla assumes the following expression:

$$B^{\mathrm{H}}(t) = \begin{cases} \dfrac{2^{t/\mathrm{RTT}_0}}{\mathrm{RTT}_0}, & 0 \leqslant t < t_{\gamma,0}, \quad \mathrm{SS} \\ \dfrac{1}{\mathrm{RTT}_0}\left[\dfrac{t - t_{\gamma,0}}{\mathrm{RTT}_0} + \gamma\right], & t \geqslant t_{\gamma,0}, \quad \mathrm{CA} \end{cases} \tag{7}$$

which is clearly independent of RTT and equal to the segment transmission rate of the reference TCP standard connection.

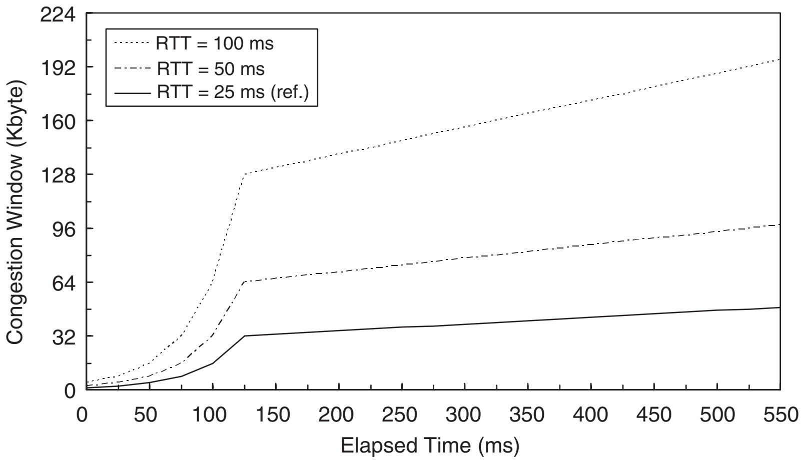

Figure 2. TCP Hybla: congestion window evolution in time for several RTT values.

From (7) it is simple to derive the analytical expression of the amount of segments transmitted from the beginning of the TCP Hybla connection,

$$T_d^{\mathrm{H}}(t) = \int_0^t B^{\mathrm{H}}(\tau)\,\mathrm{d}\tau = \begin{cases} \dfrac{2^{t/\mathrm{RTT}_0} - 1}{\ln(2)}, & 0 \leqslant t < t_{\gamma,0}, \quad \mathrm{SS} \\ \dfrac{\gamma - 1}{\ln(2)} + \dfrac{(t - t_{\gamma,0})^2}{2\mathrm{RTT}_0^2} + \dfrac{\gamma(t - t_{\gamma,0})}{\mathrm{RTT}_0}, & t \geqslant t_{\gamma,0}, \quad \mathrm{CA} \end{cases} \tag{8}$$

From (8), it is clear that TCP Hybla performance does not depend on the actual value of RTT, anymore. This observation is confirmed by numerical results reported in Figure 3, where TCP standard and TCP Hybla performance are compared for different RTT values, under ideal channel conditions. While the amount of transmitted data in TCP standard connections decays with the RTT, in TCP Hybla all the connections present the same performance of the reference one.

As far as the feasibility of the proposed approach is concerned, let us observe that the Hybla congestion window dynamic can be obtained by modifying the congestion window update rules (1) as follows:

$$W_{i+1}^{\mathrm{H}} = \begin{cases} W_i^{\mathrm{H}} + 2^{\rho} - 1, & \mathrm{SS} \\ W_i^{\mathrm{H}} + \rho^2/W_i^{\mathrm{H}}, & \mathrm{CA} \end{cases} \tag{9}$$

maintaining the ACK-based mechanism for congestion window update of the TCP standard. Note that the CA update rule is similar to the constant-rate algorithm [14], which, on the other hand, did not consider the slow start phase. Another important difference is that in the actual implementation of (9) TCP Hybla sets to 1 the minimum value for ρ. In such a

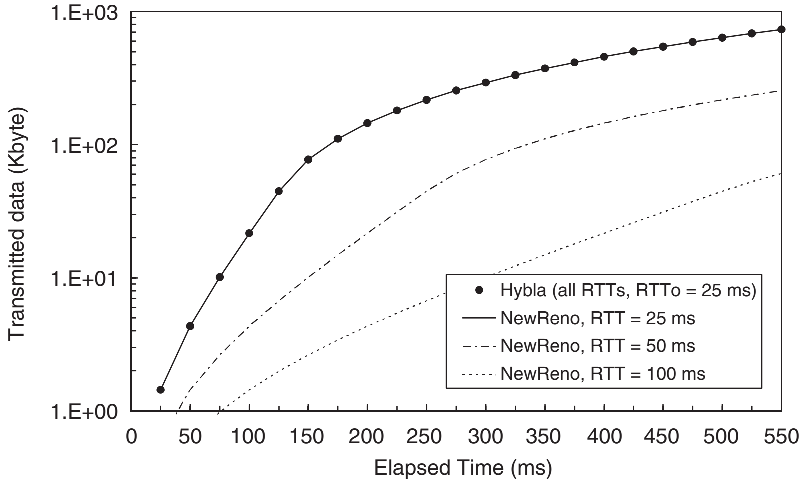

Figure 3. TCP Hybla vs. TCP standard: transmitted data for several RTT values in ideal (i.e. without losses) channel.

way, TCP Hybla simply behaves as the standard TCP for 'fast' connections (i.e. connections with $\mathrm{RTT} \leqslant \mathrm{RTT}_0$), which therefore maintain their standard increase rate. In addition to (9), both the initial *cwnd*[¶] and the original *ssthresh* must be multiplied by ρ, as well as the size of the transmission buffer (to manage the larger burstiness of TCP Hybla transmission). Finally, note that the expressions (7) and (9) hold true under the assumption that the sender is limited by the congestion window and not by the advertised window. If this assumption is not fulfilled, the limitation imposed by the receiver can be removed by increasing the advertized window by ρ, ensuring the same ceiling to the transmission rate for all connections. Note however, that this sole modification on the receiver side is not mandatory.

3.2. Loss recovery mechanism

Comparing the congestion control algorithm of TCP Hybla and TCP standard, it is evident that the former will result in a larger average cwnd than the latter. Consequently, the case of multiple losses in the same window will be more frequent, in particular when long RTT connections are considered. As this phenomenon may result in a very harmful slow down of the average transmission rate, the adoption of proper countermeasures is in order.

3.2.1. The SACK option. To overcome the TCP NewReno incapacity of recovering multiple losses in the same window, in Reference [7] the SACK option was proposed. With the SACK procedure, the data receiver can inform the sender about all the segments that have been successfully delivered, thus allowing the sender to recover more than one packet per RTT. In Reference [23], the performance of TCP Reno with SACK option and TCP NewReno were compared by means of simulations, and the SACK ability to recover from multiple losses in a shorter time was confirmed. Consequently, the SACK option is included and recommended in the TCP Hybla proposal.

3.2.2. Retransmission timeout and timestamps. Standard TCP uses a retransmission timer to ensure data delivery in the absence of any feedback from the remote receiver. The duration of this timer is referred to as RTO (retransmission time out) and its computation is based on both the current estimation of a smoothed value for RTT (SRTT) and an RTT variation (RTTVAR). These two parameters are calculated by the sender, deriving RTT estimations from the ACKs arrival time. Whenever the retransmission timer expires, the RTO is usually doubled ('exponential back-off' policy), until a new RTO value is computed. To this end it is necessary either to wait the ACK of the first non-retransmitted packet or to make use of timestamps [24]. The second alternative, largely implemented in the most widespread operating systems, is greatly preferable when dealing with large cwnds, where the number of the packet to be retransmitted after a time out event may excessively postpone the RTO update. We investigated this point by means of simulation, observing a severe penalization of longer RTT connections which, in unlucky circumstances, may often stall for an unacceptable prolonged time.

[¶]By contrast with IIW [4], in TCP Hybla the initial window increment is always proportional to the actual RTT. As a result, it is possible to adopt a larger initial cwnd without overfeeding the network, as a larger cwnd always compensates for a longer RTT, leaving $B(t)$ constant.

3.3. Other implementation features

3.3.1. Slow start threshold initial estimation. To prevent channel over feeding at the end of slow start phase, with the consequent burst of losses, the slow start threshold initial setting suggested in Reference [6] is implemented in TCP Hybla. In short, by observing the delay between the receptions of ACKs that corresponds to consecutively transmitted segments, it is possible to calculate at the connection start up a rough estimate of the channel bandwidth-delay product, from which to derive an appropriate initial slow start threshold value.

3.3.2. Burstiness and packet spacing. To obtain a satisfactory segment transmission rate, $B(t)$, in presence of long RTTs (i.e. an acceptable throughput), large cwnds must be used independently of the TCP version adopted, as can be deduced from (3). Under unfavourable circumstances, large cwnds may result in a bursty transmission, with a possible consequent performance degradation. However, the burstiness of the channel can be eliminated at the origin by spreading the cwnd segments over the RTT, as suggested in References [25, 26]. We have considered this possibility as a recommended option for the TCP Hybla, indicated as 'packet spacing'. Note that this technique eliminates the motivation of the upper bound on the cwnd CA increase set in Reference [15], when dealing with the constant-rate policy, and the consequent performance loss.

3.4. Performance evaluation on an ideal channel

To assess the accuracy of the continuous time model in representing the discrete ACK-based congestion control algorithm, a preliminary ns-2 simulation was performed considering an ideal channel (neither congestion nor link error losses), with a large bandwidth (1Gb/s) available. In Figure 4, simulation data are compared with some of the analytical results plotted in Figure 3. When the packet spacing option is not adopted, the amount of transmitted data provided by simulation increases at RTT steps, alternatively crossing the analytical curve (as a result of the bursty transmission). By contrast, when the packet spacing option is adopted, the simulation data presents a continuous slope (as the transmission burstiness has been removed), without crossing the analytical curve any more (due to the small delay in the data delivery necessarily introduced by the packet spacing option). In both the cases, the difference with the analytical data is very modest and tends to disappear after a short time. In conclusion, we can state that, independently of the packet spacing adoption, the accuracy of the continuous time analytical model is more than satisfactorily confirmed by simulation data.

When dealing with an ideal channel, it is worth noting that only the study of the connection start up is of interest, as the cwnd would tend rapidly to the receiver window for all the connections. For this reason, in Figures 3 and 4 performance has been evaluated in terms of total number of transmitted packets vs the elapsed time from the beginning of connections, instead of reporting the throughput (or goodput, see later), which, being an average value, is not functional in describing transient processes. By contrast, long connections and average values are of interest for evaluating performance on a real channel, with losses due to congestion and/or link errors, for which it is also necessary to define a network topology and a simulation environment.

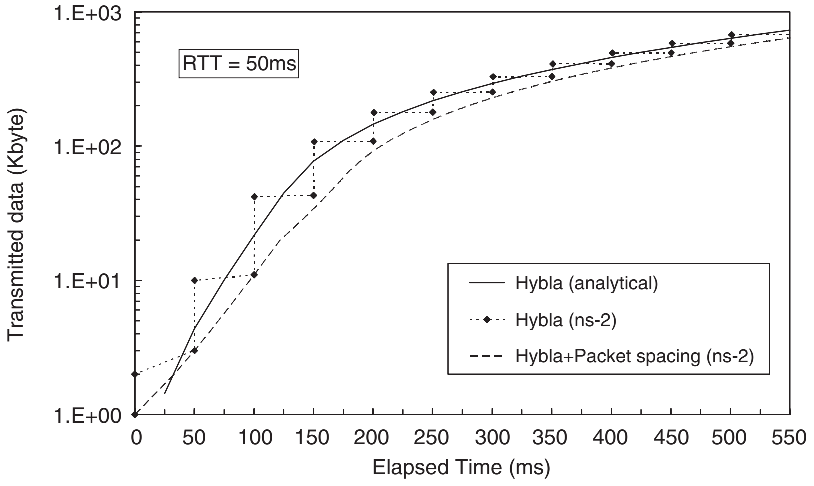

Figure 4. TCP Hybla: comparison between analytical and simulation data; transmitted data from the connection start up in ideal (i.e. without losses) channel.

4. TOPOLOGY AND SIMULATION SETUP

To insert the Hybla modifications in the ns-2 simulator, it was necessary to write a new software module, while for the standard variants of TCP we used the existing packages. The basic network topology considered in simulation runs is shown in Figure 5 (minor variants will be introduced whenever necessary). The foreground TCP connection is composed of both wired legs and a wireless link (e.g. a satellite link), while the background traffic is represented by 5 entirely wired connections. All the connections share a bottleneck link, whose limited bandwidth has been deliberately chosen to study the congestion effects, located between the routers R1 and R2. These follow a RED (random early detection) policy because of its better performance in ensuring fairness among competing TCP connections and in reducing the burstiness impact, even with moderate queues lengths ($qlen = 50$ seg., $\max_{\mathrm{th}} = 15$ seg., and $\min_{\mathrm{th}} = 5$ seg. in the simulation) [27], while all the hosts follow a simple DT (drop tail). It should also be considered that the RED policy is generally beneficial when dealing with simulations, as it reduces the variance of simulation results [14]. The two-way propagation delay of the wireless link varies in such a way that the RTT of the foreground connection ranges from 25 ms (considered only for comparison purposes with the wired connections) to 600 ms (corresponding to the case of a GEO satellite link). The wired links are supposed to be error free, while a uniformly distributed error model with a packet error rate (PER) in the range 0–5% is adopted for both the forward and the reverse links of the radio leg. All of the other link characteristics are reported in the figure.

For every TCP connection, a persistent FTP file transfer process, is considered. The performance is evaluated in terms of goodput, i.e. the amount of acknowledged packets, divided by the transfer process time (600 s). TCP Hybla requires an estimation of the wireless connection RTT to derive the suitable ρ value (RTT_0 is chosen in the simulations equal to the RTT of the wired connections, for the sake of comparison); this estimation is provided by all the TCP versions and it is also implemented in the ns-2 simulator. Finally, in the simulations the following variants of TCP protocols are examined:

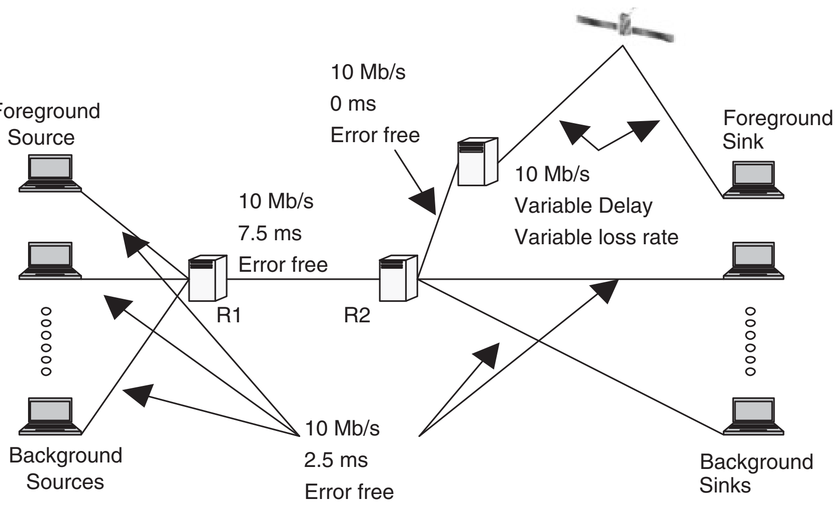

Figure 5. Benchmark network topology and simulation setup.

TCP NewReno: TCP NewReno is a collection of bug fixes and refinements of the TCP Reno Fast Recovery and Congestion Avoidance policies [18];

TCP SACK: the Reno Congestion Avoidance algorithm is combined with the SACK option for loss recovery [7];

TCP Vegas: a proposed variant of the TCP standard aiming at preventing congestion losses by means of a dynamic estimation of the available bandwidth [8];

TCP Hybla without SACK: a partial version of the present proposal, based on TCP NewReno, which does not include the SACK option;

TCP Hybla: the present proposal;

The advantages provided by packet spacing will be evaluated separately from the other protocol modifications. Therefore, TCP Hybla results will not refer to this option (although recommended), unless explicitly stated.

5. PERFORMANCE EVALUATION

5.1. *Performance in presence of congestion*

A wireless TCP connection with variable RTT and five wired TCP connections with fixed RTT (25 ms) are simultaneously active. To focus our attention on the presence of congestion, the wireless link is considered momentarily error free, so that all the losses are to be ascribed to the presence of the shared wired link, which acts as a network bottleneck. In such a situation, standard TCP suffers serious performance problems on the longer RTT connection, which 'starves', while the wired connections share most of the allowable bandwidth. To quantify this problem and to assess the effectiveness of the countermeasures introduced by the TCP Hybla protocol, in Figure 6 the goodput of the wireless connection has been reported as a function of its RTT, for several variants of TCP. Note that the performance of the wired links are deliberately not reported for intelligibility reasons. However, they are substantially independent

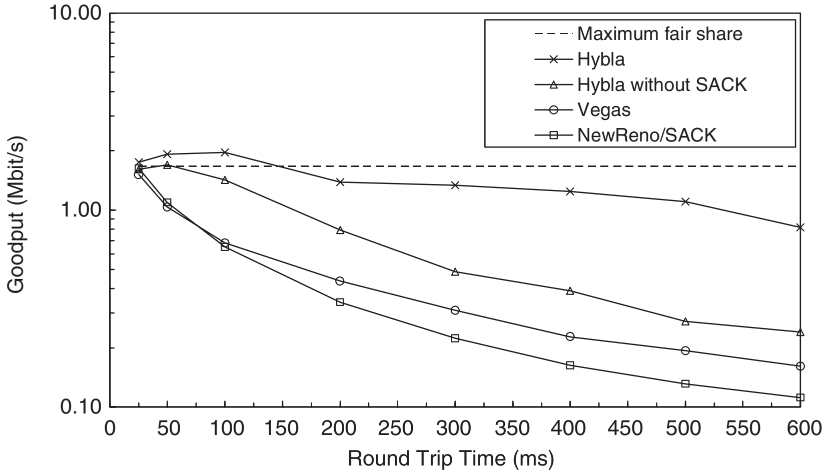

Figure 6. Performance of different TCP versions in the presence of congestion.

of the RTT and always very close to the reported 'maximum fair share', i.e. the capacity of the bottleneck link divided by the number of sharing connections. From the inspection of the figure the following observations can be made:

1. The longer the wireless connection RTT, the worse its performance in the TCP standard (NewReno), with a very fast goodput degradation rate. This problem has already been described in this paper and it is well known in the literature [28].
2. TCP Reno with SACK option presents the same performance of TCP NewReno, thus the SACK option alone is not effective in presence of congestion.
3. TCP Vegas performs only slightly better than TCP NewReno.
4. TCP Hybla without the SACK option shows an evident performance improvement; however, performance is still dependent on the RTT, although much less than before.
5. TCP Hybla (with the SACK option) proves to be very effective in counteracting the effects of long RTT and congestion losses; substantial performance independence of the RTT is achieved and the wireless connection is no longer penalized.

Some additional comments are required. As far as observation 2 is concerned, a thorough analysis of the simulation data allowed us to discover the reasons why the SACK option alone is ineffective in presence of severe congestion. This is basically due to the fact that the slow start and the congestion avoidance algorithms are left unaltered. SACK is faster in recovering multiple losses, but, when the steady state is reached, the average value of the wireless connection cwnd is usually very low even for long RTT connections, due to the inadequate TCP standard congestion control policies. As a result, multiple losses are relatively rare, and SACK benefits scarce.

The fact that TCP Vegas leaves unaltered the maximum cwnd growth rate of TCP standard justifies the limited improvement achieved by the wireless connection (observation 3).

Regarding observation 4, the decision to leave temporarily apart the SACK option is motivated by the desire of quantifying the performance improvement due to the modifications of the congestion control policy only. Note that, for the reasons reported in the

TCP Hybla section, these modifications would be enough to assure the desired performance independence of RTT in an ideal channel, i.e. in absence of losses of any nature. However, by leaving unaltered the loss recovery mechanism, every connection requires about one RTT to recover a packet loss, causing the aforementioned non compensated penalty for the longer RTT connections.

Finally, provided that the TCP protocol is able to assure a satisfactory transmission rate, multiple losses are more frequent on the wireless connection which requires a larger cwnd to assure a given transmission rate (see (3)). This is why TCP Hybla benefits highly by the inclusion of the SACK option (observation 5).

5.2. Performance in presence of link losses

For a full understanding of the influence that link errors have on TCP performance, it is convenient to leave any congestion problem temporarily apart. To this end, only the wireless connection is considered active and all the losses are a consequence of the non-null packet error rate. As TCP does not distinguish the origin of packet losses, link errors cause spurious interference on the congestion control mechanism, because a corrupted packet causes an unnecessary halving of the cwnd. The consequences of this erroneous interpretation are highlighted in Figure 7, where the performance of the wireless connection is reported versus its RTT, for the same TCP versions considered before, considering a 1% packet error rate. On examining the curves the following observations can be made:

1. Although the origin of the problem is completely different, the results are qualitatively the same as those reported in the congestion case just discussed. The superiority of TCP Hybla is apparent (performance gain higher than ten for RTT > 300 ms with respect to NewReno).
2. The performance improvement due to the inclusion of the SACK option in the TCP Hybla is less evident than in the congestion only case.
3. Although TCP Hybla provides a more than satisfactory goodput (3–2 Mbit/s), the theoretical capacity of the channel (10 Mbit/s) is still far from being reached.

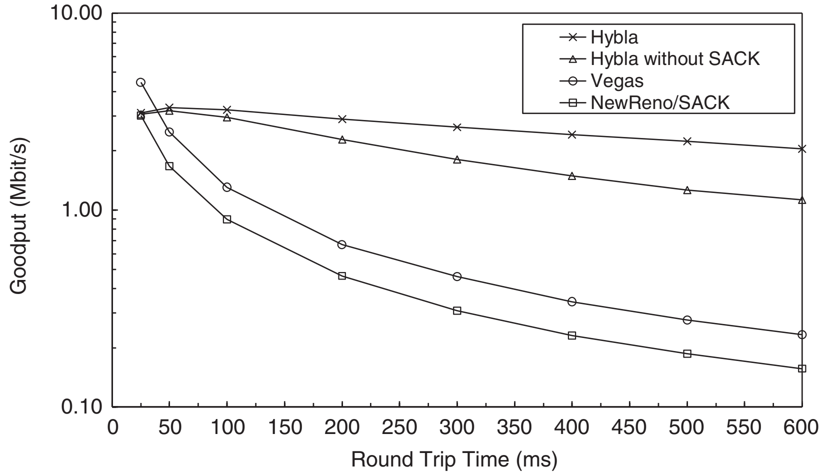

Figure 7. Performance of different TCP versions in the presence of residual link errors (PER = 1%).

As before, some comments are in order. Regarding the first point, it must be observed that the superiority of the Hybla version stems from its improved congestion control mechanism, which basically succeeds in neutralizing the effects of RTT. Concerning the second observation, the less significant advantages of the SACK option may be justified by the assumption of uniform distribution of packet errors, made in the simulation. It is likely that in a more realistic correlated fading environment the benefit provided by the SACK option could be greater. Finally, the failure of reaching the theoretical goodput limit for the wireless connection, apart from the small percentage of corrupted packets, is due to the aforementioned TCP inability to discriminate the origin of the losses. In this regard, it is worth noting that TCP Hybla aims to make the performance independent of the RTT and not (at least in the present version) to directly eliminate the consequences of link errors on wireless legs. To overcome the problem, it would be necessary to modify in depth the TCP algorithm by introducing the possibility of discriminating between congestion and error losses, for example by introducing NACK, as suggested in Reference [7]. However, effective improvements could also be obtained by introducing a different slow start threshold setting policy after a loss, as envisaged in the TCP Westwood [10]. It is worth noting that TCP Hybla is compatible with these enhancements.

5.3. Performance of both congestion and link losses

When both congestion and link errors are present, it is clear, on the basis of the previous considerations, that TCP standard versions are unable to provide a satisfactory performance, at least for long RTTs. For this reason, we prefer to focus on the more promising performance of the TCP Hybla version, considering the same topology adopted in the congestion only case, but introducing a non-null PER (namely, 1% and 5%). Results are reported in Figure 8, plotting the goodput of the wireless connection vs RTT. As a useful reference, the straight line of the 'maximum fair share', defined as in the congestion only case, it is also drawn. By inspection of the figure, the following observations can be made:

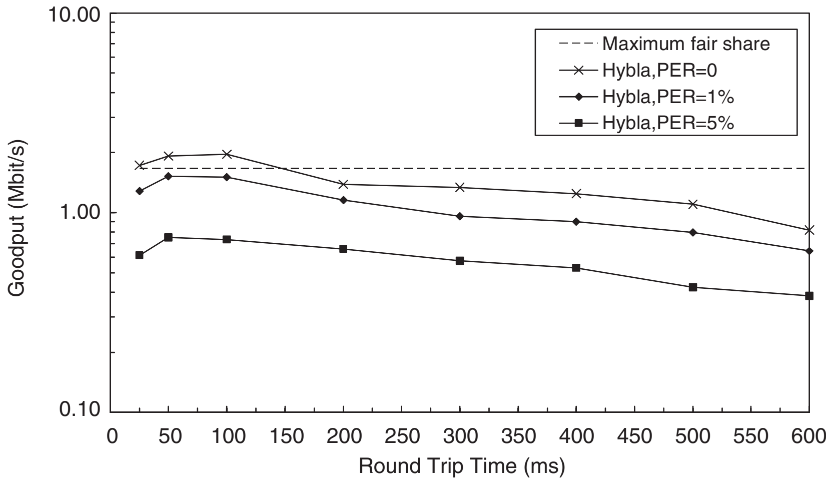

Figure 8. TCP Hybla performance in presence of both congestion and residual link errors, for several packet error rates.

1. The introduction of packet losses due to link errors causes a significant performance degradation, with respect to the congestion only case.
2. The worsening in performance due to the simultaneous presence of both congestion and link errors is lower than the sum of the two disjoint penalizations, likely because the spurious cwnd reductions caused by link errors, in presence of congestion, present the positive side effect of lowering the probability of packet overflow.

As a general comment, we can conclude by observing that TCP Hybla still offers a satisfactory performance for every RTT even in the case of a combined presence of congestion and a moderate percentage of link errors, while the standard TCP versions are unable to cope with these impairments, even if separately considered.

6. FAIRNESS AND FRIENDLINESS

Fairness and friendliness are two important features for any version of TCP protocol. Fairness refers to the capacity to assure a fair band subdivision among competing connections that use the same version of the protocol, while friendliness indicates the same ability with reference to different protocol variants. TCP standard versions are known to be fair as long as the competing connections have similar RTTs, which is usually not the case in heterogeneous networks, where wireless connections are highly penalized. This behavior is described in Figure 9, where the goodput of six TCP NewReno connections is reported in a pie chart, referring to the same simulation set-up considered previously for the congestion only case, except for the different

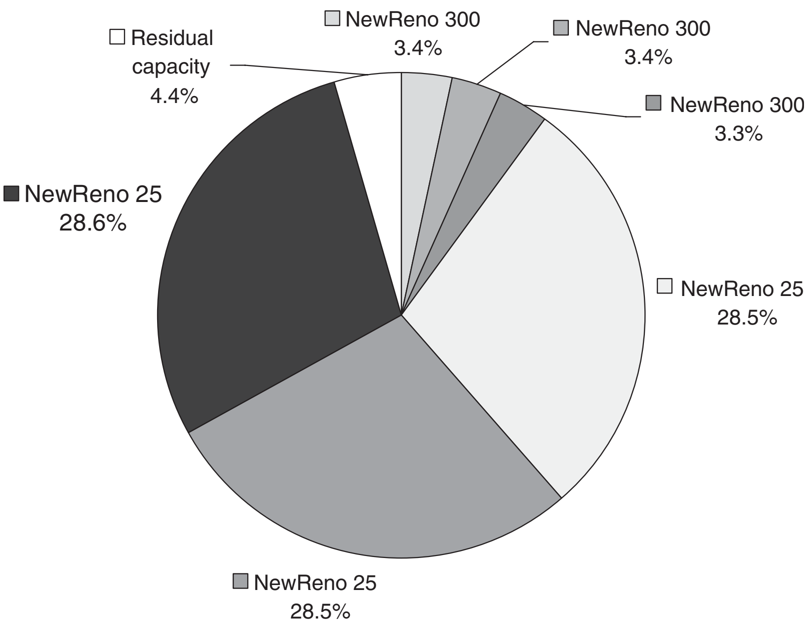

Figure 9. TCP Newreno unfairness in presence of links with different RTTs. Goodputs normalized to the bottleneck link nominal capacity.

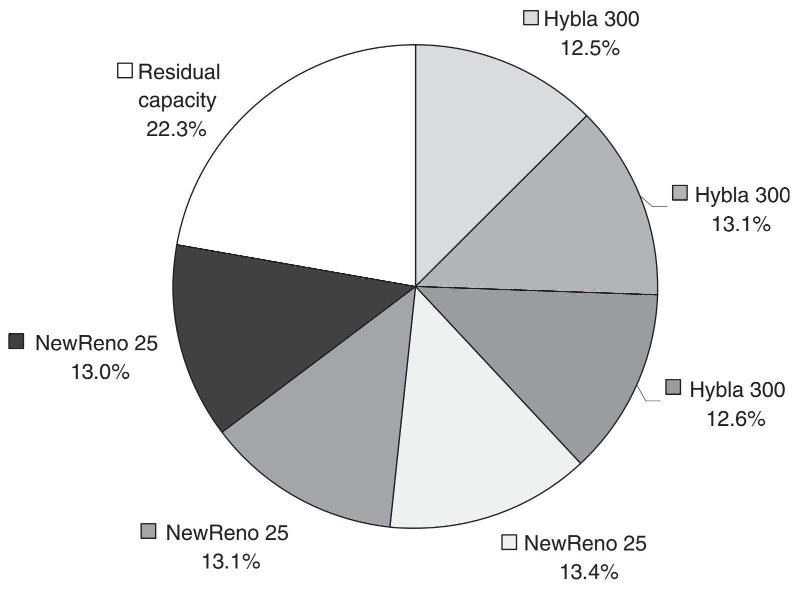

Figure 10. TCP Hybla fairness and friendliness in presence of links with different RTTs. Goodputs normalized to the bottleneck link nominal capacity.

number of wireless and wired connections (RTT $= 300$ ms and RTT $= 25$ ms, respectively). It is apparent that the wireless connections 'starve', while the short RTT connections share the majority of the bottleneck link capacity.

In contrast, the adoption of TCP Hybla for the three wireless connections leads to a totally different situation, reported in Figure 10. Not only do the TCP Hybla connections present the same goodput, proving the fairness of the Hybla proposal, but also provide the same performance of the TCP NewReno wired connections despite the different RTTs, thus giving a clear indication of the TCP Hybla friendliness.[‖]

In challenging situations, such as those considered in Figure 10 (congestion, half of the connections wireless with long RTT, i.e. large cwnds), burstiness may cause an efficiency loss, represented by the relatively high 'residual capacity', i.e. the difference between the nominal bottleneck link capacity and the sum of all the goodputs. This residual capacity, mainly used for packet retransmissions, gives an idea of bandwidth 'wasted' because of protocol inefficiencies, which are exacerbated by the bursty nature of the traffic. 'Packet spacing', by spreading the cwnd segments over the RTT, eliminates at the origin the transmission burstiness. By adopting this option on the Hybla connections considered in Figure 10, we have obtained the results presented in Figure 11, which show a great reduction of the residual capacity and a corresponding goodput improvement for all the competing connections.

Simulation results presented in Figures 10 and 11, provide evidence of the effectiveness of the TCP Hybla. The implementation of this proposal did not cause either instability or network

[‖]It might be objected that TCP Hybla would not be fair with possible competing wireless connections adopting TCP standard. However, it is apparent that would be unreasonable to pursue friendliness towards connections that *in any case* would suffer starvation from wired connections, as a consequence of the limits of the standard protocol.

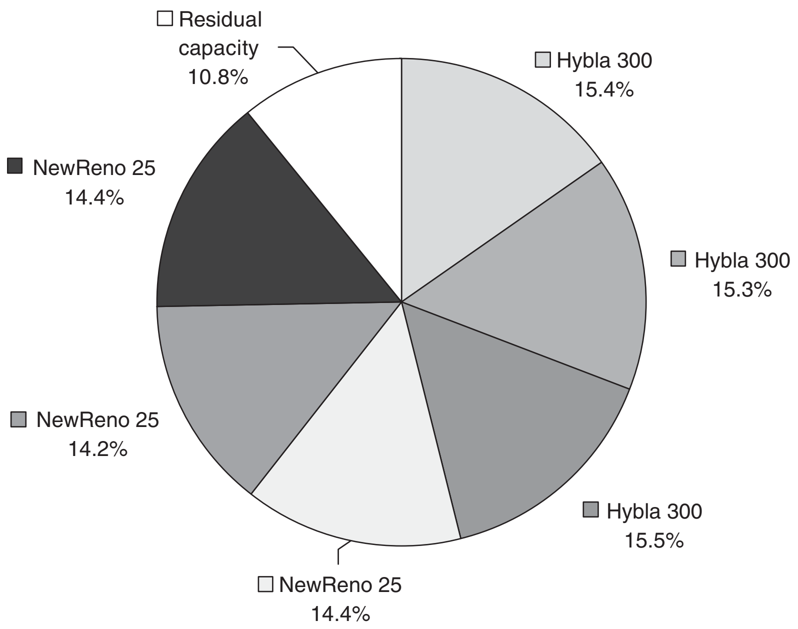

Figure 11. Packet spacing effectiveness in reducing the residual capacity. TCP Hybla fairness and friendliness in presence of links with different RTTs. Goodputs normalized to the bottleneck link nominal capacity. Packet spacing enabled.

overfeeding, as the adoption of a faster growth rate of the cwnd is always compensated (and motivated) by a longer RTT, while the increased burstiness for long RTTs, due to the better efficiency of the protocol, is effectively contrasted by the adoption of packet spacing.

7. ROUND TRIP TIME RELATED ISSUES

7.1. RTT_0 choice

The choice of the RTT_0 value is an important issue that deserves to be discussed in detail. Its implications are clear if we consider that a TCP Hybla connection aims at performing like a standard TCP connection with $RTT = RTT_0$. As a consequence, a TCP Hybla connection is friendly with other competing standard TCP connections, as long as the reference RTT_0 is comparable with their RTTs. However, if this was not the case, the entity of the unfriendliness would not be greater than the usual unfairness between TCP standard connections with different RTTs.

It is worth stating that TCP Hybla does not aim at a perfect equalization of all the TCP connections in an open network, a task that would be really problematic and would require a centralized control. Instead, it considers as a realistic goal the reduction of the performance penalization that long RTT connections, and especially satellite, are affected by. With this aim, the best RTT_0 choice for wireless and satellite connections would be the RTT of the connection itself, decreased by the additional delay due to the radio link. As this additional delay may be hard to estimate, a conservative but sensible choice for RTT_0 (for example, the average, or a given percentile value of the wired connections RTTs), could be enough to assure a significant boost of wireless links performance, without causing any real inconvenience to the wired links and to the network. Finally, note that, by contrast to [15], connections that are already 'fast' (i.e. with $\mathrm{RTT} \leqslant \mathrm{RTT}_0$) are never slowed down by TCP Hybla, which behaves for them as a standard TCP with some additional improvements (namely, SACK, timestamps and packet spacing).

7.2. *RTT estimate sensibility*

In order to determine the ρ value, TCP Hybla makes use of an estimate of the connection RTT that is already provided by the TCP standard algorithm (with sufficient granularity in its most recent implementations). The problem of a correct RTT estimate has already been addressed in literature [29] and it is out of the scope of the present paper. However, it is important to note that TCP Hybla is robust with respect to even large inaccuracies of this estimate. To realize that, it is sufficient to observe that reasonable errors on the RTT are equivalent, as far as the parameter ρ is concerned, to the choice of a slightly different RTT_0. Finally, let us observe that, in case of time variability of the RTT estimate, a conservative choice could be given by the selection of the minimum RTT value, in order to effectively compensate only the fixed time components of the RTT (propagation time, time possibly spent in interleaving and decoding, etc.), and not the variable components (buffer queues, etc.). This would prevent an overfeeding of the network in case of an increasing congestion, because additional delays due to buffer queues would result in a slow down of the transmission rate. Note however that for satellite connections the fixed components would be always dominant with respect to the time spent in buffer queues, and the practical differences minor.

8. CONCLUSIONS

TCP Hybla, the TCP enhancement proposed in this paper, represents a promising solution to the problem of performance disparity in heterogeneous networks due to different RTTs. Unlike many other proposals, this new algorithm is based on an analytical study of the congestion window evolution in time as a function of connections delays. After having modified the standard congestion control algorithm in accordance with the suggestions arising from the analysis, we also adopted the SACK option, timestamps and packet spacing to reduce the impact of multiple losses, inappropriate timeouts and burstiness, respectively. As a result of these and other minor modifications, TCP Hybla achieves its goal of substantial reduction in performance disparity against long RTT connections (e.g. satellite). In the paper, the clear advantage of TCP Hybla with respect of TCP standard versions, in the presence of congestion and link errors, is shown and discussed in details, as well as its good friendliness and fairness properties. Finally, it is worth noting that TCP Hybla does not infringe the end to end semantics of the TCP protocol and is compatible with other promising TCP enhancements.

REFERENCES

1. Hu Y, Li V. Satellite-based internet: a tutorial. *IEEE Communication Magazine* 2001; **39**(3):154–162.
2. Xylomenos G, Polyzos GC, Mahonen P, Saaranen M. TCP performance issues over wireless links. *IEEE Communication Magazine* 2001; **39**(4):52–58.
3. Barakat C, Altman E, Dabbous W. On TCP performance in a heterogeneous network: a survey. *IEEE Communication Magazine* 2000; **38**(1):40–46.
4. Allman M, Floyd S, Partridge C. Increasing TCPs initial window. *Request for Comment 2414*, September 1998, IETF.

5. Barakat C, Chaher N, Dabbous W, Altman E. Improving TCP/IP over geostationary satellite links. In *Proceedings of IEEE GLOBECOM '99*, 1999, 781–785.
6. Hoe JC. Improving the start-up behavior of a congestion control scheme for TCP. In *Proceedings of ACM SIGCOMM '96*, 1996; 270–280.
7. Mathis M, Mahdavi J. TCP selective acknowledgment options. *Request for Comment 2018*, October 1996, IETF.
8. Brakmo LS, Peterson LL. TCP Vegas: end to end congestion avoidance on a global internet. *IEEE Journal on Selected Areas in Communications* 1995; **13**(8):1465–1480.
9. Akyldiz IF, Morabito G, Palazzo S. TCP-Peach: a new congestion control scheme for satellite IP networks. *IEEE/ACM Transactions on Networking* 2001; **9**(3):307–321.
10. Casetti C, Gerla M, Mascolo S, Sansadidi MY, Ren Wang. TCP Westwood: end-to-end congestion control for wired/wireless networks. *Wireless Networks Journal* 2002; **8**:467–479.
11. Henderson TR, Katz RH. Transport protocols for internet-compatible satellite networks. *IEEE Journal on Selected Areas in Communications* 1999; **17**(2):326–334.
12. Ishac J, Allman M. On the performance of TCP spoofing in satellite networks. In *Proceedings of IEEE MILCOM'01*, 2001; 700–704.
13. Caini C, Firrincieli R, Raffaelli C. An RTT invariant congestion control scheme for heterogeneous IP networks. In *Proceedings of IST Summit on Mobile and Wireless Communications'03*, 2003; 802–806.
14. Floyd S. Connections with multiple congested gateways in packet-switched networks, part I: one-way traffic. *ACM Computer Communications Review* 1991; **21**(5):30–47.
15. Henderson TR, Sahouria E, McCanne S, Katz RH. On improving the fairness of TCP congestion avoidance. *IEEE GLOBECOM 98* 1998; **1**:539–544.
16. ns-2, network simulator (ver.2). LBL, URL: http://www-mash.cs.berkeley.edu/ns.
17. Allman M, Stevens W. TCP congestion control. *Request for Comment 2581*, April 1999, IETF.
18. Floyd S, Henderson T. The NewReno modification to TCP's fast recovery algorithm. *Request for Comment 2582*, April 1999, IETF.
19. Transmission Control Protocol, Darpa Internet Program, *Request for comment 793*, September 1981.
20. Chan ACF, Tsang DHK, Gupta S. TCP (transmission control protocol) over wireless links. In *Proceedings of IEEE Vehicular Technology Conference'97*, 1997; 1326–1330.
21. Lakshman TV, Madhow U. The performance of TCP/IP for networks with high bandwidth-delay products and random loss. *IEEE/ACM Transactions on Networking* 1997; **5**(3):336–350.
22. Stevens W. TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms. *Request for Comment 2001*, January 1997, IETF.
23. Fall K, Floyd S. Simulation-based comparison of Tahoe, Reno, and sack TCP. *Computer Communication Review* 1996; **26**(3):5–21.
24. Paxon V, Allman M. Computing TCP's retransmission timer. *Request for Comment 2988*, November 2000, IETF.
25. Aggarwal A, Savage S, Anderson T. Understanding the performance of TCP pacing. In *Proceedings of IEEE INFOCOM 2000*. 2000; 1157–1165.
26. Caini C, Firrincieli R. Packet spreading techniques to avoid bursty traffic in satellite TCP connections. In *Proceedings of IEEE VTC Spring '04*, 2004, accepted for publications.
27. Hasegawa G, Murata M. Analysis of dynamic behaviors of many TCP connections sharing tail-drop/RED routers. In *Proceedings of IEEE GLOBECOM '01*, 2001; 1811–1815.
28. Padhye J, Firoiu V, Towsley DF, Kurose JF. Modelling TCP Reno performance: a simple model and its empirical validation. *IEEE/ACM Transactions on Networking* 2000; **8**(2):133–145.
29. Karn P, Partridge C. Improving round-trip time estimates in reliable transport protocols. *ACM Transactions on Computer Systems* 1991; **9**(4):364–373.

AUTHORS' BIOGRAPHIES

Carlo Caini was born in Bologna, Italy, in 1960. He received the Dr Ing Degree (cum laude) in Electronic Engineering from the University of Bologna, Bologna, Italy, in 1986. Since 1990, he has been with the Department of Electronics Computer Science and Systems of the same University, as a Research Associate. His main scientific interests are in the field of terrestrial and satellite cellular mobile radio systems, with a special emphasis on spectrum efficiency, multiple access techniques and spread spectrum systems. The recent integration of Internet and wireless communications has led him to devote his research activity also to the development of network protocols for satellite and wireless applications. He has participated to several international research projects and he is author of many international publications on these topics. He is member of IEEE Communications Society.

Rosario Firrincieli graduated in 2001 in Telecommunications Engineering at the University of Bologna (Italy). Since 2002 he is with the Advanced Research Center on Electronics Systems for Information and Communication Technologies (ARCES) at the University of Bologna, where he is currently a PhD candidate.

His present interests involve the study and the evaluation of transport protocol performance over wireless network, congestion control algorithms for unicast and multicast protocols, traffic shaping, retransmission techniques (ARQ, Data Carousel), and packet coding at the transport layer. Further areas of interest are mobile IP and security on IP networks.