Computer Networks 54 (2010) 2327–2344

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

TCP *Libra*: Derivation, analysis, and comparison with other RTT-fair TCPs

Gustavo Marfia [c,*], Claudio E. Palazzi [b], Giovanni Pau [a], Mario Gerla [a], Marco Roccetti [c]

[a] Computer Science Department, University of California, Los Angeles, CA 90095, United States
[b] Dipartimento di Matematica Pura e Applicata, Università degli Studi di Padova, 35121 Padova, Italy
[c] Dipartimento di Scienze dell'Informazione, Università di Bologna, 40126 Bologna, Italy

ARTICLE INFO

Article history:
Received 10 March 2009
Received in revised form 9 February 2010
Accepted 24 February 2010
Available online 25 March 2010
Responsible editor: S. Mascolo

Keywords:
Fairness
RTT
TCP
Transport protocol

ABSTRACT

The Transmission Control Protocol (TCP), the most widely used transport protocol over the Internet, has been advertised to implement fairness between flows competing for the same narrow link. However, when session round-trip-times (RTTs) radically differ, the share may be anything but fair. This RTT-unfairness represents a problem that severely affects the performance of long-RTT flows and whose solution requires a revision of TCP's congestion control scheme. To this aim, we discuss TCP Libra, a new transport protocol able to ensure fairness and scalability regardless of the RTT, while remaining friendly towards legacy TCP. As main contributions of this paper: (i) we focus on the model derivation and show how it leads to the design of TCP Libra; (ii) we analyze the role of its parameters and suggest how they may be adjusted to lead to asymptotic stability and fast convergence; (iii) we perform model-based, simulative, and real testbed comparisons with other TCP versions that have been reported as RTT-fair in the literature. Results demonstrate the ability of TCP Libra in ensuring RTT-fairness while remaining throughput efficient and friendly towards legacy TCP.

1. Introduction

Traffic control functionalities in the Internet are provided by the Transmission Control Protocol (TCP) in an end-to-end fashion. TCP addresses three major issues: reliability, flow control and congestion control [1]. To achieve the third goal, TCP adapts the sending rate to avoid network overflow. The most popular versions, TCP New Reno and TCP SACK [2], implement a congestion control algorithm which falls into the AIMD (Additive Increase and Multiplicative Decrease) family of algorithms and whose very basic concepts can be summarized as follows:

- when a packet loss is detected, the TCP sender decreases its sending window by half;
- when a packet is successfully delivered, the TCP sender increases its sending window by one over the sending window.

TCP's feedback for the successful delivery of a packet is embodied by a returning acknowledgement (ACK). As a result, competing TCP senders with different end-to-end propagation delays will typically receive feedbacks at different rates and adapt their sending rate at a different pace. This phenomenon determines the RTT-bias, or *RTT-unfairness*, of TCP New Reno and TCP SACK. A number of TCP variants have been designed to limit the effects of this problem and to improve scalability over gigabit links. Most of them adopt a proactive approach based on monitoring packets' RTT and reacting to its increase in an attempt to avoid network congestion [3]. This behavior is justified by the assumption of a strong correlation between packet loss and RTT increase prior to the loss event. Yet, this dependency has been proven to be weak in [4], RTT probes may still be too coarse to correctly foresee congestion [5].

* Corresponding author.
E-mail addresses: marfia@cs.unibo.it (G. Marfia), cpalazzi@math.unipd.it (C.E. Palazzi), gpau@cs.ucla.edu (G. Pau), gerla@cs.ucla.edu (M. Gerla), roccetti@cs.unibo.it (M. Roccetti).

1389-1286/$ - see front matter © 2010 Published by Elsevier B.V.
doi:10.1016/j.comnet.2010.02.014

Examples of algorithms that fit into this category are TCP Vegas [6], TCP DUAL [7] and FAST TCP [8,9].

Instead, we have chosen a different approach, named *TCP Libra*,[1] by which, even if the sending window is controlled based on RTT measurements, the main trigger for window changes remains the packet loss [10]. Even if our scheme takes into account the delay information, it does not use it to lower the sending rate; rather, the RTT information is used to delay just the speed increase of the sending rate. In essence, TCP Libra delays the moment at which congestion will occur, instead of just preventing congestion in the network.

The contributions of this work include the complete derivation of the TCP Libra algorithm, an analysis of its stability bounds, the validation of the algorithm implemented both in Matlab, in NS2, and in the Linux stack, even through the comparison with other RTT-fair schemes.

The rest of the paper is organized as follows. In Section 2 and Section 3 we present the state of the art in RTT-fairness and TCP modeling, respectively. The congestion control algorithm of TCP Libra is introduced in Section 4, along with a subsection dedicated to a stability analysis of Libra. A model-based comparison with other RTT-fair schemes is performed in Section 5. Experimental assessment and results are reported in Section 6 and in Section 7, respectively. Finally, conclusions and future work are presented in Section 8.

2. Addressing RTT-unfairness: related work

A detailed mathematical model for the TCP throughput at steady state, including the Fast Retransmit–Fast Recovery phases and TCP's timeout impact, was first introduced by Padhye et al. in [11].

In [12–15] the congestion control problem is expressed as a utility maximization problem, where the network utility function is represented by the sum of utilities of each single source, and the constraints are given by links' interconnections and capacities. This flow of work shows that TCP stability can be achieved in the aforementioned network model if the TCP utility function is concave.

A further substantial advancement in developing the theory for network congestion control is exploited in the primal/dual modeling approach [16]. The theoretical results have been used to drive the design of an enhanced AQM technique, namely Random Early Mark (REM), and of a new transport protocol, namely FAST TCP. More in detail, the latter implements a congestion control mechanism, based on queuing time, which achieves network stability and high utilization in multi-gigabit networks [8,9,17,18].

The RTT-bias was first experimentally observed in [19]. The authors propose a solution for this problem based on a constant window increase algorithm. Henderson [20] and Henderson et al. [21] show that such solution leads to instability and thus RTT-unfairness. This is especially true for links with long propagation delays and small buffers such as satellite links.

To this aim, a few works have recently proposed new RTT-fair TCPs. Among the most relevant ones, TCP Hybla [22] implements a constant increase algorithm and provides RTT-fairness under a certain stability bound. TCP Vegas [6] provides good RTT-fairness but disregards friendliness. FAST TCP [8,9] increases TCP Vegas' stability bounds, but with a behavior that results either too timid or too aggressive when coexisting with legacy TCP protocols (e.g., TCP New Reno and TCP SACK).

Finally, CUBIC [23] features a linear RTT-fairness that claims to improve BIC [24]. In particular, CUBIC tries to decouple the window growth from the returning ACK process (a similar approach is proposed also by H-TCP [25]). With CUBIC, the window size is a function of the time elapsed since the last packet loss, thus allowing higher efficiency (in terms of total bandwidth utilization) in case of long fat RTTs and reducing, even if not completely eliminating, the throughput dependency from the RTT. Indeed, the throughput still corresponds to the ratio between the window size and the RTT, where two flows with similar packet loss trend may have the same window but different RTTs.

Instead, as we discuss in Section 4, our approach takes into account the RTT information to dynamically adapt the speed increase of the sending rate. This allows to further improve the RTT-fairness while preserving efficiency.

3. TCP model background

In this section we review the background necessary to interpret the end-to-end congestion control problem as a network utility maximization problem [26]. We show how TCP New Reno fits in this model and why it prevents fair RTT behavior. Needless to say, the following discussion about TCP New Reno model holds also for other similar protocols such as TCP SACK.

3.1. Network model and optimization problem

The network is modeled as a finite set of nodes N and links L of finite capacity, which connect the nodes in N. We define $\underline{c}$ as the vector of link capacities where each row $(c_l,\ l \in L)$ represents the capacity of link $l \in L$. S is the set of sources that accesses network resources, typically a subset of N and L. Routing matrix $\underline{\underline{R}}$ has entry one in position (i,j) if link i is utilized by source j, zero otherwise. Each source $r \in S$ is characterized by its transmission rate, $x_r(t)$. The *aggregate flow* at link l is defined as the sum of the contributions from all sources that use that link:

$$y_l(t) = \sum_r R_{lr} x_r(t - \tau^f_{lr}), \tag{1}$$

where τ^f_{lr} is the forward delay from source r to link l. We define *price* to be the marginal cost (or penalty) per unit flow that a source incurs in sending that flow increment. Intuitively, a link sends an increased price, as a feedback signal, when congestion is detected.

The *aggregate price* seen by source r is:

$$\lambda_r(t) = \sum_l R_{lr} p_l(t - \tau^b_{lr}), \tag{2}$$

[1] *Libra* in Latin means *scale*, thus indicating a balance between the sessions.modelling

where τ^b_{lr} is the backward delay in the feedback path from link l to source r, $p_l(t)$ is the price signal sent by link l at time t. We also define the *marginal link price* $f_l(y)$ as the marginal cost for sending traffic at rate $y_l = \sum_{r:l\in r} x_r$ on link l.

Let us now suppose we are able to define a function that describes precisely the return that each source r experiences when sending data at rate x_r. In fact, it is very difficult to understand which is the real advantage for a user when sending at a certain rate. The function that describes this advantage is defined in economics as a *utility function*. The utility function of a congestion control scheme shapes its equilibrium properties, such as the equilibrium sending rate and its fairness properties.

We now have all the ingredients to state the optimization problem we want to address:

$$max_{\underline{x}} V(\underline{x}), \tag{3}$$

subject to:

$$\begin{cases} \underline{\underline{R}}\,\underline{x} \leqslant \underline{c}, \\ x_r \geqslant 0, \quad \forall r \in S, \end{cases} \tag{4}$$

where $V(\underline{x}) = \sum_r U_r(x_r) - \sum_l \int_0^{\sum_{s:l\in s} x_s} f_l(y)dy$. By definition $\int_0^{\sum_{s:l\in s} x_s} f_l(y)dy$ is the total cost incurred at resource l for pushing the contributions from all sources that utilize l (i.e., $\sum_{s:l\in s} x_s$ represents the aggregate flow pushed through l). Thus, $V(\underline{x})$ is the net gain, i.e., the net utility of sources S, which must be maximized.

Theorem 3.1 ([26,27]). *Under the assumptions:*

(1) *$U_r(x_r)$ is a continuously differentiable, non-decreasing, strictly concave function;*
(2) *$f_l(y)$ is a non-decreasing, continuous function;*

starting from any initial condition $\{x_r(0) \geqslant 0\}$, the distributed congestion control algorithm,

$$\dot{x}_r = k_r(x_r)\big(U_r'(x_r) - \lambda_r(t)\big), \tag{5}$$

(where $k_r(x)$ is any non-decreasing, continuous function such that $k_r(x) > 0,\ \forall x_r > 0$) will converge to the unique solution of (3) (4). In other words, $\underline{x}(t) \to \hat{\underline{x}}$ as $t \to \infty$, where $\hat{\underline{x}}$ is the unique solution to (3) and (4).

Intuitively, we can identify packet loss or end-to-end delay as $\lambda_r(t)$ and the algorithm's behavior as $U_r'(x_r)$ in (5). A high packet loss or end-to-end delay, according to (5), provokes a lower sending rate and vice versa.

The right-hand side of (5) represents the rth component of $\nabla V(\underline{x})$, to which the multiplicative term $k_r(x_r)$ was added. Normally, in the conventional gradient method, $k_r(x_r) = 1$. There is no harm, however, in introducing a non-decreasing function that acts as a *gradient amplifier*. More intuitively, the quantities that appear in the expression are:

(1) $k_r(x_r)$, the *stepsize* of the algorithm. As mentioned earlier, this term is an amplification factor that determines the amount by which the algorithm moves towards the solution at each step. This term determines the speed of convergence and the stability of the algorithm.
(2) $(U_r'(x_r) - \lambda_r(t))$, the *direction* in which the algorithm is proceeding, searching for a solution (stable point).

A detailed proof of convergence can be found in [26]. However, we notice that the function with the derivative shown on the right-hand side of (5) is concave by construction. The multiplicative term does not change the value x_r that nullifies the gradient. Thus, the gradient method leads to the unique optimum of function $V(\underline{x})$.

In brief, Theorem 3.1 states that in the absence of feedback delay, any congestion control algorithm that can be mapped into a concave utility function attains global asymptotic stability.

3.2. TCP New Reno

Let us now consider the fluid model for congestion control of TCP New Reno. From here on we will follow the notation:

- The r subscript means we are considering the rth source.
- $x_r(t)$ is the rate of the connection at time t.
- $w_r(t)$ is the window size of the connection at time t.
- $\widetilde{RTT}_r$ is the average RTT.
- $\lambda_r(t)$ is the probability of loss at time t.
- a_r, the increase factor, is a constant that in TCP New Reno is set to 1.
- b_r, the decrease factor, is a constant that in TCP New Reno is set to 1/2.

TCP New Reno increments the window by $1/w_r(t)$ per each received ACK, hence the window increases as $\frac{x_r(t)}{w_r(t)}(1-\lambda_r(t))$. Similarly, every three consecutive duplicate ACKs (i.e., a packet loss indication), the window is cut by half. The rate of this event is $x_r(t)\lambda_r(t)$. The window then decreases at a rate of $x_r(t)\lambda_r(t)w_r(t)/2$. We now may write the fluid model for congestion control for an AIMD-like congestion control scheme (e.g., TCP New Reno) under the assumption that $RTT_r(t) = \widetilde{RTT}_r$, $w_r(t) = x_r(t)\widetilde{RTT}_r$, and taking in account feedback delays:

$$\dot{x}_r(t) = \frac{x_r(t-\widetilde{RTT}_r)}{\widetilde{RTT}_r}\left(a_r\frac{1-\lambda_r(t)}{x_r(t)\widetilde{RTT}_r} - b_r\lambda_r(t)x_r(t)\widetilde{RTT}_r\right). \tag{6}$$

In (6) we consider that a window update at time t is determined by the window state at time $t-\widetilde{RTT}_r$, because of feedback delay. This non-linear differential equation models the throughput of the rth flow.

TCP New Reno implements an approximate gradient algorithm for the resolution of the congestion control problem. In terms of (5), and considering the feedback delay as negligible, we can write [26]:

$$\dot{x}_r = a_r\left(\left(\frac{b_r}{a_r}\right)x_r^2(t) + \frac{1}{RTT_r^2}\right)\left(\frac{1}{\frac{b_r}{a_r}RTT_r^2x_r^2(t)+1} - \lambda_r(t)\right). \tag{7}$$

The above expression fits into the mathematical framework introduced in Section 3.1. Observing the structure of (7) and comparing it with (5) we have:

$$U_r'(x_r) - \lambda_t(t) = \left(\frac{1}{\frac{b_r}{a_r} RTT_r^2 x_r^2(t) + 1} - \lambda_r(t) \right). \quad (8)$$

Integrating in x_r the term $U_r'(x_r)$, the utility function of TCP New Reno follows:

$$U(x_r) = \frac{1}{\widetilde{RTT}_r} \sqrt{\frac{a_r}{b_r}} \tan^{-1} \left(\sqrt{\frac{b_r}{a_r}} \widetilde{RTT}_r x_r \right). \quad (9)$$

By observing (9) we note that setting $a_r = c\widetilde{RTT}_r^2$ (with c some constant value) would produce an RTT-independent utility function. This solution is discussed in [19–21]. The main drawbacks may be summarized in a slow convergence speed to the fair share and a decreased stability of the protocol. In the following we introduce TCP Libra's design, which leads to fast convergence and stable behavior; we intuitively justify the former and analytically demonstrate the latter.

4. The TCP Libra algorithm

In the previous literature, Floyd et al. in [19] and Henderson in [20] prove that the simple constant increase approach proposed earlier for RTT-fairness fails. This simple approach consists in multiplying the congestion window by the square of the RTT during the additive increase portion of the TCP algorithm. Even though this approach claims to make TCP's utility function RTT-independent, it fails for stability reasons, as Kelly proved in [12].

In the rest of this section we show how TCP Libra's utility function is not free from RTT components, as the reader might at this point expect; yet, we prove that TCP Libra, with a correct setting of its parameters, can achieve a good stability region. Intuitively, what TCP Libra does is to take into account how close the flow is to overflowing the network with packets. This is measured by observing how close the current RTT is to the maximum experienced RTT: the closer these two values are, the slower the congestion window will grow and congest the network.

4.1. Enhancing TCP New Reno

The feedback control system for regular TCP New Reno is described in (6), where x_r is the state variable, λ_r is the input to the system, a_r and b_r are control parameters that can be tuned. The new terms $\hat{a}_r$ and $\hat{b}_r$ that we propose in TCP Libra and that substitute a_r and b_r are:

$$\hat{a}_r = \frac{\alpha_r \widetilde{RTT}_r^2}{T_0 + \widetilde{RTT}_r} a_r, \quad (10)$$

$$\hat{b}_r = \frac{T_1}{T_0 + \widetilde{RTT}_r} b_r. \quad (11)$$

Please note that compared to the solution discussed in [19–21] we here have two new entries, the coefficients α_r and $T_1/(T_0 + \widetilde{RTT}_r)$, where T_0 and T_1 are constants.

In brief:

$$\alpha_r = k_1 C_r e^{-k_2 \frac{RTT_r(t) - RTT_r^{min}}{RTT_r^{max} - RTT_r^{min}}}, \quad (12)$$

where k_1 and k_2 are constants, C_r is the capacity of the narrow link seen by the rth source, RTT_r^{min} and RTT_r^{max} are the minimum and maximum RTT seen by source r. The component $k_1 C_r$, introduced in [10] as the *scalability factor*, adapts the convergence speed of the protocol as the narrow link capacity increases. Instead, $e^{-k_2 \frac{RTT_r(t) - RTT_r^{min}}{RTT_r^{max} - RTT_r^{min}}}$ is the *penalty factor* discussed in [10]. The careful reader may object that we are here re-introducing the dependency from the RTT. Instead, we shall see from our results that this term keeps the protocol asymptotically stable and preserves RTT-fairness.

The fluid model for TCP Libra can be derived by substituting $\hat{a}_r$ and $\hat{b}_r$ into (7) with (10) and (11), respectively, and by setting $a_r = 1,\ b_r = 1/2$. The resulting equation is:

$$\dot{x} = \left(\frac{T_1/2}{\widetilde{RTT}_r + T_0} x^2(t) + \frac{\tilde{\alpha}_r}{\widetilde{RTT}_r + T_0} \right) \left(\frac{1}{\frac{T_1}{2\tilde{\alpha}_r} x^2(t) + 1} - \lambda_r(t) \right). \quad (13)$$

The marginal utility function $U'(x)$ loosely depends on $\widetilde{RTT}_r$ (the average RTT) through $\tilde{\alpha}_r$, but we shall see that this dependence is very low. We can therefore state that TCP Libra minimizes the sum of the transfer delays in the network, yet independently from the RTT experienced by the source.

These values lead to the following stable point for the rth source:

$$\tilde{x}_r = \sqrt{2 \frac{\tilde{\alpha}_r}{T_1} \frac{1 - \tilde{\lambda}_r}{\tilde{\lambda}_r}}. \quad (14)$$

Summarizing now the design choices in (10) and (11) for $\hat{a}_r$ and $\hat{b}_r$, respectively, we can state that:

(1) we achieve a stabilized rate, almost independent from RTT, as we shall see in the next sections;
(2) convergence speed to the stable point is preserved as link speed increases through the introduction of a *scalability factor*;
(3) stability and RTT-fairness are enforced through the *penalty factor*.

Algorithm 1. Congestion Window Update in TCP Libra

$window_n \Leftarrow$ congestion window at step n
$threshold_n \Leftarrow$ slow start threshold at step n
if a packet is successfully delivered **then**
  $window_{n+1} \Leftarrow window_n + \frac{1}{window_n} \frac{\alpha_n RTT_n^2}{RTT_n + T_0}$
else
  if three duplicate acknowledgements **then**
    $window_{n+1} \Leftarrow window_n - \frac{T_1 window_n}{2(RTT_n + T_0)}$
    $threshold_{n+1} \Leftarrow window_n - \frac{T_1 window_n}{2(RTT_n + T_0)}$
  end if
end if

4.2. Algorithm

The resulting algorithm is represented in Algorithm 1. Simulations in Section 6 are obtained setting $T_0 = 1$, $T_1 = 1$, $k_1 = 2$, $k_2 = 2$. A higher value of k_2 would have improved the link utilization; this can be explained by observing that a higher k_2 would strengthen the dependence of the window increase algorithm on the RTT and thus delay packet loss events. In brief, a higher k_2 enforces the sending window to be at its maximum for a longer time, but we have noticed that a higher k_2 generates an excessively timid behavior of TCP Libra toward TCP New Reno. Parameters k_2 and k_1 are strictly related and are adjusted as a tradeoff between utilization, fairness, and friendliness. T_1 is the parameter that sets the multiplicative decrease term, whereas T_0 is the parameter that sets the sensitivity of the protocol to RTT. The window increase is driven, for $RTT_n \ll T_0$ (the typical case if $T_0 = 1$ [28]), by the α factor and by the square of the RTT. In this case, RTT-fairness is enforced and the algorithm helps large bandwidth-delay-product flows, by letting their windows grow much faster than in TCP New Reno. If instead $RTT_n \gg T_0$ (a rather rare event where pathological congestion or routing problems are affecting the connection), the window increase is driven by the α factor and the RTT; in this case, RTT-fairness is not preserved, yet, it is weighted only as the inverse square root of the RTT.

4.3. Stability analysis

As we have seen in the previous section, parameters k_1, k_2, T_0, T_1 play an important role in TCP Libra. We here show, extending the stability analysis of TCP New Reno to TCP Libra, how they should be set. We will also understand the importance of the *penalty factor* in keeping the protocol within the asymptotic stability bounds.

Let us consider a single TCP source on a single link, where the probability of loss is modeled as the probability of having a queue of length $\geqslant \beta$ on a M/M/1 queuing system. From [26] we have that sufficient condition to achieve asymptotical local stability for an AIMD protocol is:

$$\kappa \widetilde{RTT} \frac{\tilde{\lambda}'}{\tilde{\lambda}} < \frac{\pi}{2}, \tag{15}$$

where κ depends from the particular TCP scheme in the AIMD family. In the case of a TCP New Reno flow, $\kappa = \kappa_{NewReno} = a_r / \widetilde{RTT}^2$ (we substitute $a_r = 1$ from now on):

$$\frac{\tilde{\lambda}'}{\tilde{\lambda}} < \frac{\pi \widetilde{RTT}}{2}. \tag{16}$$

Let us now consider a TCP Libra flow. For a TCP Libra flow we have that $\kappa_{Libra} = \kappa_{NewReno} * \hat{a}_r = \frac{\alpha}{\widetilde{T_0 + RTT}}$.[2] We substitute this value in (15) and obtain:

$$\frac{\tilde{\lambda}'}{\tilde{\lambda}} < \frac{\pi(\widetilde{RTT} + T_0)}{2\alpha \widetilde{RTT}}. \tag{17}$$

By satisfying the condition that holds for TCP New Reno in (16), we find an upper bound for α. In particular, $\alpha < (\widetilde{RTT} + T_0)/\widetilde{RTT}^2$ gives us the values of α for which the stability region of Libra is greater or equal than New Reno's. Recalling from Section 4.1 that $\alpha = Scalability\ Factor * Penalty\ Factor$, we see that the above bound gives a condition on k_2 once k_1 and T_0 are fixed, and vice versa. We derive the following from (12) and (17):

$$k_2 > -\frac{RTT_{max} - RTT_{min}}{\widetilde{RTT} - RTT_{min}} log\left(\frac{\widetilde{RTT} + T_0}{k_1 C \widetilde{RTT}^2}\right). \tag{18}$$

The above is clearly a qualitative analysis, since it relates to the simple case of a single link and a single flow, but it gives us the feeling of how a choice of k_1, k_2 and T_0 should be made. Specifically, two important considerations emerge from (18):

- an increase of k_1 should correspond to an increase of k_2 or T_0 in order to keep the same stability bound as New Reno;
- as expected, higher values of C and $\widetilde{RTT}$ stress the protocol and require a higher value of k_2 to prevent potential aggressiveness of the transport protocol.

The role of k_2 can be appreciated in Figs. 1 and 2, where we see the results of a simulation performed in NS2. Two flows, with round-trip propagation delay of 400 ms and 10 ms, respectively, in a dumbbell topology, compete on a 100 Mbps narrow link. In the simulation related to Fig. 1, we set $k_1 = 2$, $k_2 = 2$, $T_0 = 1$, and $T_1 = 1$ the two flows share the link fairly. Instead, in the simulation related to Fig. 2 we set $k_2 = 0$ (i.e., we omit the penalty function) and leave other parameters unchanged. The second experiment shows that without a penalty function (thus, similar to [19–21]) RTT-fairness is not preserved.

Outcomes from other simulative configurations, implemented both in Matlab and NS2, allow us to observe the behavior of two Libra flows as T_0 and k_1 increase (other parameters are set to the default values, $T_1 = 1$ and $k_2 = 2$). Specifically, we simulate two flows sharing a single 100 Mbps narrow link: flow 1 and flow 2, with round-trip propagation delays of 10 ms and 83 ms, respectively. In our NS2 simulations, we have set the slow start threshold to 1 and the packet size to 1500 Bytes. We have also considered RED implemented on the bottleneck router with the following parameters: $\mu_{max} = 0.1$, $b = 150$ pkts, $B = 600$ pkts, and *queue averaging weight* = 0.002.

Results achieved by flow 1 in three configurations employing different T_0 and k_1 values are reported in Figs. 3–5; whereas concurrent results of flow 2 are shown in Figs. 6–8. In the three simulative configurations the average throughput achieved by each flow is about the same, regardless of the flow's round-trip propagation delay. Indeed, stability is preserved by having utilized a constant T_0/k_1 value. Moreover, a reduction of the instantaneous throughput variance can be observed when increasing T_0 and k_1. This is coherent with what already observed in [14]: the throughput variance depends on the AIMD decrease rule thus having higher T_0 values producing the shown effect.

We can also approach the problem from the opposite direction. We first set $T_0 = T_1 = 1$, $k_1 = 2$; then, we esti-

[2] Here we substitute $a_r = 1$.

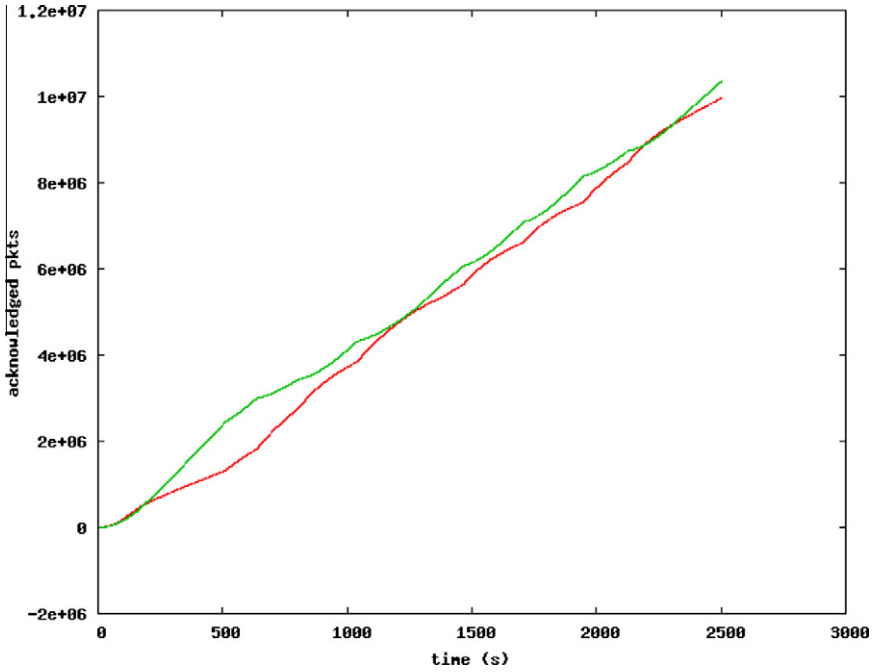


Fig. 1. Throughput of two flows, using a penalty function.

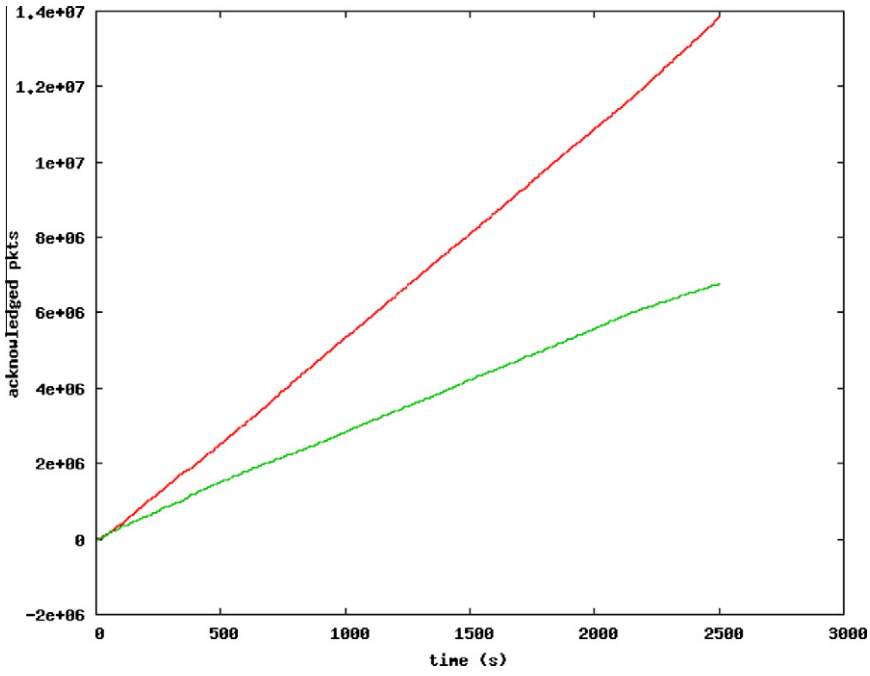


Fig. 2. Throughput of two flows, omitting the penalty function.

mate the variables in a typical connection and use (18). We here assume a 100 Mbps link speed with 100 ms one-way delay. The buffer is set to the pipe size, 833 packets for a packet length set to 1500 Bytes. We also consider the case in which the link is congested, the average RTT is equal to the maximum RTT, $\widetilde{RTT} = RTT_{max}$. From (18), we derive:

$$k_2 > -log\left(\frac{0.2+1}{2*100*(0.2)^2}\right) = -log(0.15) = 1.89. \qquad (19)$$

Coherently, we have set $k_2 = 2$ in our simulations. We will see in Section 7 that the chosen set of values for the TCP

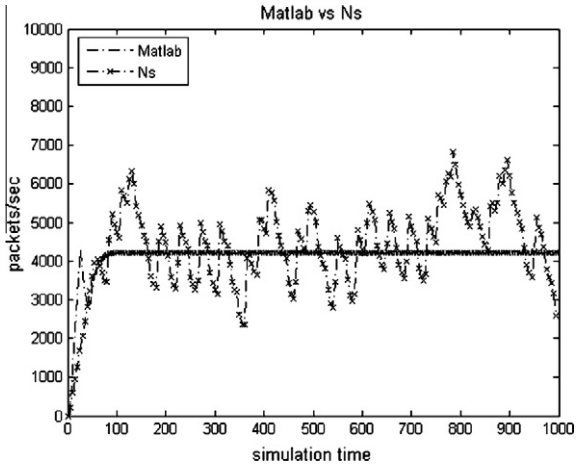


Fig. 3. Flow 1, with round-trip propagation delay = 10 ms, competing with flow 2 on a 100 Mbps bottleneck. $T_0 = 1$ and $k_1 = 2$.

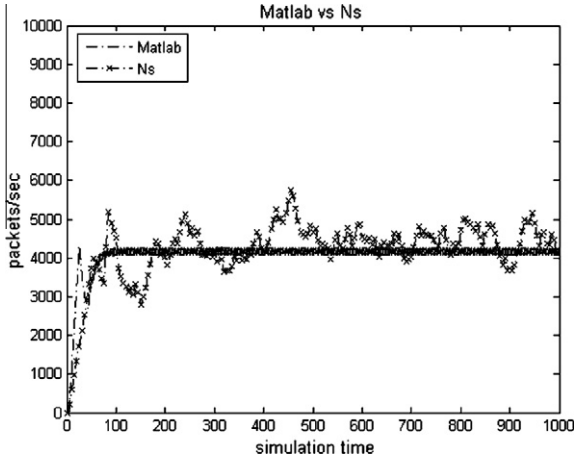


Fig. 4. Flow 1, with round-trip propagation delay = 10 ms, competing with flow 2 on a 100 Mbps bottleneck. $T_0 = 4$ and $k_1 = 8$.

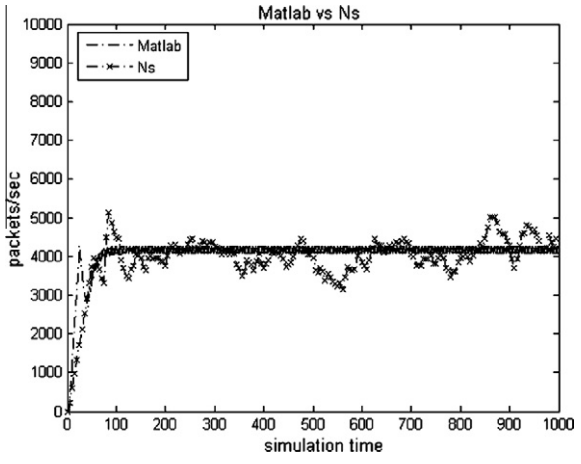


Fig. 5. Flow 1, with round-trip propagation delay = 10 ms, competing with flow 2 on a 100 Mbps bottleneck. $T_0 = 8$ and $k_1 = 16$.

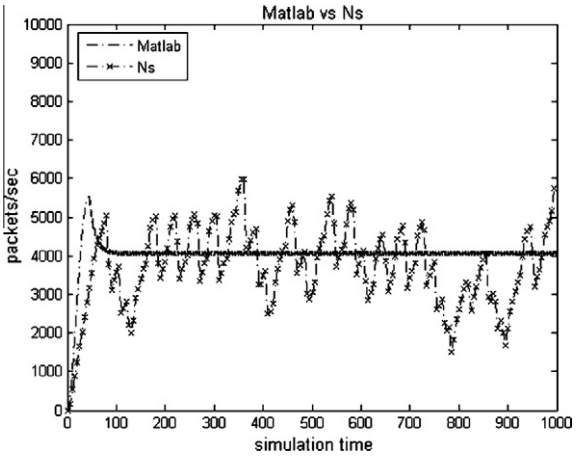


Fig. 6. Flow 2, with round-trip propagation delay = 83 ms, competing with flow 1 on a 100 Mbps bottleneck. $T_0 = 1$ and $k_1 = 2$.

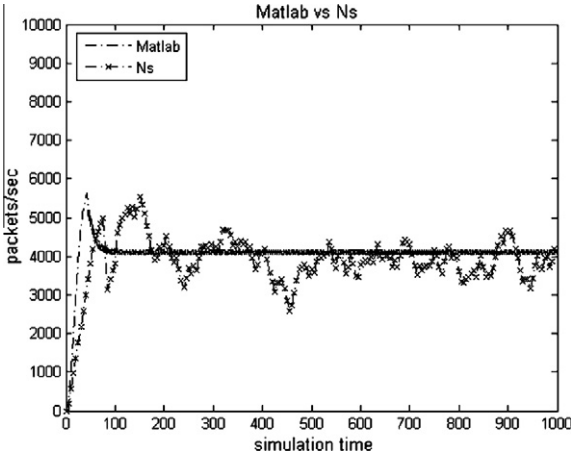


Fig. 7. Flow 2, with round-trip propagation delay = 83 ms, competing with flow 1 on a 100 Mbps bottleneck. $T_0 = 4$ and $k_1 = 8$.

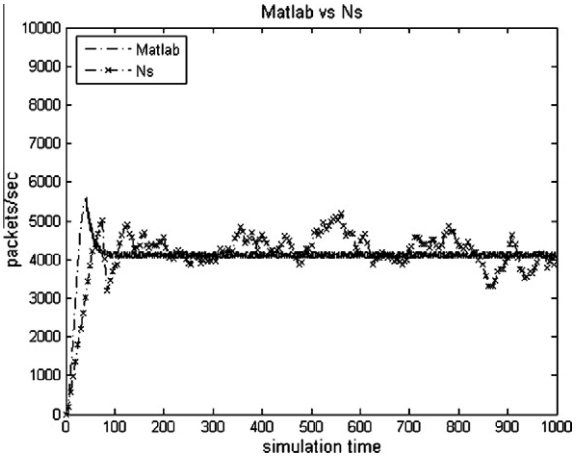


Fig. 8. Flow 2, with round-trip propagation delay = 83 ms, competing with flow 1 on a 100 Mbps bottleneck. $T_0 = 8$ and $k_1 = 16$.

Libra parameters shows a good tradeoff in all performance measures.

As a result of this discussion we can summarize that k_2 should be big enough to preserve stability. Similarly, we could increase also both T_0 and k_1 to the aim of preserving stability; instead, T_1 does not enter into the stability discussion, but a small T_1 will limit the throughput variance (same effect on the decrease rule of choosing a high T_0). From a large number of simulations we observed that k_2 has also another key effect: it tunes the friendliness of TCP Libra to TCP New Reno. For this reason we have chosen $k_2 = 2$, as it shows a good degree of friendliness to TCP New Reno in a broad range of network/traffic conditions.

5. RTT-fair schemes: model-based comparison

Table 1 summarizes the qualitative behavior of TCP schemes in terms of RTT-fairness as can be derived also taking inspiration from [6,11–15,22,29]. The key parameter to observe is the steady state solution (i.e., the stable point).

In Fig. 9 we plot Column III, from the left, of Table 1. We are here assuming that: (i) the two flows experience the same maximum queuing delay, $RTT_{max} - RTT_{min}$; (ii) $\widetilde{RTT} \gg RTT_{min}$; and (iii) the two flows have the same steady state probability of loss and queuing delay. In Fig. 9 we plot the throughput ratio of two flows, with different RTTs, which are competing on the same narrow link. In this figure, RTT_1 and RTT_2 represent the average RTT of flow 1 and flow 2, respectively, whereas x_1 and x_2 are the average throughput of flow 1 and flow 2. As we can see in Fig. 9, when the RTT of flow 1 (i.e., RTT_1) is equal to 100 ms, TCP Libra's response function is very close to the constant increase algorithm. This means that, no matter what RTT ratio is between flow 1 and flow 2, they will almost always achieve the same rate. When T_1 is equal to 500 ms, TCP Libra still improves over the linear behavior of TCP New Reno. Likewise, [28] proves through measurements that 50% of the TCP flows experience a RTT equal to or below 100 ms. Therefore, TCP Libra performs as a constant increase algorithm in the great majority of cases. This approach is a tradeoff between fairness and stability. Fairness is affected by including the RTT, as shown in Fig. 9; however, for realistic RTTs, TCP Libra behaves very closely to a constant RTT algorithm and preserves RTT-fairness.

From the steady state solution, it is evident how TCP New Reno implements linear RTT-fairness; the throughput that a connection may expect depends inversely from the average RTT of the connection.

Table 1
Dynamic and equilibrium properties.

Algorithm	Stable point	$\tilde{x}_1/\tilde{x}_2$	Stepsize	Marginal utility	Congestion measure
NewReno	$\frac{1}{\widetilde{RTT}_r}\sqrt{\frac{a_r}{b_r}\frac{1-\tilde{s}_r}{\tilde{s}_r}}$	$\propto \frac{\widetilde{RTT}_2}{\widetilde{RTT}_1}$	$\left(\frac{b_r}{a_r}\right)x_r^2(t) + \frac{1}{\widetilde{RTT}_r^2}$	$\frac{1}{\frac{b_r}{a_r}\widetilde{RTT}_r^2 x_r^2(t)+1}$	Loss probability
Hybla	$\frac{1}{T_0}\sqrt{\frac{a_r}{b_r}\frac{1-\tilde{s}_r}{\tilde{s}_r}}$	$\propto 1$	$a_r\left(\frac{b_r}{a_r}x_r^2(t) + \frac{1}{T_0^2}\right)$	$\frac{1}{\frac{b_r}{a_r}T_0^2 x_r^2(t)+1}$	Loss probability
Vegas	$\frac{1}{\phi_r}$	$\propto 1$	$\frac{1}{\widetilde{T}_r^2}$	$\frac{1}{x_r}$	Queueing delay
Libra	$\sqrt{\frac{a_r}{b_r}\frac{\alpha_r}{T_1}\frac{1-\tilde{s}_r}{\tilde{s}_r}}$	$\propto e^{-\frac{k_2}{2}(\widetilde{RTT}_1 - \widetilde{RTT}_2)}$	$\frac{b_r T_1}{\widetilde{RTT}_r + T_0}x^2(t) + \frac{a_r \tilde{x}_r}{\widetilde{RTT}_r + T_0}$	$\frac{1}{\frac{b_r T_1}{a_r \tilde{x}_r}x^2(t)+1}$	Loss prob. and queuing delay

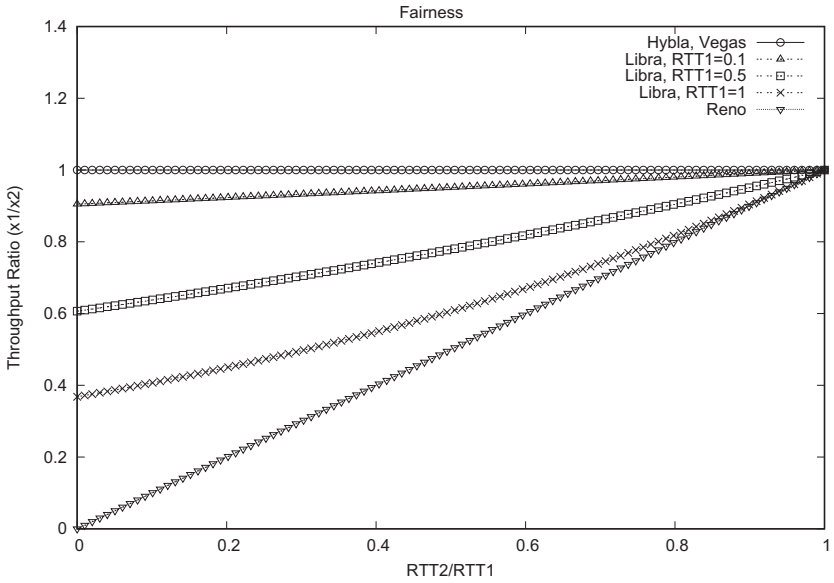


Fig. 9. Throughput ratio between two flows, for varying RTT ratios.

TCP Hybla is a constant increase algorithm that forces all flows to act as if they saw the same RTT: in fact, Table 1 shows that the stable point does not depend on the RTT, but on T_0, which is constant for all flows. In this way the algorithm is ideally fair as we see in Fig. 9. Yet, in [14], through a stability analysis, it is highlighted that constant increase algorithms experience delay instability on routes where ratio RTT_r/x_r is large and slow convergence on routes where the ratio is small. To confirm this statement, extensive simulation results in Section 6 show how this problem can lead to unfairness.

TCP Vegas implements an RTT-fair scheme. From simulation results we verify that TCP Vegas improves TCP's RTT-fairness. However, the problem of TCP Vegas is the choice of congestion measure, namely, queuing delay. This measure makes TCP Vegas too timid while competing with TCP New Reno (or with similar legacy protocols: see results about TCP SACK in Section 6).

Finally, Fig. 9 shows that TCP Libra is not constant RTT-fair as TCP Vegas or TCP Hybla, but it approaches such limit as RTT_1 approaches zero and it theoretically improves TCP New Reno. Furthermore, in Section 6 we show how TCP Libra's algorithm results both RTT-fair and friendly towards TCP New Reno, with a choice of parameters that works for a broad set of tests, whereas other RTT-fair protocols do not.

6. Performance evaluation

We have used the NS2 platform to evaluate TCP Libra [30]. We have divided our experiment campaign into three main sets. In the first one, we have created a simple simulative scenario, i.e., a dumbbell topology, and considered the protocols discussed and modeled in the previous sections: (i) TCP New Reno (with the SACK option enabled, i.e., TCP SACK), (ii) our TCP Libra, and other RTT-fair TCP versions such as (iii) TCP Vegas, and (iv) TCP Hybla; furthermore, we have also added CUBIC as this protocol represents the default TCP version on current Linux releases (since kernel 2.6.18 [31]).

For TCP SACK and Vegas, we have used existing NS2 modules; for TCP Hybla and CUBIC we have used the code provided by their respective developers; and we have created our own module for TCP Libra. Default parameters have been changed in the case of TCP Vegas as inspired by the literature [32].

The purpose of this first set of experiments is that of achieving deep understanding of protocols discussed in Section 5 (including TCP Libra), confirming their properties in a controlled and noise free scenario.

Instead, the aim of the second set of experiments is that of testing TCP Libra in a highly realistic simulative scenario, as similar as possible to the real world. We have hence taken large inspiration from [33] and created a complex scenario with background cross traffic, different queue sizes, and both drop-tail and RED queue management. The importance of including background traffic lies in its ability to prevent phase effects and in its impact on the fairness and convergence of the protocols [33,34]. The presence of background traffic causes noise in the RTT measurements, which are an important component of TCP Libra's algorithm; we have hence to consider also this factor in order to enhance the trustworthiness of achieved results.

Finally, as a confirmation for excellent simulative results achieved by our TCP Libra, we have tested our transport protocol even in a real network testbed scenario. Real experiments are generally more difficult to be performed than simulations. Yet, they embody an unrivaled testbed scenario as no simulation can generate the same realism.

In the following subsections, we present the three aforementioned experiment settings in detail. Where not differently stated, TCP packet size has been set equal to 1500 Bytes, all simulations have been run for 1000 s in order to reach steady state, and the advertised window for each connection has been set larger than the corresponding pipe size so that occasional packets may be dropped, even when that connection is the only active one.

6.1. Experiment setting #1

The simulated network topology for the first set of experiments is reported in Fig. 10. Four FTP connections are established between the source–destination pairs (S_i and R_i shown in Fig. 10). The pairs S_1–R_1 and S_2–R_2 have round-trip propagation delay equal to 40 ms, thus representing intra-continental links. The pairs S_3–R_3 and S_4–R_4 have round-trip propagation delay equal to 161 ms, representing inter-continental links. The link X–Y embodies the shared narrow link. The buffer size in node X is set either with a value suggested for Cisco Systems routers (i.e., 200 or 500 packets [35]), or as the product of the narrow link capacity by the largest round-trip propagation delay (i.e., 161 ms in most of the simulations) divided by packet size. In the remainder of this paper we refer to this latter value as the *longest pipe size*.

In this context, we focus on the following performance measures:

(1) intra-protocol RTT-fairness (Section 7.1);
(2) inter-protocol RTT-fairness (Section 7.2);
(3) scalability of TCP Libra to many flows (Section 7.3).

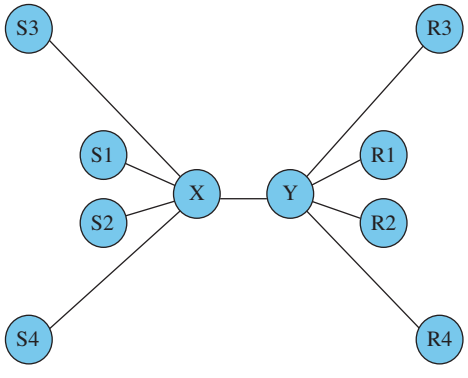


Fig. 10. Dumbbell topology for experiment setting #1.

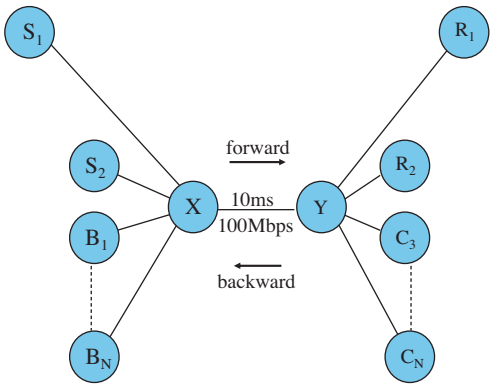


Fig. 11. Dumbbell topology for experiment setting #2.

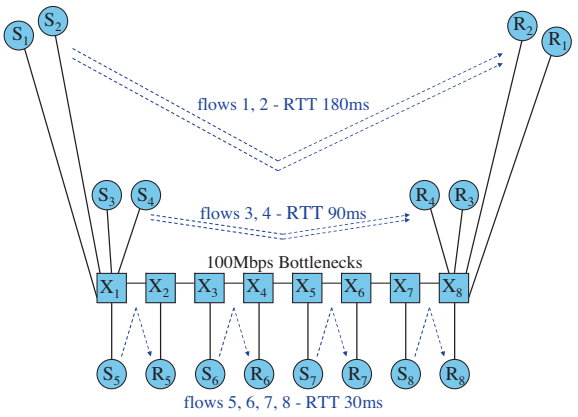


Fig. 12. Parking lot topology for experiment setting #2.

6.2. Experiment setting #2

In the second experiment setting, we compare again the same set of transport protocols considered in the first set of experiments. However, we focus on two different topologies, each featuring different scenarios.

First, we have considered again the dumbbell topology; but, in this case, we have adapted the simulation script provided on the BIC/CUBIC website [31]. We have chosen those scripts as they improve the classic dumbbell topology by including background traffic. Indeed, each link is configured to have different RTTs and different start times and end times to reduce the phase effect [36,34]. Being more precise with the help of Fig. 11, the one way propagation delay on the links is 21 ms for the short connections (from S_2 to R_2 and between B_i and C_i) and 119 ms for the longest one (from S_1 to R_1). Background traffic flows in both directions between nodes B_i to C_i. This background traffic is composed by 4 forward regular long-lived TCP SACK flows, 4 backward regular long-lived TCP SACK flows, 25 small TCP flows with advertised window limited to 64 segments and an amount of web traffic in both directions able to occupy from 20% to 50% of the available bottleneck link capacity when no other flow is present [24,31].

Second, following the suggestions provided in [33] about considering different network topologies in simulative experiments, we have considered also the so called parking lot topology (see Fig. 12). In particular, this topology includes eight end-to-end flows: flows 1 and 2 have 180 ms of minimum RTT and traverse 9 links; flows 3 and 4 have 90 ms of minimum RTT and traverse 9 links too. The remaining flows, 5–8, are short flows; they utilize 3 link paths with 30 ms minimum RTT. To overcome phase effects, flows were started at random times within the first 5 s of simulation [36]. The bottleneck buffer can take two different values: (a) the number of packets that would fill the bottleneck link or (b) the packets that would fill the longest path.

Space limitations allow us to present only a subset of the obtained simulation results. Therefore, we report here only on results related to the case with a 100 Mbps bottleneck link, since the use of other bandwidth values did not show significant difference.

Results obtained through this experiment setting are discussed in Section 7.4.

6.3. Experiment setting #3

After the comprehensive simulative comparison of different TCP protocols, we also provide results attained through a real testbed evaluation. The test has been conducted comparing TCP Libra against legacy TCP SACK.

The testbed is simply composed of two end hosts and a dummynet bridge. The two end hosts run Linux Kernel 2.6.14, whereas the bridge runs FreeBSD 6.1 with kernel polling enabled and Dummynet [37] configured to simulate a 100 Mbps link with a 250 packets queue size. As for the one-way propagation delays, 100, 150, and 200 ms have been set associating different delays to different ports. Finally, Iperf 2.0.2 has been used to generate the network load.

Each experiment has been run for 900 s and, during this time, three concurrent TCP flows using the same transport protocol enter and exit at different times: a 150 ms one-way propagation delay flow runs from the beginning to the end, a 100 ms one is active during the 180–720 s time frame, and a 200 ms one transmits between 360 and 540 s.

Our comparison aims at experimentally confirming: (i) the unfairness problem of legacy TCP protocols and (ii) TCP Libra flows' capability to converge to an equal share, even when experiencing very different propagation delays.

Results obtained through this experiment setting are shown in Section 7.5.

7. Results

In this section we present the outcome of the experiment settings discussed in Section 6. In particular, Sections 7.1, 7.2, and 7.3 refer to the simple simulative setting discussed in Sections 6.1, 7.4 refers to the complex simulative setting discussed in Sections 6.2 and 7.5 refers to the real testbed experiment setting discussed in Section 6.3.

7.1. Experiment setting #1: intra-protocol fairness

Jain's Fairness Index is adopted in the literature to evaluate the fairness degree of data flows that share a single narrow link [38]. We have computed its value for the

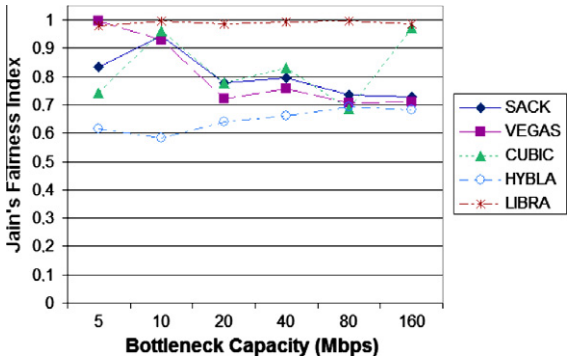


Fig. 13. Jain's Fairness Index vs. narrow link capacity for TCP SACK, TCP Vegas, CUBIC, TCP Hybla, and TCP Libra. Buffer size at the narrow link is equal to the *longest link pipe size*.

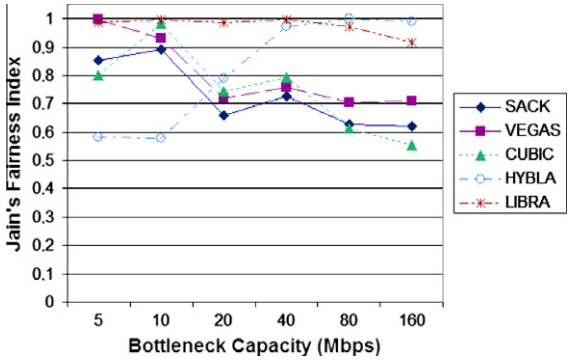


Fig. 14. Jain's Fairness Index vs. narrow link capacity for TCP SACK, TCP Vegas, CUBIC, TCP Hybla, and TCP Libra. Buffer size at the narrow link is equal to 200 packets: suggested default value in the CISCO Systems configuration manual(s) [35].

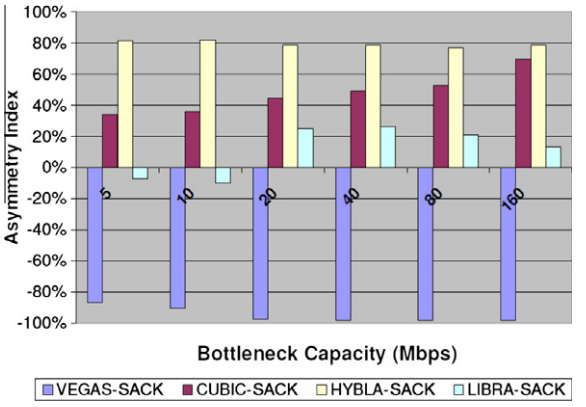


Fig. 15. SLAC Asymmetry Index [39]. Buffer size at the narrow link is equal to the *longest link pipe size*.

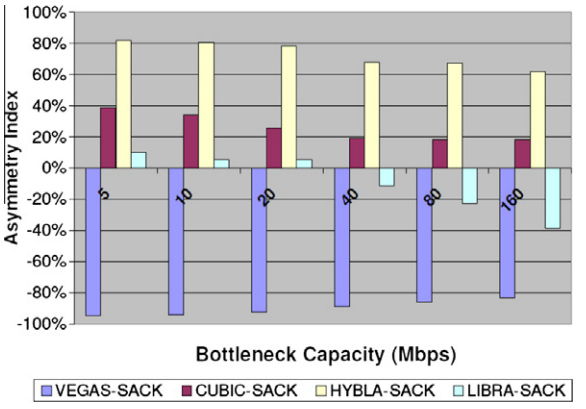


Fig. 16. SLAC Asymmetry Index [39]. Buffer size at the narrow link is equal to 200 packets [35].

tested protocols while considering different narrow link capacities and buffer sizes. In Fig. 13 the buffer size is set equal to the longest pipe size, while in Fig. 14 the buffer size corresponds to 200 packets. TCP Libra shows a better RTT-fairness than its competitors in almost all the considered scenarios. An exception is TCP Hybla that shows a slightly better RTT-fairness for a narrow link of 80 Mbps and 160 Mbps, and 200-packet buffer. Indeed, TCP Hybla was specifically intended to be RTT-fair through proportionally increasing the congestion window of long-RTT flows so as to make them behave like a reference-RTT (i.e., 25 ms) flow. However, this solution works if the queuing delay of buffers along the path does not significantly modify the ratio between the minimum RTTs of the various flows. This means that TCP Hybla can ensure RTT-fairness in case of big pipes and small buffers, simultaneously present, as demonstrated by the charts.

7.2. Experiment setting #1: inter-protocol fairness (Friendliness)

We here evaluate the case where a TCP variant competes on the same narrow link with legacy TCP SACK flows. In this setting TCP SACK is used for one short round-trip propagation delay flow (from S2 to R2 in Fig. 10) and one long round-trip propagation delay flow (from S4 to R4 in Fig. 10), while the remaining two concurrent flows are driven by one of the other TCP flavors. This allows us to investigate the impact of the coexistence on the RTT-fairness degree and the friendliness of the alternative TCP versions towards TCP SACK. The SLAC Asymmetry Index [39] is used as the friendliness metric; this index is defined as:

$$A = \frac{\bar{x}_1 - \bar{x}_2}{\bar{x}_1 + \bar{x}_2}, \tag{20}$$

where $\bar{x}_1$ and $\bar{x}_2$ correspond to the average throughput values achieved by two different protocols competing for the same channel. The index can be employed to linearly indicate the degree of aggressiveness between two protocols. In essence, when $A = 0$, the two protocols evenly share the narrow link. Conversely, $A > 0$ indicates that x_1 is more aggressive than x_2, whereas $A < 0$ implies the inverse situation.

In Figs. 15 and Fig. 16 we report the SLAC Asymmetry Index when TCP SACK is coexisting alternatively with TCP Vegas, CUBIC, TCP Hybla, and TCP Libra. In the corresponding simulation, two TCP SACK flows with round-trip

propagation delays set to 40 ms and 161 ms, respectively, compete with two other flows (again, with round-trip propagation delays set to 40 ms and 161 ms). An index value equal to zero means perfect friendliness; an index value lower than zero means that the considered transport protocol is more conservative than TCP SACK when coexisting; an index value higher than zero means that the considered transport protocols is more aggressive than TCP SACK when coexisting. In Fig. 15 the buffer size is set to the longest pipe size (2133 packets), whereas in Fig. 16 the buffer size is much smaller: 200 packets, as suggested default value in the CISCO Systems configuration manual (s) [35].

The charts show that when TCP Vegas competes against TCP SACK, the latter is able to reach higher throughputs (the Asymmetry Index for TCP Vegas is: $A < 0$). This was expected; indeed, when TCP Vegas detects that a queue is building up, it reduces the transmission rate. Instead, TCP SACK keeps probing the channel and gaining shares of bandwidth. As a consequence, TCP Vegas quickly brings its congestion window to a low value, thus achieving a low throughput level.

Conversely, TCP Hybla shows aggressiveness towards TCP SACK, thus using most of the bandwidth and reducing TCP SACK's throughput. In fact, TCP Hybla's Asymmetry Index is positive in all cases. This is coherent with the fact that TCP Hybla increases its transmission rate as if it were experiencing a pre-defined RTT value (i.e., 25 ms [22]), ignoring the factual RTT value. Since the competing TCP SACK flows see a round-trip propagation delay of 40 ms and 161 ms, their bandwidth share is obviously less than the bandwidth share of TCP Hybla flows.

When competing with concurrent TCP SACK flows, CUBIC is neither as conservative as TCP Vegas, nor as aggressive as TCP Hybla. However, in general, TCP Libra showed an even better friendliness degree. In summary, TCP Libra shows to be able to fairly share the available bandwidth with TCP SACK. The only configuration where another protocol (i.e., CUBIC) achieves a better asymmetry index than TCP Libra is when a small buffer of 200 packets is employed in combination with a 160 Mbps narrow link (see the rightmost series of columns in Fig. 16).

As a final note, we have to report that TCP Libra also attained intra-protocol fairness when coexisting with TCP SACK. In fact, the simulation results showed that the two TCP Libra connections tend to achieve the same throughput, which is close to the mean throughput of the two TCP SACK connections.

7.3. Experiment setting #1: TCP Libra scalability discussion

We here study how TCP Libra scales in relation to the number of flows sharing the same narrow link. To this aim, we have set the dumbbell configuration presented in Fig. 10 with a narrow link of 622 Mbps (i.e., an OC12 link). We perform three experiments involving 110 contemporaneous flows each. The round-trip propagation delay is set to the following values:

- 16 ms for 30 flows (i.e., a regional connection);
- 61 ms for 60 flows (i.e., an intra-continental connection);
- 181 ms for 20 flows (i.e., an inter-continental connection).

TCP SACK performance is shown in Fig. 17 in terms of acknowledged packets per time for each of the 110 simulated flows. As expected, TCP SACK is affected by heavy

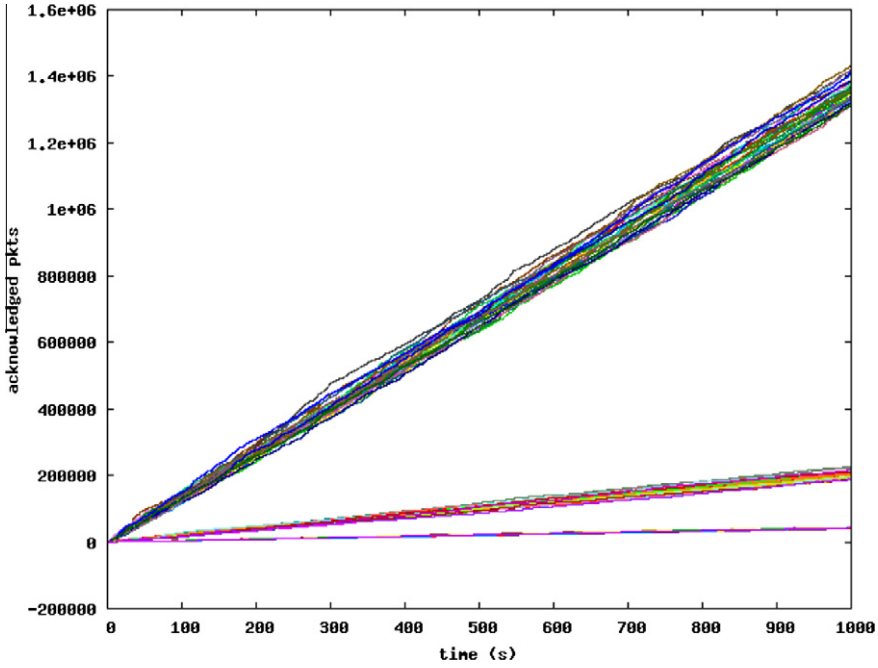


Fig. 17. Acknowledged packets for TCP SACK. Narrow link of 622Mbps (i.e., OC12), 110 flows with heterogeneous RTTs. Buffer size at the narrow link has been set equal to 500 packets: suggested default value for high speed core routers in the CISCO Systems configuration manual (s) [35].

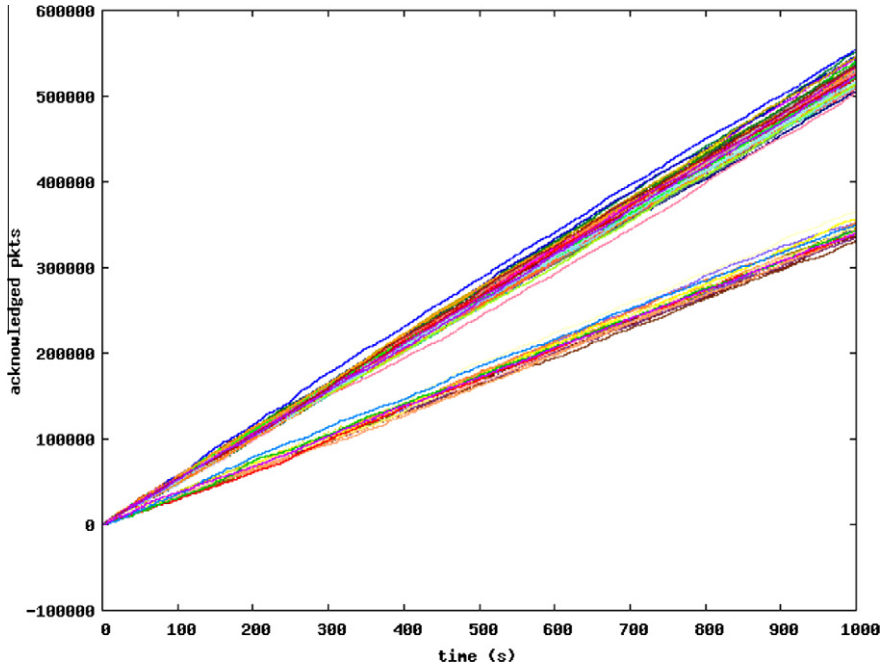


Fig. 18. Acknowledged packets for TCP Libra with $k_2 = 2$. Narrow link of 622 Mbps (i.e., OC12), 110 flows with heterogeneous RTTs. Buffer size at the narrow link has been set equal to 500 packets: suggested default value for high speed core routers in the CISCO Systems configuration manual (s) [35].

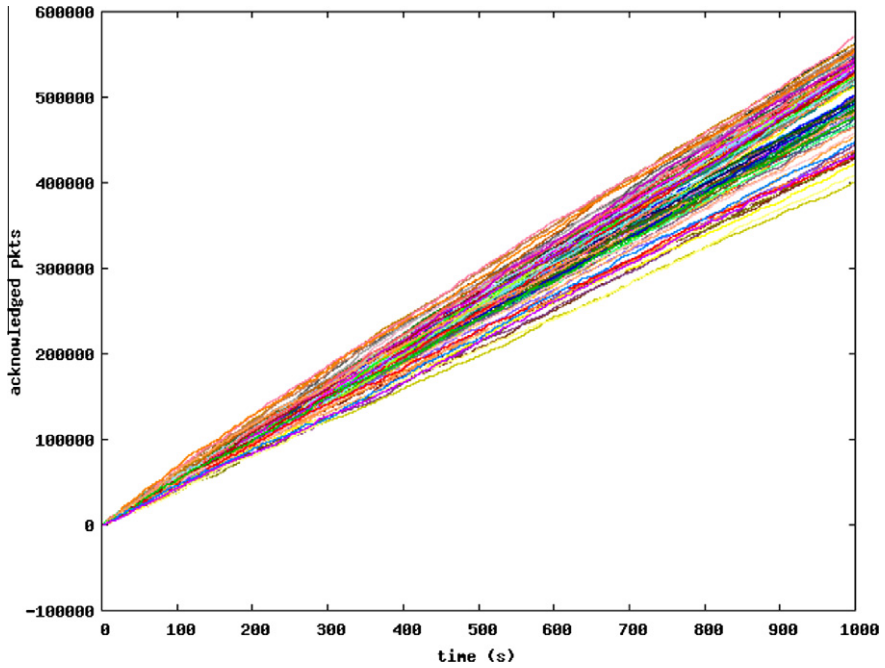


Fig. 19. Acknowledged packets for TCP Libra with $k_2 = 12$. Narrow link of 622 Mbps (i.e., OC12), 110 flows with heterogeneous RTTs. Buffer size at the narrow has been set equal to 500 packets: suggested default value for high speed core routers in the CISCO Systems configuration manual (s) [35].

RTT-unfairness and generates three distinct clusters of lines with a wide gap in between. Using the same experiment scenario and metric, we evaluate TCP Libra's performance. In particular, Fig. 18 and Fig. 19 show TCP Libra's results when employing $k_2 = 2$ or $k_2 = 12$, respectively. As it is evident from the charts, both with $k_2 = 2$ and with $k_2 = 12$, TCP Libra achieves a better RTT-fairness degree among its flows than TCP SACK does. Yet, in this simulative

configuration, Fig. 19 shows a better fairness than Fig. 18. This outcome can be explained through the fact that in the configuration referring to Fig. 19 TCP Libra operates at the boundaries of its stability region defined by (18); by increasing k_2 from 2 to 12, it safely returns to stability and achieves a good RTT-fairness level as shown by Fig. 19.

7.4. Experiment setting #2: complex simulative scenarios

In this subsection we compare the performance of the considered transport protocols, utilizing the complex simulative configuration explained in Section 6.2.

We start with the dumbbell topology including also a significant amount of background traffic, both forward and backward. For the sake of conciseness, we show here only the outcome for the case in which the bottleneck buffer is small (i.e., equal to bottleneck link pipe size): the most demanding case for the transport protocols. Two different queuing policies are tested: drop tail and RED (Random Early Detection). The Jain's index values are reported in Fig. 20. Clearly, TCP Libra and TCP Hybla outperform the other protocols in terms of provided RTT-fairness; among the other protocols, CUBIC performs better than TCP Vegas and TCP SACK.

Results in Fig. 20 also confirm the partial RTT-independency of CUBIC we mentioned in Section 2. This protocol tries to decouple the window growth from the returning ACK process. With CUBIC, the window size is a function of the time elapsed since the last packet loss, thus allowing higher efficiency (in terms of total bandwidth utilization) in case of long fat RTTs and reducing the throughput dependency from the RTT. Yet, this smart solution is not able to completely eliminate the RTT-unfairness as, even if the window size becomes independent from the RTT, the final throughput does not. The throughput still corresponds to the ratio between the window size and the RTT, where two flows with similar packet loss trend may have the same window but different RTTs. For instance, if we link the window size to the time elapsed since the last packet loss, two flows with different RTTs (say, RTT_1 and RTT_2) but same time elapsed since the last loss may have the same window (say, W). This implies that W bytes are sent in RTT_1 and RTT_2 seconds by the two flows, respectively, before sending out another window of data. As evident, even with the same window the throughputs will be different. Of course, this is better than using regular TCP as, in that case, we also have larger windows for smaller RTT flows. Instead, TCP Libra approach tries to eliminate the RTT-bias by having a higher window growing rate for flows with longer RTT. In this way, if $RTT_1 < RTT_2$, the two windows (W_1 and W_2) will be computed so that $W_1 < W_2$. This improves the RTT-fairness with respect to CUBIC. On the other hand, our solution is not specifically designed for long fat pipes thus resulting less efficient in terms of total bandwidth utilization.

Another interesting property shown in Fig. 20 is that RED improves fairness; this result is a consequence of the fact that RED was indeed designed to prevent capture by aggressive flows.

Focusing on the parking lot topology, in Fig. 21 we report the Jain's index values considering for its computation only flows 1–4. Both the bottleneck pipe size and the longest pipe size have been considered as the bottleneck buffer size. As evident, TCP Libra is the only protocol among the considered ones that provides good fairness in both cases. Indeed, the *penalty* factor in Libra adapts the window increase slope to the relative backlog time, thus reducing sensitivity to buffer size. The chart also shows a high Jain's index for TCP SACK in the case with the longest pipe as buffer size. This apparently surprising result is indeed due to the fact that the flows 5–8 (i.e., the short-

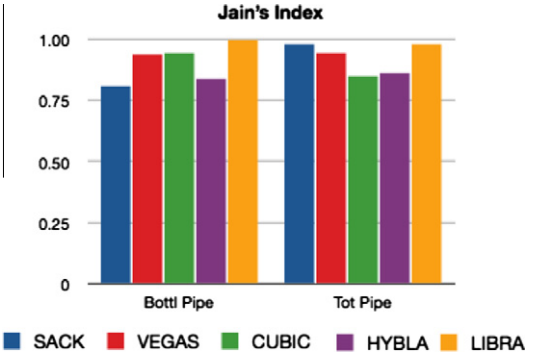


Fig. 21. Jain's index values among flows 1–4, for each different protocol; parking lot topology.

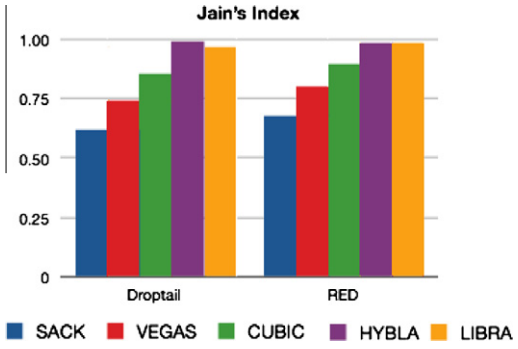


Fig. 20. Jain's index values achieved by the evaluated protocol while competing with a large amount of concurrent traffic.

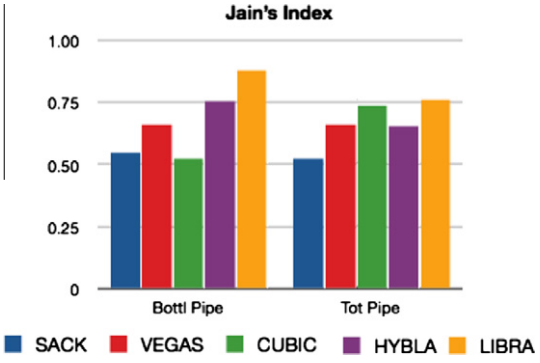


Fig. 22. Jain's index computed over all the 8 connections; parking lot topology.

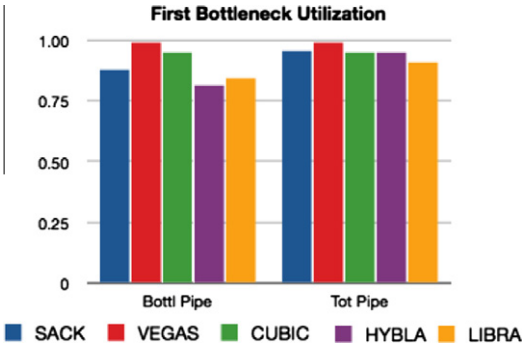


Fig. 23. Efficiency in the parking lot topology.

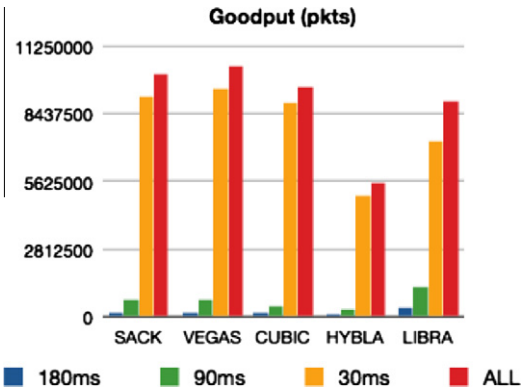


Fig. 24. Friendliness evaluation. Goodput achieved by TCP SACK flows (on *y*-axis) when another transport protocol (on *x*-axis) is supporting half of the concurrent flows.

RTT flows) capture the whole bandwidth thus leaving flows 1–4 with very low (and similar) bandwidth.

This is confirmed by the Jain's index resulting when considering both short RTT connections 5–8 and longer RTT connections 1–4 (see Fig. 22). A significant fluctuation of fairness index is noted and TCP SACK, which had an acceptable Jain's index if computing only the goodputs of connections 1–4, obtains a very poor value when considering all the eight connections together.

The throughput efficiency on the first bottleneck of the parking lot topology is shown in Fig. 23. TCP Libra's utilization of the first bottleneck is somehow conservative to allow the short flows on subsequent bottlenecks to better exploit the shared channel and achieve a better fairness. Furthermore, as expected, the utilization of the available bandwidth increases with the buffer size, for all protocols.

Finally, the coexistence between the legacy TCP (i.e., TCP SACK) and the new protocols is evaluated in Fig. 24. The bar chart presents the relative TCP SACK goodputs when TCP SACK is competing with itself first, and then with each of the new protocols. We measure the TCP SACK goodput achieved in each of the RTT flow classes (long to short) as well as the aggregate goodput over all of its connections. More precisely, TCP SACK was used for flows 2 (180 ms of RTT), 4 (90 ms of RTT), 6 and 8 (30 ms of RTT each), while the new protocol was used for flows 1 (180 ms of RTT), 3 (90 ms of RTT), 5 and 7 (30 ms of RTT each).

As expected, when coexisting with TCP Vegas, TCP SACK achieves a slight increase in its achieved goodput, whereas the aggressive behavior of TCP Hybla penalizes concurrent TCP SACK flows. CUBIC and TCP Libra do not harm significantly concurrent TCP SACK flows. In particular, CUBIC shows a balanced behavior toward TCP SACK, whereas

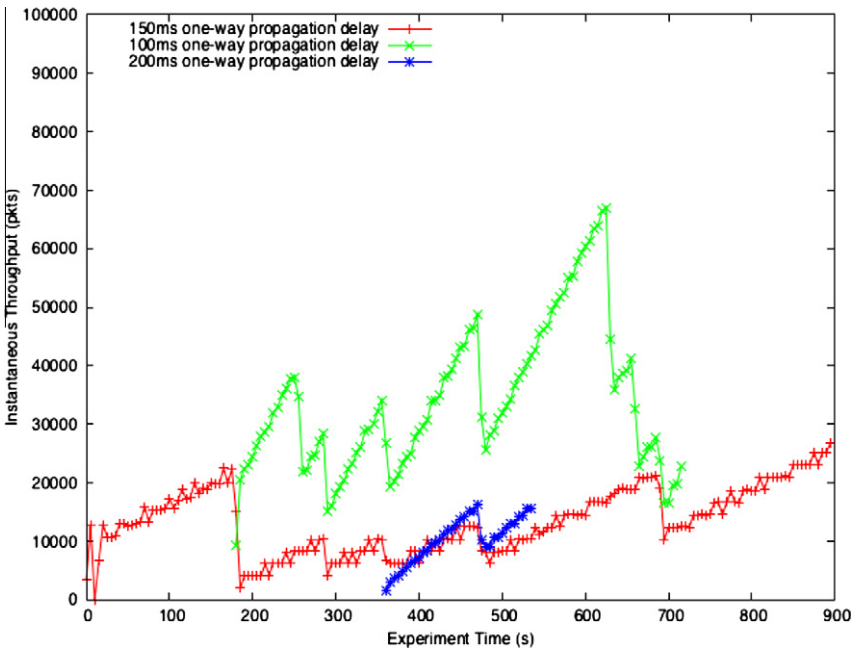


Fig. 25. Dynamic Scenario: instantaneous throughput of three TCP SACK flows.

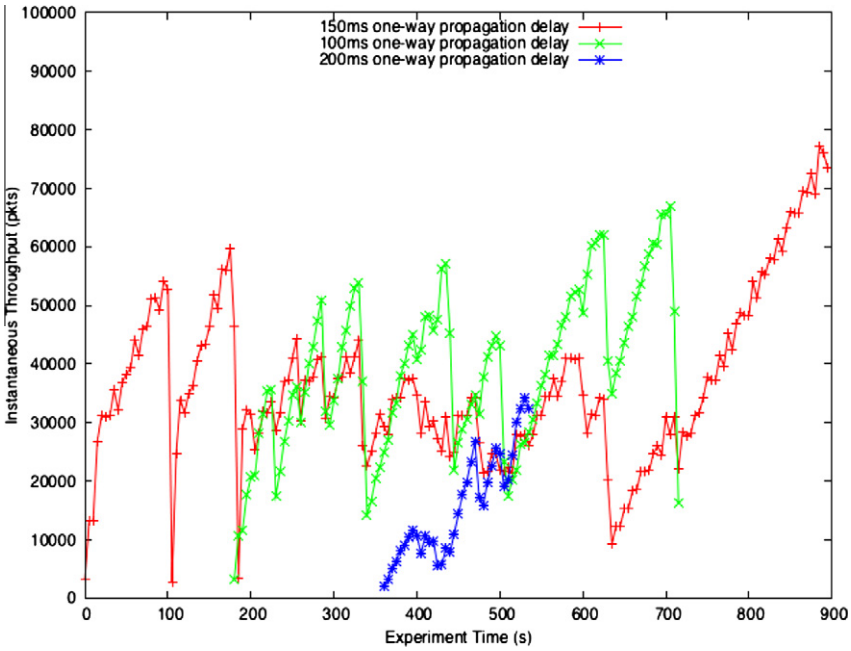


Fig. 26. Dynamic Scenario: instantaneous throughput of three TCP Libra flows.

the aggregate throughput of the TCP SACK flows diminishes by 11% when coexisting with TCP Libra. However, there is a desirable, even if limited, redistribution of the TCP SACK goodput from the 30 ms RTT flows to the 90 ms and 180 ms RTT ones when coexisting with TCP Libra flows; hence, TCP Libra seems to help the coexisting TCP SACK flows by (slightly) improving their fairness degree.

7.5. Experiment setting #3: real testbed experiments

In this subsection we compare the performance of our TCP Libra against a legacy TCP version, TCP SACK, utilizing the real experiment testbed discussed in Section 6.3.

In Figs. 25 and 26, we depict the instantaneous throughput achieved by three concurrent TCP flows when employing one of the two transport protocols we are considering for comparison: TCP SACK and TCP Libra, respectively. In each of the charts it is possible to clearly see the different activity periods of the three flows. In summary, the smallest RTT flow starts (and ends) for second, whereas the longest RTT flow starts (and ends) for third. It is hence interesting to see whether the employed transport protocol privileges one of the flows or behaves fairly.

To this aim, Fig. 25 confirms how TCP SACK generates a RTT-unfair share of the available bandwidth. Indeed, the flow with the smallest RTT unfairly captures most of the shared bandwidth as soon as it starts transmissions. Moreover, as expected, the flow having longer RTTs are characterized by a slowly growing instantaneous throughput.

Instead, Fig. 26 depicts the behavior of TCP Libra flows in the same scenario. Regardless of the RTT and of the start/end time, all the three flows converge to an equal share of the bandwidth resource when coexisting. This is clearly visible in the chart around 500 s, when the three flows have very similar throughput values and growth rates. Even before, from 200 s to 350 s, when only two flows where competing for the shared bandwidth, they have very similar throughput and trend.

Therefore, comparing these two preliminary real-testbed evaluations, it is evident how TCP Libra confirms its ability in reaching the fair share of the channel, even in dynamic conditions and regardless of RTT differences. These experimental results are coherent with what we have obtained through dumbbell topology simulation in Fig. 20, in the case with drop tail queue management of the buffer, which is the simulation configuration that more closely resembles the setting of our real testbed.

8. Conclusions and future work

This paper analyzes TCP Libra, a new protocol designed to be RTT-fair while maintaining a good friendliness towards legacy TCP and providing bandwidth scalability. We have showed how TCP Libra can be analytically derived from TCP New Reno, explained the differences with previous approaches aimed at building an RTT-fair TCP, and analyzed its stability bounds. Furthermore, we have also experimentally confirmed TCP Libra's qualities, comparing it against other RTT-fair TCPs in different simulative settings and in demanding scenarios, even considering an OC12 link shared by more than a hundred of concurrent flows. Finally, we have also provided preliminary results of a real testbed implementation based on Linux stack that confirms the efficacy of our new protocol.

In summary, we have presented here a really comprehensive evaluation of the interesting properties possessed by TCP Libra, exploiting model analysis, simulations, and

real testbed experiments, whereas the vast majority of papers in the literature consider only one or two among these three methods. The encouraging results achieved strongly motivate us in continuing this work to deploy a final product that could be factually exploited to improve the Internet.

References

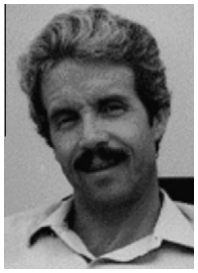
[1] V. Jacobson, Congestion avoidance and control, in: ACM SIGCOMM'88, Stanford, CA, USA, August 1988.
[2] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, Rfc2018: TCP selective acknowledgment options, Network Working Group, Technical Report, 1996.
[3] R. Jain, A delay based approach for congestion avoidance in interconnected heterogeneous computer networks, Computer Communications Review, ACM SIGCOMM, vol. 19, October 1989, pp. 56–71.
[4] J. Martin, A.A. Nilsson, I. Rhee, The incremental deployability of RTT-based congestion avoidance for high speed TCP internet connections, in: ACM SIGMETRICS 2000, Santa Clara, CA, USA, June 2000.
[5] R.S. Prasad, M. Jain, C. Dovrolis, On the effectiveness of delay-based congestion avoidance, in: 2nd International Workshop on Protocols for Fast Long-Distance Networks, Argonne National Laboratory Argonne, IL, USA, February 2004.
[6] L.S. Brakmo, S.W. O'Malley, L.L. Peterson, TCP vegas: New techniques for congestion detection and avoidance, in: SIGCOMM'94, London, UK, August–September 1994.
[7] Z. Wang, J. Crowcroft, Eliminating periodic packet losses in the 4.3-tahoe BSD TCP congestion control algorithm, ACM Computer Communication Review 22 (1992) 9–16.
[8] C. Jin, D. Wei, S. Low, Fast TCP: motivation, architecture, algorithms, performance, in: IEEE INFOCOM 2004, Hong Kong, China, March 2004.
[9] C. Jin, D. Wei, S.H. Low, G. Buhrmaster, J. Bunn, D.H. Choe, R.L.A. Cottrell, J.C. Doyle, W.C. Feng, O. Martin, H. Newman, F. Paganini, S. Ravot, S. Singh, Fast TCP: from background theory to experiments, IEEE Network 19 (2005) 4–11.
[10] G. Marfia, C.E. Palazzi, G. Pau, M. Gerla, M. Sanadidi, M. Roccetti, TCP Libra: exploring RTT-fairness for TCP, in: IFIP/TC6 Networking Conference, NETWORKING 2007, Atlanta, GA, USA, May 2007.
[11] J. Padhye, V. Firoiu, D. Towsley, J. Kurose, Modeling TCP throughput: a simple model and its empirical validation, in: ACM SIGCOMM '98, Vancouver, BC, Canada, August–September 1998.
[12] F.P. Kelly, A.K. Maulloo, D.K.H. Tan, Rate control in communication networks: shadow prices, proportional fairness, and stability, Journal of the Operational Research Society 49 (1998).
[13] R. Gibbens, F. Kelly, Resource pricing and the evolution of congestion control, Automatica 35 (12) (1999) 1969–1985.
[14] F. Kelly, Mathematical modelling of the internet," in: Bjorn Engquist, Wilfried Schmid (Eds.), Mathematics Unlimited – 2001 and Beyond@Springer, 2001.
[15] F. Kelly, Fairness and stability of end-to-end congestion control, European Journal of Control 9 (2003) 159–176.
[16] F. Paganini, Z. Wang, S. Low, J.C. Doyle, A new TCP/AQM for stable operation in fast networks, in: IEEE INFOCOM 2003, San Francisco, CA, USA, March–April 2003.
[17] S. Athuraliya, D.E. Lapsley, S.H. Low, An enhanced random early marking algorithm for internet flow control, in: IEEE INFOCOM 2000, Tel Aviv, Israel, March 2000.
[18] S. Low, R. Srikant, A mathematical framework for designing a low-loss, low-delay internet, Networks and Spatial Economics 4 (2003) 75–102.
[19] S. Floyd, V. Jacobson, Traffic phase effects in packet-switched gateways, Internetworking: Research and Experience 3 (1992) 115–156.
[20] T. Henderson, Networking Over Next-generation Satellite Systems, Ph.D. Dissertation, University of California, Berkeley, 1999.
[21] T.R. Henderson, R.H. Katz, Transport protocols for internet-compatible satellite networks, IEEE Journal on Selected Areas in Communications 17 (1999) 326–344.
[22] C. Caini, R. Ferrincelli, TCP Hybla: a TCP enhancement for heterogeneous networks, International Journal of Satellite Communications and Networking 22 (2004) 547–566.
[23] I. Rhee, L. Xu, Cubic: a new TCP-friendly high-speed TCP variant, in: 3rd International Workshop on Protocols for Fast Long-Distance Networks, Lyon, France, February 2005.
[24] L. Xu, K. Harfous, I. Rhee, Binary increase congestion control for fast, long distance networks, in: IEEE INFOCOM, 2004, Hong Kong, China, March 2004.
[25] R. Shorten, D. Leith, H-TCP: TCP for high-speed and long-distance networks TCP, in: Second International Workshop on Protocols for Fast Long-distance Networks, Argonne, IL, USA, February 2004.
[26] R. Srikant, The Mathematics of Internet Congestion Control, A Birkhauser Book, 2004.
[27] D.P. Bertsekas, Nonlinear Programming, Athena Scientific, 1999.
[28] J. Aikat, J. Kaur, F.D. Smith, K. Jeffay, Variability in TCP round-trip times, in: 3rd ACM SIGCOMM Conference on Internet Measurement, New York, NY, USA, 2003.
[29] L.A. Grieco, S. Mascolo, Performance evaluation and comparison of westwood+, new reno, and vegas TCP congestion control, ACM Computer Communication Review 34 (2) (2004) 25–38.
[30] The vint project, ns2, nsnam. [Online] <http://nsnam.isi.edu/nsnam/>.
[31] BIC and CUBIC protocol – default TCP algorithm in linux. [Online]. Available: <http://netsrv.csc.ncsu.edu/twiki/bin/view/Main/BIC>.
[32] S.H. Low, L.L. Peterson, L. Wang, Understanding TCP vegas: a duality model, in: SIGMETRICS/Performance, Cambridge, MA, USA, June 2001.
[33] L. Andrew, C. Marcondes, S. Floyd, L. Dunn, R. Guillier, W. Gang, L. Eggert, S. Ha, I. Rhee, Towards a common TCP evaluation suite, in: 6th International Workshop on Protocols for FAST Long-Distance Networks (PFLDnet 2008), Manchester, UK, 2008.
[34] S. Mascolo, F. Vacirca, The effect of reverse traffic on TCP congestion control algorithms, in: 4th International Workshop on Protocols for Fast Long-distance Networks, Nara, Japan, February 2006.
[35] C.S. Inc. Buffer tuning for all cisco routers – document id: 15091. [Online] <http://www.cisco.com/warp/public/63/buffertuning.html>.
[36] S. Floyd, E. Kholer, Internet research needs better models, ACM SIGCOMM Computer Communication Review 33 (2003) 29–34.
[37] Dummynet home page. [Online] <http://info.iet.unipi.it/luigi/dummynet/>.
[38] R. Jain, D. Chiu, W. Hawe, A quantitative measure of fairness and discrimination for resource allocation in shared computer systems, DEC Research Labs, Technical Report TR-301, 1984.
[39] L. Cottrell, H. Bullot, R. Hughes-Jones (2004, February) Evaluation of advanced TCP stacks on fast long-distance production networks. Presentation at SLAC, Stanford. EPFL, SLAC and Manchester University. [Online]. <www.slac.stanford.edu/grp/scs/net/talk03/pfld-feb04.ppt>.

Gustavo Marfia received a Laurea degree in Telecommunications Engineering from the University of Pisa in 2003. He received a Ph.D. in Computer Science from the University of California, Los Angeles, in 2009. He is currently a Postdoctoral Researcher at the Department of Computer Science of the University of Bologna. His research interest include: ad hoc networks, transportation systems and multimedia streaming.

Claudio E. Palazzi received the M.S. degree in computer science from the University of California, Los Angeles (UCLA), in 2005, the Ph.D. degree in computer science from the University of Bologna, Bologna, Italy, in 2006, and the Ph.D. degree in computer science from UCLA in 2007, through the joint Ph.D. Program in Computer Science organized by the University of Bologna and UCLA. His Ph.D. thesis was focused on interactive online gaming over wired/wireless networks. He is currently an Assistant Professor with the Dipartimento di Matematica Pura e Applicata, Università degli Studi di Padova, Padova, Italy. His research is primarily in the area of protocol design and analysis for wired/wireless networks, with emphasis on network-centric multimedia entertainment and vehicular networks. Dr. Palazzi was the recipient of the Best Full Paper Award at the 3rd ACM International Conference on Computer Game Design and Technology and

the Best Paper Award at the Eurosis GAMEON'2007 International Conference for his Ph.D. thesis.

Giovanni Pau received the Italian Laurea degree in Computer Science and a Ph.D. in Computer Engineering, both from the University of Bologna, Bologna, Italy. He is currently a research scientist at UCLA. His area of expertise includes mobile computer network environment, fields in which he has co-authored over 60 technical refereed papers.



Mario Gerla is a Professor in the Computer Science at UCLA. He holds an Engineering degree from Politecnico di Milano, Italy and the Ph.D. degree from UCLA. He became IEEE Fellow in 2002. At UCLA, he was part of the team that developed the early ARPANET protocols under the guidance of Prof. Leonard Kleinrock. He joined the UCLA Faculty in 1976. At UCLA he has designed and implemented network protocols including ad hoc wireless clustering, multicast (ODMRP and CODECast) and Internet transport (TCP Westwood). He has lead the $12M, 6 year ONR MINUTEMAN project, designing the next generation scalable airborne Internet for tactical and homeland defense scenarios. He is now leading two advanced wireless network projects under ARMY and IBM funding. His team is developing a Vehicular Testbed for safe navigation, urban sensing and intelligent transport. A parallel research activity explores personal communications for cooperative, networked medical monitoring (see www.cs.ucla.edu/NRL for recent publications).

Marco Roccetti received the Italian Laurea degree in electronic engineering from the University of Bologna, Bologna, Italy. He has been a Full Professor with the Dipartimento di Scienze dell'Informazione, Università di Bologna since 2000. His research interests include digital audio and video, computer entertainment, and web-2.0-based applications, fields in which he has authored almost 250 technical refereed papers.