TCP *Libra*: Exploring RTT-Fairness for TCP

UCLA Computer Science Department Technical Report #TR050037

Gustavo Marfia*, Claudio Palazzi*†, Giovanni Pau*,
Mario Gerla*, M. Y. Sanadidi*, Marco Roccetti†,
*Computer Science Department - University of California Los Angeles, CA 90095
e-mail: {gmarfia|cpalazzi|gpau|gerla|medy}@cs.ucla.edu
†Dipartimento di Scienze del'Informazione, University of Bologna - Italy 40126
e-mail: roccetti@cs.unibo.it

Abstract

The majority of Internet traffic relies on the Transmission Control Protocol (TCP[1]) devised in the early 1970s to provide a reliable data transfer across the ARPANET. Today's users download large multimedia files from remote servers using TCP. If these TCP sessions share the same bottleneck, they are expected to receive the same share of bandwidth, thus achieving the same transfer rate. Unfortunately, this is not the case when the round trip delay RTT is very different among sessions. This may have negative practical implications in downloads of delay-sensitive (though non-real-time) information from servers. For instance, suppose in a popular internet cafè in New York City several users are simultaneously downloading multimedia files from various servers. Most of the servers are local. However, one customer is downloading a large file from a remote server in Australia. All customers share the same 11 MBPS WiFi bottleneck. The customer connected to Australia will make no visible progress until all the customers downloading from local servers are done! One solution would be to set up a time day when only remote customers download. But this is not quite application transparent. Here we propose a new version of TCP, namely TCP Libra, that guarantees fair sharing regardless of RTT. Libra in Latin means scale, thus indicating a balance between the sessions.

In this paper we describe the design of TCP Libra and prove that it is indeed RTT fair. The key element of TCP Libra is the window adjustment algorithm. The algorithm is derived by modeling TCP performance as a utility function and by optimizing this function such that the result is independent of RTT, yet maintaining the function stability. Non-linear optimization is used, leading to a solution that provides fairness among TCP flows that share the same bottleneck link regardless of RTT. The remarkable property of TCP Libra is that it achieves fairness while still maintaining throughput efficiency and friendliness with respect to TCP New Reno. Another important property from the implementation standpoint is the "sender-side-only modification introduced by TCP Libra. The fairness and stability properties are proved analytically and are extensively tested via simulation. A comparison is also carried out with other TCP versions that have been reported as RTT fair in the literature.

I. Introduction

The success of the Internet is based on the ability to provide a reliable medium for information exchange. In the current Internet, traffic control functionalities are provided by the Transmission Control Protocol (TCP) in an end-to-end fashion. TCP was initially designed to provide a connection-oriented reliable service in the ARPANET [1], and later, in the Internet. TCP addresses two major issues: reliability and congestion control [2]. To achieve the second goal, TCP adapts the sending rate to avoid network overflow or falling into service starvation. TCP congestion control has been studied by the research community for the last 25 years, leading to several TCP variants for congestion control with and without the explicit intervention of the network layer (a survey can be found in [3]).

This work is partially supported by the Italian Ministry for Research via the ICTP/E-Grid Initiative and the Interlink Initiative, the National Science Foundation through grants CNS-0435515/ANI-0221528, and the UC-Micro Grant MICRO 04-05 private sponsor STMicroelectronics. Libra is the Latin word for "scale". This paper has been also submitted for publication to the IEEE JSAC —Special Issue on Non Linear Optimization. Reference Author: Giovanni Pau, Computer Science Department, University of California Los Angeles, CA 90095, e-mail:gpau@cs.ucla.edu

[1]With TCP, unless specified, we refer to TCP New Reno.

The most popular version, TCP New Reno, implements a congestion control algorithm, known as the AIMD (Additive Increase, Multiplicative Decrease) algorithm. The very basic concept 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, it increases its sending window by one.

Naturally, the TCP New Reno implementation goes well beyond the basic AIMD principle. To begin, if the packet loss is due to a timeout, the network is assumed to be severely congested, hence the sending window is drastically reduced to 1 and a new slow-start initiated. A sequence of three duplicate acknowledgments indicates a mild congestion and leads to the Fast Retransmit and Fast Recovery algorithm followed by a congestion window reduction by half. The window increase rate slows down when a window threshold is reached from one unit per ACK received to one unit after an entire RTT. In summary, the data-sending rate of TCP[2] is determined by the rate of incoming acknowledgments (ACKs) to previous packets. At a steady state, the sending rate of a TCP source will match the arrival rate of the related ACKs. The congestion control mechanisms triggered by a segment loss guarantees that the protocol automatically detects congestion buildup and thus decreases its sending rate. This behavior has been referred to as TCPs "self-clocking behavior". Unfortunately this self-regulation procedure may be unfair. Experimentally It has been observed that competing TCP senders, with different end-to-end propagation delays, will typically have different rates of feedback from their peers and will increase their sending window at a different pace. The different increase rates of the sending windows (the faster, the shorter the RTT between sender and receiver) determine the RTT-bias or unfairness of TCP New Reno. This phenomena can also be analytically derived from the formula that models the TCP rate by Kurose in [4]. In particular, it may be seen that competing flows will share a bottleneck link based on the inverse of their round-trip times. The RTT-bias has been recognized from early protocol deployment. In [5], the TCP New Reno authors proposed a solution to the protocol's RTT-unfairness; unfortunately this has been found unstable by Henderson stability studies in [6]. The RTT-bias shown by TCP negatively affects users, content providers, and network providers. In particular, users with a larger RTT will experience a lower throughput and higher latency and completion time. To overcome this imbalance, content providers are required to use (for a price) a content delivery network with a fine-grain geographic distribution, where content is downloaded off-line to sites relatively close to users. The RTT-unfairness may also affect decisions on cache location, CDN deployment, overlay networks topology, and capacity expansion [7]. In this paper we study the RTT-bias in TCP, approaching the problem from a non-linear optimization and a system dynamics point of view. In particular we leverage the seminal work on TCP modeling done by Kelly, Vinniecombe, Low, Paganini and Doyle. Our contribution lies in the design of a new utility function for TCP that doesn't include RTT and therefore is not sensitive to the self-clocking behavior. This results in an enhanced congestion control algorithm for TCP. Our scheme, namely *TCP Libra* (where Libra in Latin means "scale," i.e., balance, fairness), shows RTT-fairness as well as scalability (related to network bottleneck size), and good friendliness to TCP New Reno.

The reminder of the paper is organized as follows. In Section II we introduce the current state of the art, while in Section III we present our solution to the RTT-bias along with the dynamic and stability analysis for TCP Libra, as well as several competing protocols. The simulation experiments are presented in section IV and the paper is finally concluded by sections V.

II. Related Work

In recent years a large amount of research has been conducted to understand TCP behavior under different scenarios and network conditions. Particularly remarkable are the efforts to provide a well-defined mathematical model suitable to study the behavior and stability of TCP and active queuing management (AQM) techniques in the Internet. In this section we will report the leading theoretical work on congestion control and AQM, focusing mainly on RTT-fairness and stability.

The RTT-bias, as well as the bursty nature of TCP traffic, was first experimentally discovered by Sally Floyd and Van Jacobson in [5]. They also proposed a solution for the RTT-bias based on a constant increase algorithm

[2]We use as equivalents the terms congestion window, sending window, and data-sending rate.

for the TCP window at steady state. In a further study they also introduced a network layer technique, namely RED, designed as a relief from the effects of traffic bursts [8] [9]. Briefly, RED monitors the queue length at the routers and probabilistically drops a random TCP packet from the queue upon reaching a certain queue length —thus leading to a more regular traffic pattern [10]. Henderson in [11] [6] shows that using constant increase and RED to achieve RTT-fairness leads to an unstable solution for links with long propagation delays and small buffers such as satellite links.
A detailed mathematical model for the TCP throughput at steady, including the the Fast Retransmit–Fast Recovery phases and TCP's timeout impact, was first introduced by Towsley in [4] and [12]. Additionally, in [13] the same authors developed a new fluid based modeling methodology for studying TCP behavior in a network that features AQM routers. In particular, the proposed approach models TCP behavior that includes the transient effects introduced by AQM routers such as RED gateways.[3]

A fresh impulse to congestion control modeling in communication networks has to be acknowledged: Frank Kelly introduced a new mathematical formulation for congestion control in communication networks in terms of non-linear optimization problem (primal/dual). The network is modeled as an interconnection of users/sources which generate data and resources/links. The key constraint is that the network control information can only be passed along the same routes as the data that is being transmitted, and with the same propagation delay as data [14]. In [15] and [16] it was shown that TCP stability can be achieved in the aforementioned network model if the TCP utility function is concave [17] [7]. Kelly's results have been extended by several researchers. In particular, Vinnicombe, Massoulie, Johari and Tan addressed the stability in network congestion control schemes [18]. Vinnicombe showed that delay instability is characterized by the increase rule; while Ott has shown that stochastic instability is primarily influenced by the decrease rule [19] [20] [21]. Additionally Vinnicombe and Massoulie derived the stability condition for scenarios with heterogeneous delays [22] [23]. A further substantial advancement in developing the theory for network congestion control was made by Low, Paganini and Doyle. They fully exploited the *Primal/Dual* modeling approach, finding the conditions needed for a scalable and stable congestion control for the Internet [24] [25] [26] [27] [28] [29]. The theoretical results have been used to drive the design of an enhanced AQM technique namely Random Early Mark (REM) that improves RED performance while reducing the drawbacks, and TCP Fast that contains a congestion control mechanism, based primarily on queuing time, able to guarantee network stability and high utilization in multi-gigabit networks [30], [31] [32] [33].

Focusing our attention on the TCP RTT-bias it is worth noting that there are few recent TCP variations that have tried to address RTT-unfairness. In particular, TCP CUBIC [34], developed by North Carolina University, features a linear RTT-fairnes, while TCP Hybla [35], that enhances the solution proposed by Floyd in [9], provides RTT-fairness under a certain stability bound, and TCP Vegas [36] that provides good RTT-vairness if it is the only TCP in the Network.

In summary, in almost a decade new modeling techniques for network congestion control have been developed which offer a new set of mathematical tools for understanding scalability and stability limits of network algorithms and protocols. In particular, network congestion control has been modeled as a non-linear optimization problem in which the primal models the system as seen from the source/destination point of view, while the dual models the system as seen from the resource point of view. The use of this formulation benefits theoretical studies of the system dynamics from both network and transport layers.

III. Model

In this section we will first describe the network model and present the optimization problem. We will then give a brief review of TCP New Reno and TCP Vegas, and introduce and analyze the TCP Hybla solution. We will finish considering other solutions by adding some comments on TCP CUBIC. Finally, we will show the steps in the design choices of TCP Libra and include a stability analysis, using a control theoretic approach.

[3]The fluid model presented in [13] shows, as a potential source of instability, the RED adaptive sampling and averaging algorithms. These algorithm, indeed, are dependent on packet size and arrival rates.

A. Network Model and Optimization Problem

The network is modeled as a finite set of nodes N and links L, which connect the nodes in N. Each link is characterized by a finite capacity. We define $\underline{c}$ as the vector of 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. We may define the routing matrix $\underline{\underline{R}}$ as:

$$R_{lr} = \begin{cases} 1,\ if\ link\ l \in L\ is\ utilized\ by\ source\ r \in S \\ 0,\ otherwise \end{cases} \tag{1}$$

Each source $r \in S$ is characterized by $x_r(t)$, the transmission rate. The $aggregateflow$ at link l is defined as the sum of the contributions of the sources that use that link:

$$y_l(t) = \sum_r R_{lr} x_r(t - \tau_{lr}^f) \tag{2}$$

where τ_{lr}^f 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. The price is sent back to source as a feedback signal by the link when congestion is detected. The $price(s)$ to which an algorithm is sensitive shapes its dynamics: TCP New Reno is, for example, sensitive to packet loss, while TCP Vegas is sensitive to queueing delay. The aggregate price seen by source r is:

$$q_r(t) = \sum_l R_{lr} p_l(t - \tau_{lr}^b) \tag{3}$$

where τ_{lr}^b 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. The price, also known as $barrier$ [37] is a function of the link flow. We define $f_l(y)$ as the $price$ for sending traffic at rate $y_l = \sum_{r:l \in r} x_r$ on link l.

Let's 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 $true$ advantage for a user when sending at a certain rate. The function that describes this advantage, $U_r(x_r)$, is defined in economics as a $Utility$ function. The utility function of a congestion control scheme is very important, since it determines the equilibrium properties of the scheme, 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:

$$max_{\underline{x}} V(\underline{x}) \tag{4}$$

subject to:

$$\begin{cases} \underline{\underline{R}}\ \underline{x} \leq\ \underline{c} \\ x_r \geq 0,\ \forall r \in S, \end{cases} \tag{5}$$

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 cost incurred by resource l when it pushes flow y through the network. Thus, $V(\underline{x})$ is the net gain, i.e. net utility of sources $\underline{x}$, which must be maximized.

Theorem 3.1: Under the assumptions [38] [37]:

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 $\{x_r(0) \geq 0\}$, the congestion control algorithm:

$$\dot{x_r} = k_r(x_r)(U_r'(x_r) - q_r(t)) \tag{6}$$

(where q_r is the aggregate price function defined earlier and $k_r(x)$ is any non-decreasing, continuous function such that $k_r(x) > 0$, $\forall x_r > 0$) will converge to the unique solution of (4) (5), $\underline{x}(t) \to \hat{\underline{x}}$ as $t \to \infty$.

The left-hand side of (6) represents the component r of the gradient of $V(\underline{x})$, 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 an amplificator of the gradient. The convergence point does not

change. The reason for this term will become clearer later when the utility function will be used to interpret various existing heuristic flow control schemes.

The derivative $\dot{x}_r$ on the right-hand side reflects the fact that in the gradient optimization method, the incremental change of x_r over a unit time should be proportional to the value of the gradient.

The detailed proof of convergence is found in [40]. The careful reader, however, will notice that the function with the derivative shown on the right hand side of (6) 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 $V(\underline{x})$.

This theorem is therefore telling us that in the absence of feedback delay, any congestion control algorithm that can be mapped into a concave utility function, attains global asymptotic stability. In the following sections we will show how the protocols that we use for comparison fit into this framework and how they implement different versions of this algorithm.

B. TCP New Reno

TCP New Reno implements a very simple and effective algorithm. This algorithm is able to scale various orders of magnitude in terms of network dimensions and speed. The algorithm may be briefly summarized as follows:

- On receiving an ACK: $window_{n+1} = window_n + \frac{1}{window_n}$.
- On an indication of mild congestion (3 DUPACK): $window_{n+1} = \frac{1}{2} window_n$.

This may be understood as an attempt to probe the network trying to sense which sending rate may be achieved at the cost of packet loss when the probing fails [39]. Lets now consider the fluid model for congestion control of TCP New Reno. From now on we will follow the notation:

- The r subscript means we are considering the r-th 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.
- $\tilde{T}_r$ is the average round trip time.
- $\lambda_r(t)$ 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 $1/2$.

With the AIMD algorithm, TCP New Reno increments the window by $1/w_r(t)$ per each received ACK, hence the window increases instantaneously as $\frac{x_r(t)}{w_r(t)}(1 - \lambda_r(t))$. Similarly, per each 3 DUPACK, 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 $x_r(t)\lambda_r(t)w_r(t)/2$. We now may write the fluid model for congestion control for an AIMD-like congestion control scheme (of which TCP New Reno is an example):[4]

$$\dot{w_r}(t) = a_r \frac{1 - \lambda_r(t)}{T_r(t)} - b_r x_r(t)\lambda_r(t)w_r(t) \tag{7}$$

By setting $T_r(t) = \tilde{T_r}$, $w_r(t) = x_r(t)\tilde{T_r}$, we have:

$$\dot{x}_r(t) = a_r \frac{1 - \lambda_r(t)}{\tilde{T_r}^2} - b_r x_r(t)\lambda_r(t)x_r(t) \tag{8}$$

This may be re-written in a more general form, taking in account feedback delays, as:

$$\dot{x}_r(t) = \frac{x_r(t - \tilde{T_r})}{\tilde{T_r}}(a_r \frac{1 - \lambda_r(t)}{x_r(t)\tilde{T_r}} - b_r \lambda_r(t)x_r(t)\tilde{T_r}) \tag{9}$$

We took in account that the update at time t is determined by the flow that left the source at a time $\tilde{T_r}$ in advance. The increase or decrease in the second parenthesis of the right-hand side of equation (9) is then done in terms of the rate at time $t - \tilde{T}$, because of feedback delay.

[4]Here and in the rest of the paper, we will ignore the $\frac{4}{3}$ factor that takes in account that the window, $w_r(t)$, oscillates between $\frac{2}{3}w_r(t)$ and $\frac{4}{3}w_r(t)$ with an average $w_r(t)$. We will also avoid substituting the values of a_r and b_r in order to keep a clear understanding of how these parameters influence the dynamics of the algorithm.

TCP New Reno implements an approximate gradient algorithm for the resolution of the congestion control problem. In terms of (9), and considering feedback delay negligible, we may write [38]:

$$\dot{x_r} = a_r(\frac{1-\lambda_r(t)}{T_r^2} - \frac{b_r}{a_r}x_r^2(t)\lambda_r(t)) = a_r((\frac{b_r}{a_r})x_r^2(t) + \frac{1}{T_r^2})(\frac{1}{\frac{b_r}{a_r}T_r^2x_r^2(t)+1} - \lambda_r(t)) \tag{10}$$

Following the structure of eq. (6), the derivative of the net utility function that TCP New Reno implements and the congestion measure TCP New Reno is sensitive to are:

$$U_r'(x_r) - q_t(t) = (\frac{1}{\frac{b_r}{a_r}T_r^2x_r^2(t)+1} - \lambda_r(t)) \tag{11}$$

Integrating in x_r the term $U_r'(x_r)$, we get the utility function as:

$$U(x_r) = \frac{1}{\tilde{T}_r}\sqrt{\frac{a_r}{b_r}}\ tan^{-1}(\sqrt{\frac{b_r}{a_r}}\tilde{T}_r x_r) \tag{12}$$

C. TCP Hybla

The Hybla scheme has been introduced in [35]. This scheme, from a congestion control point of view, is mainly based on the modification to the increase of the AIMD algorithm proposed in [5] and furtherly analysed in [8] [9] [6] [11]. The fluid flow model may be easiliy obtained from (9) by making the following substitution:

$$a_r \leftarrow a_r\frac{\tilde{T}_r^2}{T_0^2} \tag{13}$$

We then have:

$$\frac{dx_r(t)}{dt} = \frac{x_r(t-\tilde{T}_r)}{\tilde{T}_r}(a_r\frac{\tilde{T}_r}{x_r(t)T_0^2}(1-\lambda_r(t)) - b_r x_r(t)\tilde{T}_r\lambda_r(t)) \tag{14}$$

where the new variable, T_0, is a constant reference value. We may interpret T_0 as the reference round-trip time to which Hybla is adjusting the behavior of a flow. More precisely, whatever the round-trip time of a generic flow, the flow will attempt to behave as a flow that is a experiencing a round-trip time of T_0 with the sink. This property becomes clearer by observing the value of the stable point:

$$\tilde{x}_r = \frac{1}{T_0}\sqrt{\frac{a_r}{b_r}\frac{1-\tilde{\lambda}_r}{\tilde{\lambda}_r}} \approx \frac{1}{T_0}\sqrt{\frac{a_r}{b_r}\frac{1}{\tilde{\lambda}_r}} \tag{15}$$

$$\tilde{w}_r^2 \approx \frac{a_r}{b_r}\frac{\tilde{T}_r^2}{\tilde{\lambda}_r T_0^2} \tag{16}$$

By comparing (15) with (35), it is clear that TCP Hybla achieves the same average throughput as a New Reno flow with average RTT T_0. From equation (16) we expect that in the future Hybla will suffer from the same scalability problem known per New Reno. The fluid model is showing that the probability of loss necessary to support a large window size may become exceedingly small, so that the equilibrium window value expressed in (16) may be difficult to achieve.

Here we recall the sufficient condition for the local asymptotic stability of a network with hosts with heterogenous delays operating an AIMD-like congestion control scheme [22] [19] and apply it to Hybla:

$$\frac{a_r}{\tilde{x}_r\tilde{T}_r} = \frac{\tilde{T}_r}{\tilde{x}_r T_0^2} < \frac{\pi}{2\beta} \tag{17}$$

Observing (17) and referring to Kelly [7], one concludes that for Hybla, *we expect there may be delay instability on routes where the ratio T_r/x_r is large and approaches the threshold. Conversely, the convergence will be slow on routes where the ratio is small.*

We also derived the expression of the approximate gradient projection algorithms that Hybla implements:

$$\dot{x_r} = a_r(\frac{1-\lambda_r(t)}{T_0^2} - \frac{b_r}{a_r}x_r^2(t)\lambda_r(t)) = a_r(\frac{1}{T_0^2} + \frac{b_r}{a_r}x_r^2(t))(\frac{1}{1+\frac{b_r}{a_r}T_0^2x_r^2(t)} - \lambda_r(t)) \tag{18}$$

The utility function that TCP Hybla optimizes is:

$$U(x_r) = \frac{1}{T_0}\sqrt{\frac{a_r}{b_r}}\ tan^{-1}(\sqrt{\frac{b_r}{a_r}}T_0 x_r) \tag{19}$$

We note the similarity with TCP New Reno, where $\tilde{T}_r$ is replaced with constant T_0. Thus, the utility function does not depend on round-trip time.

D. TCP Vegas

Here we follow the simple model described in [40]. We here assume that buffer size is large enough so that equilibrium queue length is smaller than the buffer and that there is no packet loss in equilibrium. In this situation, we may assume that the only *price* that is sensed for congestion is queueing time. As stated in [24], *a source monitors its difference between its expected rate and its actual rate, and increments or decrements its window by one in the next round-trip time according to whether the difference is less or greater than a parameter* α_r. Source r sets its window according to the following rule:

$$w_r(t+1) = \begin{cases} w_r(t) + \frac{1}{T_r(t)} & if\ \frac{w_r(t)}{T_r^*} - \frac{w_r(t)}{T_r(t)} < \alpha_r \\ w_r(t) - \frac{1}{T_r(t)} & if\ \frac{w_r(t)}{T_r^*} - \frac{w_r(t)}{T_r(t)} > \alpha_r \\ w_r(t) & else \end{cases} \tag{20}$$

where T_r^* is the propagation delay experienced by the r-th flow. We can then observe that:

$$\begin{aligned} \frac{w_r(t)}{T_r^*} - \frac{w_r(t)}{T_r(t)} &= w_r(t)(\frac{1}{T_r^*} - \frac{1}{T_r(t)}) && (21)\\ &= \frac{w_r(t)}{T_r(t)}(\frac{T_r(t) - T_r^*}{T_r^*}) && (22)\\ &= x_r(t)(\frac{T_r(t) - T_r^*}{T_r^*}) && (23)\\ &= x_r(t)\frac{\phi_r(t)}{T_r^*} && (24) \end{aligned}$$

where in (24), $\phi_r(t) = T_r(t) - T_r^*$ is the instantaneous queueing time experienced by the r-th flow. We then see that we can express (20) more simply as:

$$\dot{w}_r(t) = \frac{1}{T_r(t)} sgn(\frac{\alpha_r T_r^*}{\phi_r(t)} - x_r(t)) \tag{25}$$

where $sgn(x) = 1$ if $x > 0$, $sgn(x) = -1$ if $x < 0$ and $sgn(x) = 0$ if $x = 0$. By substituting $w_r(t) = \tilde{T}_r(t)x_r(t)$ and assuming that $T_r(t) \approx \tilde{T}_r$ we see that we can write (25) in terms of throughput as:

$$\begin{aligned} \dot{x}_r(t) &= \frac{1}{\tilde{T}_r^2} sgn(\frac{\alpha_r T_r^*}{\phi_r(t)} - x_r(t)) && (26)\\ &= \frac{1}{\tilde{T}_r^2} sgn(\frac{\alpha_r T_r^*}{x_r(t)} - \phi_r(t)) && (27) \end{aligned}$$

From (27) we may notice that:

1) The step of the Vegas algorithm is given by $\frac{1}{\tilde{T}_r^2}$.
2) Considering that in the Vegas algorithm α_r is a paramater that depends on the inverse of T_r^*, we may then assume $\alpha_r = k/T_r^*$, where k is some constant. The equilibrium sending rate is then given by:

$$\tilde{x}_r = \frac{\alpha_r T_r^*}{\tilde{\phi}_r} = \frac{k}{\tilde{\phi}_r} \tag{28}$$

   The above equation states that the equilibrium rate achieved by the Vegas algorithm is independent from propagation delay; it depends only from average queuing time $\tilde{\phi}_r$.

3) The utility function that TCP Vegas attempts to maximize is:

$$U(x_r) = \alpha_r T_r^* log(x_r) \tag{29}$$

E. TCP CUBIC

TCP CUBIC [34] has been designed to enhance the window control of TCP BIC [41]. The congestion window of TCP CUBIC is determined by the following:

$$W_{cubic} = C(t-K)^3 + W_{max} \tag{30}$$

where t is the elapsed time from the last window reduction, W_{max} is the window size just before the last window reduction, C is a constant and $K = \sqrt[3]{W_{max}\beta/C}$, where β is a constant multiplication decrease factor applied for window reduction at the time of loss event (i.e., the window reduces to βW_{max} at the time of the last reduction). TCP CUBIC claims *linear* RTT fairness in terms of throughput.

Suppose two competing flows, with two different round-trip times T_1 and T_2, share the same path. Let's also suppose both flows sense losses in a synchronized manner [42], and that the initial parameters are the same. We then have that the instantaneous throughputs for the two flows (making the assumption that $T_1(t) \approx \tilde{T}_1$ and $T_2(t) \approx \tilde{T}_2$) may be written as:

$$x_1(t) = \frac{w_1(t)}{\tilde{T}_1} \tag{31}$$

$$x_2(t) = \frac{w_2(t)}{\tilde{T}_2} \tag{32}$$

But since the time elapsed from the last window reduction is the same, we have that $w_1(t) = w_2(t)$. It is easy then to see that $\frac{x_1(t)}{x_2(t)} = \frac{\tilde{T}_2}{\tilde{T}_1}$, from which we may derive the *linear* RTT-fairness property that is claimed by this algorithm.

F. Comparison

We have written all of the congestion control algorithms (with the exception of TCP CUBIC which we propose to study in more detail in the future) that we have so far analysed in the form:

$$\dot{x}_r(t) = k_r(x_r)(U_r'(x_r) - q_r(t)) \tag{33}$$

The quantities that appear in the expression are:

1) $k_r(x_r)$ is the *stepsize* of the algorithm. As we mentioned earlier, this term is an amplification factor that determines the amount by which the algorithm moves toward the solution at each step. This term determines the speed of convergence and the stability of the algorithm. For all the algorithms we have considered here, $k_r(x_r)$ is a continuous, non-decreasing function such that $k_r(x_r) > 0$, $\forall x_r > 0$.
2) $(U_r'(x_r) - q_r(t))$ is the *direction* in which the algorithm is proceeding, searching for a solution (stable point). More in particular, $U_r'(x_r)$ is the marginal utility function and $q_r(t)$ is the congestion measure (i.e., the marginal penalty) to which the algorithm is sensitive to. The marginal (utility-penalty) functions characterizes the location of the equilibrium point of the algorithm (see Table I).

Algorithm	Stable Point	Stepsize	Marginal Utility	Congestion Measure
NewReno	$\frac{1}{\tilde{T}_r}\sqrt{\frac{a_r}{b_r}\frac{1-\tilde{\lambda}_r}{\tilde{\lambda}_r}}$	$(\frac{b_r}{a_r})x_r^2(t) + \frac{1}{\tilde{T}_r^2}$	$\frac{1}{\frac{b_r}{a_r}\tilde{T}_r^2 x_r^2(t)+1}$	loss probability
Hybla	$\frac{1}{T_0}\sqrt{\frac{a_r}{b_r}\frac{1-\tilde{\lambda}_r}{\tilde{\lambda}_r}}$	$a_r(\frac{b_r}{a_r}x_r^2(t) + \frac{1}{T_0^2})$	$\frac{1}{\frac{b_r}{a_r}T_0^2 x_r^2(t)+1}$	loss probability
Vegas	$\frac{\alpha_r T_r^*}{\tilde{\phi}_r}$	$\frac{1}{\tilde{T}_r^2}$	$\frac{\alpha_r T_r^*}{x_r}$	queueing delay

TABLE I

DYNAMIC AND EQUILIBRIUM PROPERTIES

G. Dynamic Analysis

Let us start by recalling the fluid model for the TCP New Reno congestion control scheme:

$$\frac{dx_r(t)}{dt} = \frac{x_r(t-\tilde{T}_r)}{\tilde{T}_r}\left(\frac{a_r}{x_r(t)\tilde{T}_r}(1-\lambda_r(t)) - b_r x_r(t)\tilde{T}_r\lambda_r(t)\right) \tag{34}$$

This is an NL differential equation in the variable $x_r(t)$ and models the rate's behavior of the r-th flow. For steady state equilibrium we let $t \to \infty$. The stable point corresponds to the value of $x_r(t)$ that nullifies the gradient. It is found by setting the right hand side $= 0$.

We then have:

$$\tilde{x}_r = \frac{1}{\tilde{T}_r}\sqrt{\frac{a_r}{b_r}\frac{1-\tilde{\lambda}_r}{\tilde{\lambda}_r}} \tag{35}$$

where $\tilde{x}_r, \tilde{T}_r, \tilde{\lambda}_r$ are the stable points that are reached after the transient phase by the r-th connection in the network.

A dynamic system may be generally expressed as:

$$\begin{cases} \underline{\dot{x}} = f(\underline{x}, \underline{u}) \\ \underline{y} = g(\underline{x}, \underline{u}) \end{cases} \tag{36}$$

where $\underline{x}$ is the vector of state variables, $\underline{u}$ is the vector of inputs to the system and $\underline{y}$ is the vector of outputs. We want to linearize system (34) around its stable point, in order to express it as (for a strictly causal system):

$$\begin{cases} \underline{\dot{x}} = \underline{\underline{A}}\ \underline{x} + \underline{\underline{B}}\ \underline{u} \\ \underline{y} = \underline{\underline{C}}\ \underline{x} \end{cases} \tag{37}$$

In (34) we may identify $x_r(t)$ and $x_r(t-T_r)$ as state variables [44], and $\lambda_r(t)$ as the input to the system. We linearize (34) around its stable point in order to study the behavior of the system in its neighborhood. In equations (38) to (40) we show the results of computing the derivatives of (34) with respect to the state and input variables, and we substitute the value of (35) for $x_r(t)$ and $x_r(t-\tilde{T}_r)$.

$$\left(\frac{df}{dx_r(t)}\right)_{eq} = -2\frac{b_r\tilde{\lambda}_r}{\tilde{T}_r}\sqrt{\frac{a_r}{b_r}\left(\frac{1-\tilde{\lambda}_r}{\tilde{\lambda}_r}\right)} \tag{38}$$

$$\left(\frac{df}{dx_r(t-\tilde{T}_r)}\right)_{eq} = 0 \tag{39}$$

$$\left(\frac{df}{d\lambda_r(t)}\right)_{eq} = -\frac{a_r}{\tilde{T}_r^2\tilde{\lambda}_r} \tag{40}$$

In the linearization we assume that $x_r(t) = \tilde{x}_r + \delta x_r(t)$, $x_r(t-T_r) = \tilde{x}_r + \delta x_r(t-\tilde{T}_r)$, and $\lambda_r(t) = \tilde{\lambda}_r + \delta\lambda_r(t)$ and that these quantities vary slowly around their equilibrium point. We are now able to find $\underline{\underline{A}}$ and $\underline{\underline{B}}$, both $|S|$ x $|S|$ matrices, where $|S|$ is the number of sources, in the linearized system:

$$\underline{\underline{A}} = diag\left\{-2\frac{b_r\tilde{\lambda}_r}{\tilde{T}_r}\sqrt{\frac{a_r}{b_r}\left(\frac{1-\tilde{\lambda}_r}{\tilde{\lambda}_r}\right)}\right\} \tag{41}$$

$$\underline{\underline{B}} = diag\left\{-\frac{a_r}{\tilde{T}_r^2\tilde{\lambda}_r}\right\} \tag{42}$$

The linearized form of (34), for the r-th flow, is then :

$$\delta\dot{x}_r(t) = -2\frac{b_r\tilde{\lambda}_r}{\tilde{T}_r}\sqrt{\frac{a_r}{b_r}\left(\frac{1-\tilde{\lambda}_r}{\tilde{\lambda}_r}\right)}\delta x_r(t) - \frac{a_r}{\tilde{T}_r^2\tilde{\lambda}_r}\delta\lambda_r(t) \tag{43}$$

The Laplace transform of the transfer function results to be (here we have that $\underline{\underline{C}} = \underline{\underline{I}}$, the identity matrix, in (37), since the output $\underline{y}$ is equal to state variable $\underline{x}$):

$$\underline{\underline{G}}(s) = \underline{\underline{C}}(s\underline{\underline{I}} - \underline{\underline{A}})^{-1}\underline{\underline{B}} \tag{44}$$

$$= diag\left\{-\frac{a_r}{\tilde{T}_r^2\tilde{\lambda}_r}\left(\frac{1}{s + 2\frac{b_r\tilde{\lambda}_r}{\tilde{T}_r}\sqrt{\frac{a_r}{b_r}\left(\frac{1-\tilde{\lambda}_r}{\tilde{\lambda}_r}\right)}}\right)\right\} \tag{45}$$

Recall that a_r and b_r vary from implementation to implementation of the AIMD-type control mechanism. Now we choose these parameters in a way that will lead to the desired fairness results. The values $\hat{a}_r$ and $\hat{b}_r$ that we propose are:

$$\hat{a}_r \leftarrow \frac{\alpha_r\tilde{T}_r^2}{T_0 + \tilde{T}_r}a_r \tag{46}$$

$$\hat{b}_r \leftarrow \frac{T_1}{T_0 + \tilde{T}_r}b_r \tag{47}$$

Note that a new coefficient α_r was introduced. Its setting and function will be explained later.

These values lead to the following stable point for the r-th source:

$$\tilde{x}_r = \sqrt{\frac{a_r}{b_r}\frac{\alpha_r}{T_1}\frac{1-\tilde{\lambda}_r}{\tilde{\lambda}_r}} \tag{48}$$

We now have that (41) and (42) become:

$$\underline{\underline{A}} = diag\left\{-\frac{2T_1\tilde{\lambda}_r b_r}{T_0 + \tilde{T}_r}\sqrt{\frac{\alpha_r a_r}{T_1 b_r}\frac{1-\tilde{\lambda}_r}{\tilde{\lambda}_r}}\right\} \tag{49}$$

$$\underline{\underline{B}} = diag\left\{-\frac{\alpha_r a_r}{(\tilde{T}_r + T_0)\tilde{\lambda}_r}\right\} \tag{50}$$

Kelly in [7] derives the variance for a flow in the case of N flows through a single resource (bottleneck), where all flows share the same values of $\hat{a}_r$, $\hat{b}_r$, $\tilde{T}_r$; and assuming that queues are modeled as M/M/1, with a finite buffer β. In our case we have that:

$$var\left\{x_r(t)\right\} = \frac{\hat{b}_r x_r^2}{2N}\left(\frac{1}{2(1-\tilde{\lambda}_r) + \beta} + \frac{N-1}{2(1-\tilde{\lambda}_r)}\right) \tag{51}$$

$$= \frac{T_1 b_r x_r^2}{2N(\tilde{T}_r + T_0)}\left(\frac{1}{2(1-\tilde{\lambda}_r) + \beta} + \frac{N-1}{2(1-\tilde{\lambda}_r)}\right) \tag{52}$$

We can now explain the design choices in (46) and (47) for $\hat{a}_r$ and $\hat{b}_r$:

1) We achieve a stabilized rate independence from round-trip time by having $\hat{a}_r \propto \tilde{T}_r^2$.
2) We achieve a reduced sensitivity to round-trip time in (49), the system matrix (for $T_0 >> \tilde{T}_r$).
3) We do not penalize the convergence speed of the system to the stable point by multiplying per factor α_r.
4) We achieve scalability (this point will be throughly discussed in the next section).
5) We reduce the variance of throughput with respect to TCP New Reno, according to expression (52). The variance of a flow may be controlled by setting the T_0 and T_1 parameters. In particular, by setting a value of $T_0 \geq 1$ and $T_1 \leq 1$, we can diminish the variance by a factor of at least $1/(\tilde{T}_r + 1)$.
6) The variance of a flow scales with $\tilde{x_r}^2$. The coefficient of variation (i.e. the ratio: standard deviation/mean) of $x_r(t)$ does not depend on x_r. This scale invariance property was first identified by Ott in [45].

H. The α Coefficient

In the previous section we introduced the new α coefficient. The design of the α factor was accomplished with the objective of pursuing the following main objectives:

1) Increase convergence speed and achieve scalability for the algorithm.
2) Keep the algorithm behavior stable.[5]

For the above reasons, the α factor has been decoupled in the product of two components say: $\alpha = S * P$, where

1) S = *Scalability* factor.
2) P = *Penalty* or *Damping* factor.

We achieve scalability by adjusting the *scalability* factor to the capacity of the bottleneck link. To compute the latter, we use packet pair techniques that are run in parallel to the algorithm. In particular, we set:

$$S = \ k_1 C_r \tag{53}$$

where k_1 is a constant and C_r is the capacity of the bottleneck link seen by the r-th source.

The *penalty* factor P has been designed in order to make the increase rate of the algorithm adaptive to the amount of congestion in the network. We adopted queueing time as a measure of congestion of the network, the same measure on which TCP Vegas relies, namely:

$$(T_r(t) - T_r^{min}) \tag{54}$$

The role of the *penalty* factor is then to penalize the increase of the window in the presence of congestion and to maintain the window as close to the maximum value as long as possible. We may use different expressions of penalty functions for this purpose. In Tab. II we show a few options, where $T_r(t)$ represents the instantaneous round-trip time, T_r^{max} the maximum round-trip time and T_r^{min} the minimum round-trip time experienced by the r-th connection. All these functions tend to *penalize* the growth of the congestion window when the instant round-trip time gets close to the maximum round-trip time experienced during the connection. In Fig. 1 and 2 we see the

Penalty Functions
$k_2(\frac{T_r^{max}-T_r(t)}{T_r^{max}-T_r^{min}})$
$k_2(\frac{T_r^{max}-T_r(t)}{T_r^{max}-T_r^{min}}(\frac{2T_r^{max}-T_r^{min}-T_r(t)}{T_r^{max}-T_r^{min}}))$
$e^{-k_2\frac{T_r(t)-T_r^{min}}{T_r^{max}-T_r^{min}}}$

TABLE II

MONOTONICALLY DECREASING FUNCTIONS IN TERMS OF QUEUEING DELAY

results of a simulation where we have two flows, with round-trip times of 400 and 10 ms respectively, in a butterfly configuration, competing in a 100Mbps bottleneck. In the simulation related to Fig. 1, we used a *penalty* function, while we omitted it in the simulation related to Fig. 2.

[5]Here we mean stability in terms of the local asymptotic stability. We then expect that in its stable behavior, the protocol will achieve the average throughput expressed in (48).

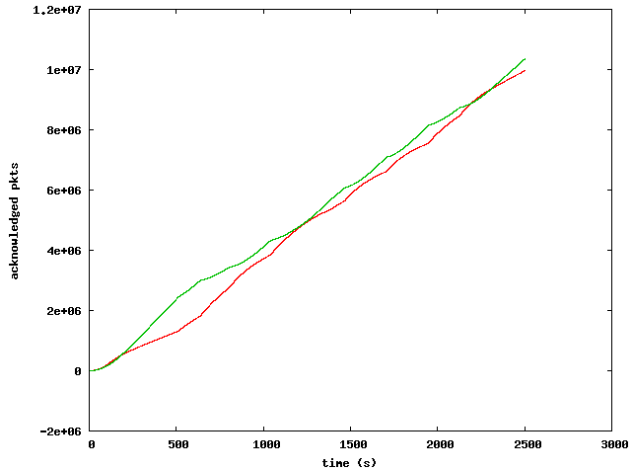

Fig. 1. Throughput of two flows, using a penalty function

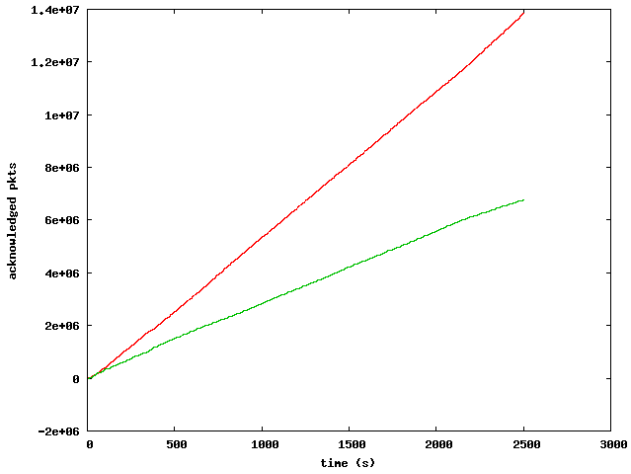

Fig. 2. Throughput of two flows, omitting the penalty function

I. Stability Analysis

Here we will give a simple stability analysis for TCP Libra, which will show the importance of the choice of the α coefficient.

Let's consider a single TCP source on a single link. We will recall the results from the analysis available in [38] for TCP New Reno and compare them with the same analysis for TCP Libra.

We assume that the price is set as in [22] for simplicity:

$$\lambda(x) = (\frac{x(t)}{C})^{\beta} \tag{55}$$

which represents the probability of having a queue of length β or greater for a M/M/1 queueing system; $x(t)$ is the arrival rate and C is the capacity of the link.

We make the assumption that the queue is close to full most of the time. We then have that the round-trip time is equal to the sum of the queueing delay and propagation delay. The sufficient condition for asymptotical local stability is:

$$\kappa \tilde{T} \frac{\tilde{\lambda}'}{\tilde{\lambda}} < \frac{\pi}{2} \tag{56}$$

considering that for a TCP New Reno flow $\kappa = 1/\tilde{T}^2$ we have that:

$$\frac{\tilde{\lambda}'}{\tilde{\lambda}} < \frac{\pi \tilde{T}}{2} \tag{57}$$

Let's now consider a TCP Libra flow. For a TCP Libra flow we have that $\kappa = \frac{\alpha}{T_0 + \tilde{T}}$[6]. The analysis is the same, we can simply substitute this value in (56) and obtain:

$$\frac{\tilde{\lambda}'}{\tilde{\lambda}} < \frac{\pi(\tilde{T} + T_0)}{2\alpha \tilde{T}} \tag{58}$$

from which we are able to find the upper bound $\alpha < \frac{\tilde{T}+T_0}{\tilde{T}^2}$, which gives us the values that α may assume in order to keep a stability region greater than that of New Reno for this simple case. Recalling that $\alpha = Scalability\ Factor * Penalty\ Factor = S * P$, we see that the above bound gives a condition on k_2 once k_1 is fixed, and vice versa. We can find the following expression:

$$k_2 > -\frac{T_{max} - T_{min}}{\tilde{T} - T_{min}} log(\frac{\tilde{T} + T_0}{k_1 C \tilde{T}^2}) \tag{59}$$

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

Once k_1 is fixed and the other parameters and characteristics of the link are known, the above expression establishes the minimum value of k_2 in order to guarantee a stability region at least as wide as New Reno's.

The above is clearly a qualitative analysis, since we have considered the simple case of a single link and a single flow, but it gives us a feeling of how the choice of k_1 and k_2 must be made. As we intuitively expected, (59) is telling us that:

- If k_1 is increased, then k_2 must be also increased in order to keep the same stability bound as New Reno.
- Higher values of C and $\tilde{T}$ require a higher value of k_2. .

J. TCP Libra Algorithm

Finally, we are ready to derive the algorithmic expression of TCP Libra. We will use a synthesis process that is the reverse of what we employed in the analysis of previous TCP protocols. As we did for the algorithms analyzed and presented until now, we will derive the utility function that TCP Libra optimizes (though we will do it before knowing the actual algorithm), and will derive the fluid model algorithm TCP Libra implements, with the modification we designed in (46) and (47).

TCP Libra's fluid flow algorithm is:[7]

$$\frac{dx_r(t)}{dt} = \frac{x_r(t-\tilde{T}_r)}{\tilde{T}_r}\left(\frac{\tilde{\alpha}_r\tilde{T}_r}{x_r(t)(\tilde{T}_r+T_0)}(1-\lambda_r(t)) - \frac{T_1}{2}\frac{x_r(t)\tilde{T}_r\lambda_r(t)}{\tilde{T}_r+T_0}\right) \tag{60}$$

We may write the fluid model algorithm as in (6), by performing the same substitutions in (10):

$$\dot{x} = \left(\frac{T_1}{\tilde{T}_r+T_0}x^2(t) + \frac{\tilde{\alpha}_r}{\tilde{T}_r+T_0}\right)\left(\frac{1}{\frac{T_1}{\tilde{\alpha}_r}x^2(t)+1} - \lambda_r(t)\right) \tag{61}$$

and we can see that the marginal utility function, $U'(x)$, is not dependent on $\tilde{T}_r$. TCP Libra is attempting to minimize the sum of the transfer delays in the network, independently from the round-trip time experienced by the source.

Finally, the TCP Libra algorithm:

- $window_{n+1} \leftarrow window_n + \frac{1}{window_n}\frac{\alpha_n T_n^2}{T_n+T_0}$ in case of a successful transmission.
- $window_{n+1} \leftarrow window_n - \frac{T_1 window_n}{2(T_n+T_0)}$ in case of loss (and the threshold is set accordingly).

In the following section we will show the simulation results for TCP Libra. The simulations have been obtained by substituting $T_0 = 1$, $T_1 = 1$, $k_1 = 2$, $k_2 = 2$ and using the exponential *penalty* function reported in Table II. While a higher value of k_2 would have improved the link utilization by keeping the window at its maximum for a longer time, we have noticed that a higher value k_2 generates an excessively timid behavior of TCP Libra toward TCP New Reno. We adjusted this value as a tradeoff between utilization and friendliness. The parameter k_1 is adjusted accordingly to k_2, as we discussed in the previous section. We finally set T_0 to 1, since in the great majority of cases this will result in $T_r << T_0$ [46], which gives a diminished sensitivity of (49) from T_r, without excessively penalizing the stepsize in (61).

IV. Performance Evaluation

In this section we evaluate TCP Libra using the NS-2 [47] simulation platform. We also compare it to TCP New Reno and with other RTT-fair TCP versions namely TCP Vegas, CUBIC and Hybla. For TCP New Reno and Vegas, we used existing NS-2 modules; for TCP Hybla and TCP CUBIC we used the code provided by the developers; and we developed our own code for TCP Libra. Default parameters had to be changed only in the case of TCP Vegas, as inspired by literature [48].

Each experiment was run for 1000 seconds of simulation time (if not otherwise specified in the relative figure). We chose such a long period because we wanted to evaluate the behavior of the various protocols after they stabilize. Unless stated differently, the network topology was set as in Figure 3. Four FTP connections are established between the source destination pairs Si, and Ri shown in Fig 3. The pairs S1, R1 and S2, R2 have RTT = 40 ms,

[7]Here we will substitute the values for $\hat{a}_r$ and $\hat{b}_r$; we also substitute $a_r = 1$, $b_r = 1/2$ in (46) and (47).

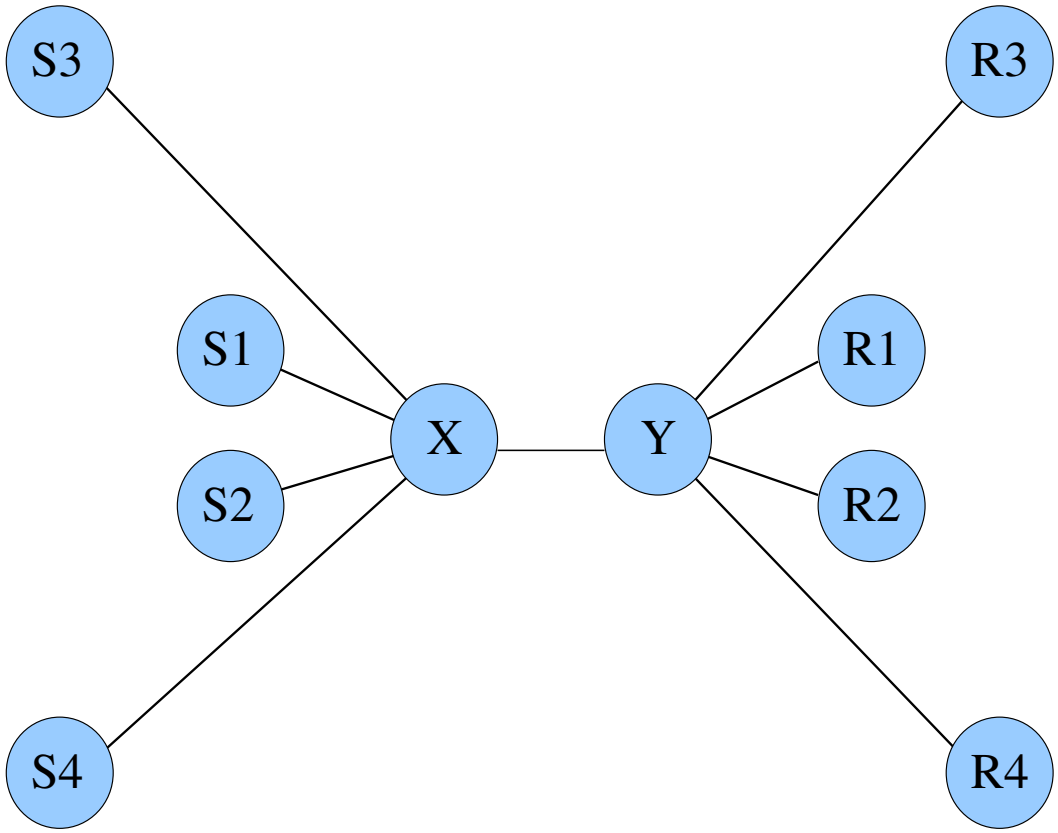

Fig. 3. NS-2 simulation topolgy

representing cross country links. The pairs S3, R3 and S4, R4 have RTT = 160ms, representing intercontinental links. Link capacities are uniform for all links. Link X, Y is thus the common bottleneck. Buffer size at router X (the bottleneck buffer) varies with the experiments. In one series of experiments we set the buffer size in X to the pipe size, i.e., to the number of outstanding packets that can fill the pipe. This is equal to the product of bottleneck capacity by the largest RTT (i.e., 160ms in most of the simulations) divided by packet size. In the remainder of this paper we refer to this value as the longest pipe size. In other experiments, the buffer size was set equal to 200 or 500 packets which are the suggested values for buffers in Cisco Systems routers [49]. We are, in this manner, able to test protocols behavior with buffers sizes that resemble the real routers, as well as with buffer sizes similar to those found in the literature; moreover, we had the opportunity to study TCP Libra stability in relation to the buffer size (see eq.59). Finally, the TCP packet size, including TCP/IP headers, was set to 1500 Bytes. We evaluated the following performance measures:

1) available bandwidth utilization

2) intra-protocol RTT-fairness

3) inter-protocol RTT-fairness

4) friendliness to the legacy protocol TCP New Reno

5) scalability of TCP Libra to many flows, large capacity and large RTT

Is worth noting that in our experiments there are no random errors on links. Thus TCP New Reno behaves the same as TCP Sack. Thus, our friendliness results will hold also for TCP Sack.

A. *Intra-Protocol Fairness*

The Jains Fairness Index [50] is generally adopted in scientific literature to evaluate the RTT-fairness degree among data flows sharing a single bottleneck; therefore we have adopted it even in our analysis. We computed its values to compare the various protocols in various configurations characterized by different bottleneck capacities and buffer sizes. In particular, Figure 4 considers the scenario having the buffer size always equal to the longest pipe size, while Figure 5 corresponds to the case where the buffer size is set 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 bottleneck of 80 Mbps and 160 Mbps, and 200-packet buffer. This result can be traced back to Hybla's stability with large bottleneck capacities and small buffers, and can be derived from the bound in equation (17). The slightly worst behavior of TCP Libra in the experiment with a buffer size of 200 packets may be explained by observing that for smaller buffers the penalty portion in the α factor clearly has a less important influence compared to the larger buffers sizes. This problem may be reduced by increasing the value of k_2 in the functions of table II.

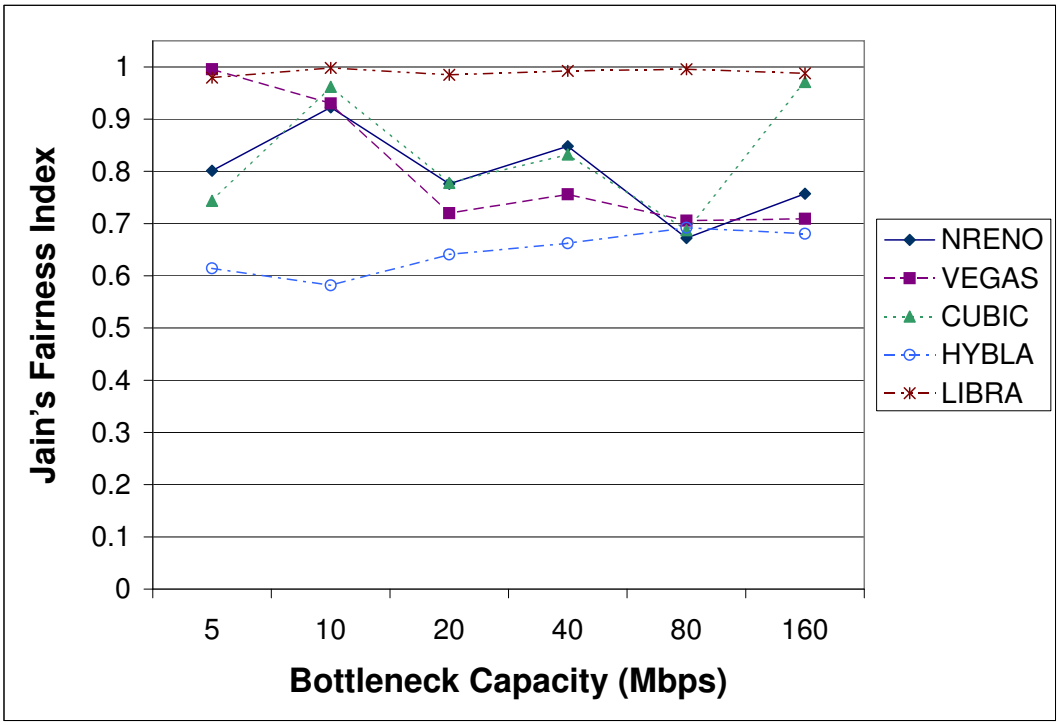

Fig. 4. Jain's Fairness Index vs. bottleneck capacity for TCP New Reno, TCP Vegas, TCP Cubic, TCP Hybla, and TCP Libra. Buffer size at the bottleneck was set equal to the longest pipe size.

B. *Inter-Protcol Fairness (Friendliness)*

With an aim at verifying the compatibility of newly proposed protocols with Internet's de facto standard we need to evaluate scenarios where each of the new protocols competes for the same bottleneck with the traditional TCP

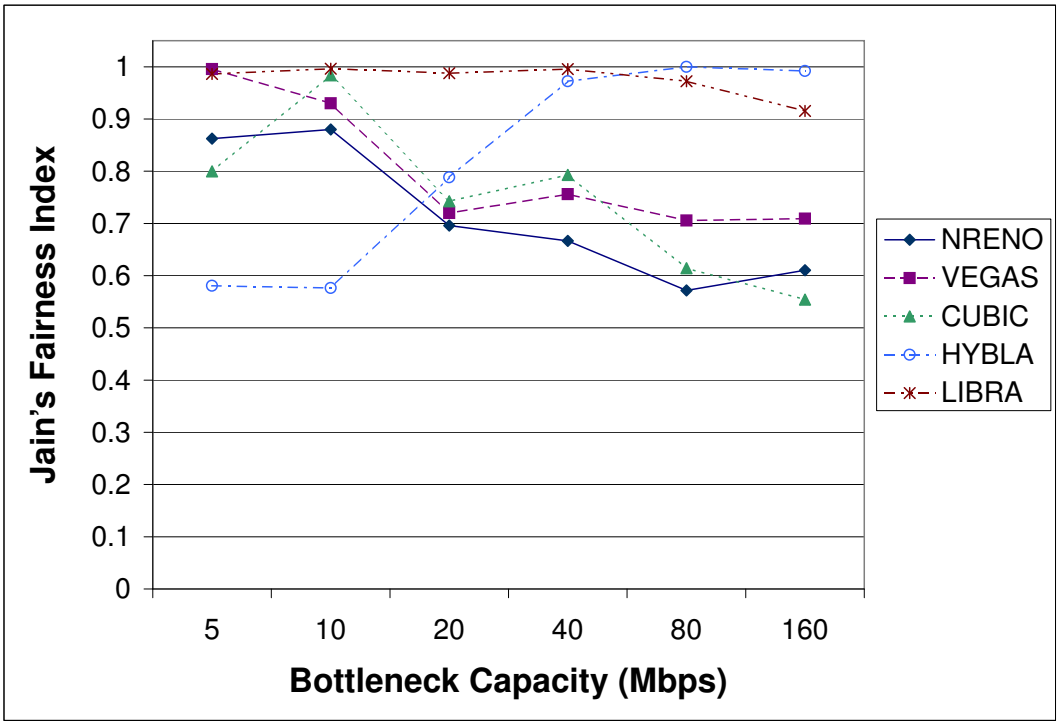

Fig. 5. Jain's Fairness Index vs. bottleneck capacity for TCP New Reno , TCP Vegas, TCP Cubic, TCP Hybla, and TCP Libra. Buffer size at the bottleneck was set equal to *200 packets*, as suggested default value in the CISCO Systems configuration manual(s) [49]

New Reno. We therefore considered various configurations where TCP New Reno was used for one short RTT flow (from S2 to R2) and one long RTT flow (from S4 to R4), while the other two concurrent flows were driven by one of the other TCP flavors. This allowed us to investigate the impact of coexistence on the RTT-fairness degree and the friendliness of the alternative TCP versions towards TCP New Reno. We chose the asymmetry index [51] as friendliness metric. The Asymmetry Index A is defined in equation (62) below.

$$A = \frac{\bar{x}_1 - \bar{x}_2}{\bar{x}_1 + \bar{x}_2} \tag{62}$$

In this formula $\bar{x}_1$ and $\bar{x}_2$ correspond to the average throughputs achieved by two different protocols competing for the same channel. If, as in our case, the two competing protocols are provided with exactly the same initial conditions and each of them is able to exploit (almost) all the available bandwidth when employed alone, then this index can be employed to linearly indicate the degree of aggressiveness of the two protocols toward one another. In essence, when $A = 0$, the two protocols share the bottleneck perfectly evenly. Conversely $A > 0$, correspond to having the first protocol more aggressive than the second, while $A < 0$ imply the inverse situation.

In Fig. 6 and Fig. 7 the asymmetry index is presented for all the bandwidth configurations and, for each of them, TCP New Reno coexists with one of the alternative TCP versions. In particular, in Fig. 6 the buffer size was equal to the longest pipe size (2133 packets), while in Fig. 7 the buffer size was 200 packets.

As can be seen in the charts, when TCP Vegas is used concurrently with TCP New Reno, the latter always presents very high increments of throughput (the Asymmetry Index for TCP Vegas is : $A < 0$). This is because as soon as TCP Vegas detects that the queue is starting to build up, it reduces the transmission rate. The same

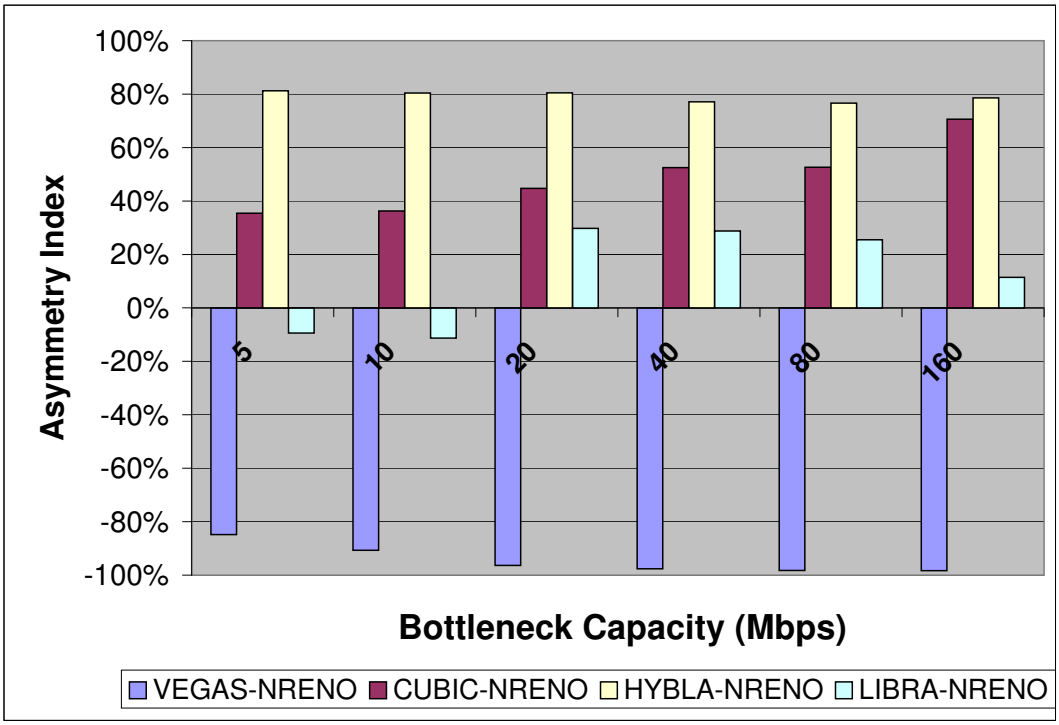

Fig. 6. SLAC Asymmetry Index [51]. TCP New Reno vs. TCP Vegas, TCP CUBIC, TCP Hybla, and TCP Libra. Two TCP New Reno flows compete with two other flows. RTT values 40ms and 160ms respectively. Index $= 0 \longrightarrow$ perfect friendliness; Index $< 0 \longrightarrow$ TCP New Reno is more aggressive; Index $> 0 \longrightarrow$ the compared TCP is more aggressive than TCP New Reno. Buffer size at the bottleneck has been set equal to the *longest link pipe size*.

conservative behavior is not adopted by the concurrent TCP New Reno, which continues to probe the channel fully exploiting available queues and bandwidth along the path and making TCP Vegas sense a continuous congestion in the network. As a consequence, TCP Vegas quickly brings its congestion window to a stable but very low value, thus achieving very low throughput levels.

In opposition, TCP Hybla is predisposed to be too aggressive toward TCP New Reno, thus using most of the bandwidth and sensibly reducing TCP New Reno's throughput when competing for the same channel, as confirmed by Hybla's Asymmetry Index always positive. This is because regardless of the actual current RTT, TCP Hybla tries to make all the TCP connections achieve a transmission rate equivalent to the one that would be attained by a connection experiencing the reference RTT value (i.e., in this experiment, 25 ms as reported in [35]).

TCP CUBIC was not as highly conservative or aggressive toward TCP New Reno as TCP Vegas and TCP Hybla were. However, in all the configurations simulated, even in those that we did not presented here due to space limitation, TCP Libra showed the highest friendliness degree among all the protocols compared in this work. Indeed, TCP Libra is the protocol that had the least impact on the throughput achieved by the concurrent TCP New Reno connections and, in a few configurations, (slightly) reduced the throughput attained by TCP New Reno as confirmed by the Asymmetry Index.

In summary, TCP Vegas is excessively timid, while TCP Hybla and TCP CUBIC are too aggressive. TCP

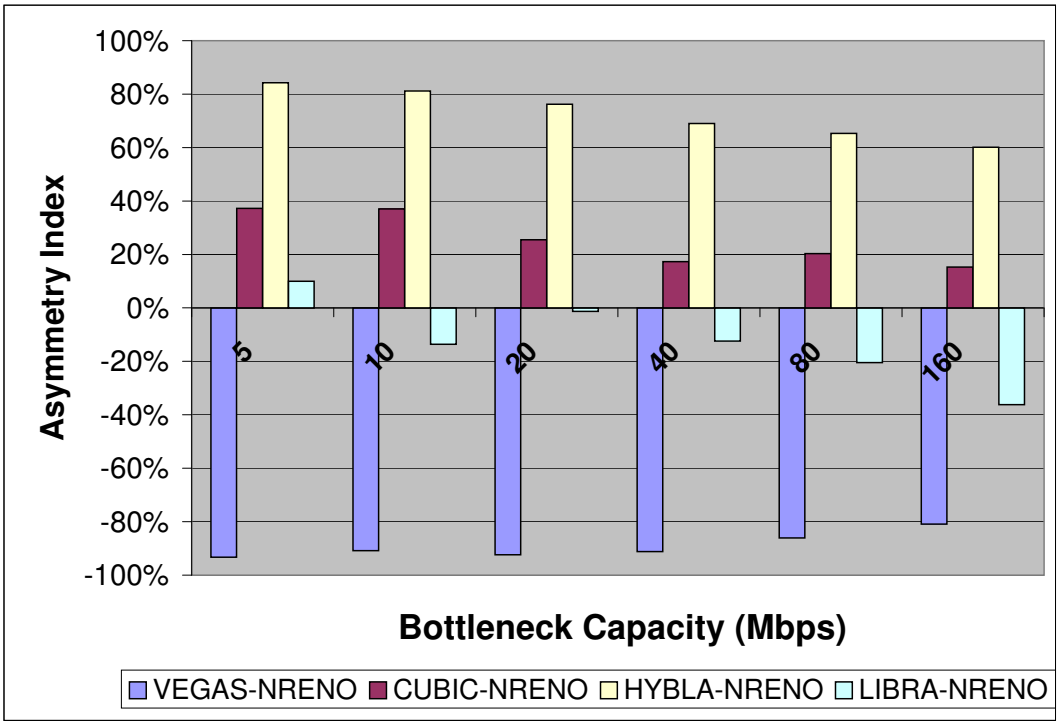

Fig. 7. BIC, TCP Hybla, and TCP Libra. Two TCP New Reno flows compete with two other flows. RTT values 40ms and 160ms respectively. Index $= 0 \longrightarrow$ perfect friendliness; Index $< 0 \longrightarrow$ TCP New Reno is more aggressive; Index $> 0 \longrightarrow$ the compared TCP is more aggressive than TCP New Reno. Buffer size at the bottleneck has been set equal to 200 packets. This is the suggested default value in the CISCO Systems configuration manual(s) [49].

Libra demonstrates that is generally able to fairly share the available bandwidth with TCP New Reno. The only configuration where another protocol (i.e., TCP CUBIC) achieves a better asymmetry index than TCP Libra is when a buffer of 200 packets is employed in combination with a 160Mbps bottleneck link (in this configuration, the actual pipe size would be 2133 packets —an order of magnitude greater).

Finally, not only is TCP Libra RTT-fair when employed as the only transport protocol and friendly to TCP New Reno when coexisting with it, but in this latter case it also maintains the intra-protocol RTT-fairness. Indeed, the simulation results obtained using the aforementioned symmetric topology (see Fig. 3) showed that each of the two TCP Libra connections tends to achieve a throughput which is close to the average throughput attained by the two RTT-unfair connections utilizing TCP New Reno.

C. TCP Libra Scalability Discussion

We studied TCP Libra scalability focusing on three different aspects: (1) scalability in relation to link capacity, (2) scalability in relation to number of flows across a given bottleneck and, (3) scalability in relation to the displacement of RTT(s) among the flows.

1) Link capacity scalability: The current slow start and congestion control mechanism implemented by traditional TCP flavors has proved to have difficulty reaching full utilization on high-speed[8] links, particularly on connections

[8]Here we are targeting core links at gigabit speeds

characterized by large RTT [52]. We have therefore decided to evaluate the scalability of TCP Libra with a 160ms RTT connection over a 1244Mbps link (i.e., OC24). Figure 8 shows the acknowledged packets as a function of time. The derivative is the throughput. One can see that the slope (i.e., throughput) stabilizes very rapidly in TCP Libra. In contrast, TCP NReno is still climbing after 1000 seconds. In fact, in 8 one can verify that TCP Libra hits 96% of channel utilization after 1000 sec. Under the same conditions, TCP New Reno only achieves about 9% (see Figure 9). We tested also configurations with 400Mbps, 500Mbps, and 622Mbps (i.e., OC12) obtaining similar results.

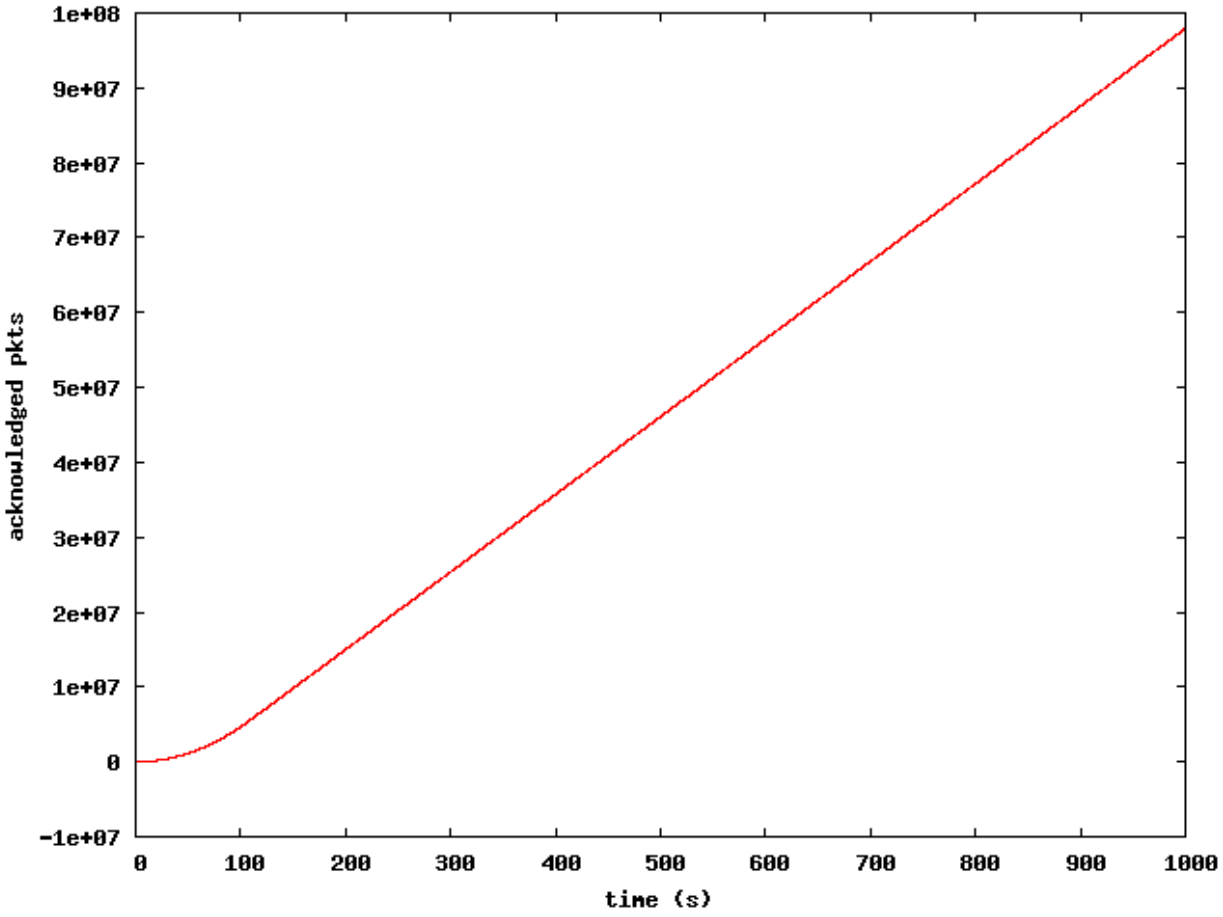

Fig. 8. TCP Libra. Acknowledged packets vs.time. Bottleneck link bandwidth equal to Oc24(i.e., 1.244Gbps), RTT set to 160ms. The TCP Libra link utilization is about = 96%. Buffer size at the bottleneck has been set equal to the longest link pipe size.

2) Number of flows scalability: In this experiment we studied TCP Libra scalability in relation to the number of flows sharing the same bottleneck. In particular, we used the butterfly configuration in Fig. 3 with a bottleneck link of 622 Mbps (i.e., OC12). We designed three experiments involving 100 contemporaneous flows each. The RTT was set as follows:

- 30 flows with RTT 16 ms (i.e., a regional connection)
- 60 flows with RTT 60 ms (i.e., a cross-country connection)
- 20 flows with RTT 180 ms (i.e., transatlantic connection)

First we run TCP New Reno, and then we run TCP Libra using two different settings for the k_2 parameter (see eq. 59). TCP New Reno performance is shown in Fig. 10. As expected from the model, in particular from the utility function in equation (12), TCP New Reno shows RTT-unfairness due to dependency on the RTT during the increase phase. Using the same experimental scenario, we performed a set of experiments directed at studying TCP Libra's stability and friendliness. In particular, Fig.11 shows TCP Libra's performance with $k_2 = 2$, while Fig. 12 shows the results with $k_2 = 12$. It is worth noting the key role played by TCP Libra's parameter k_2. In

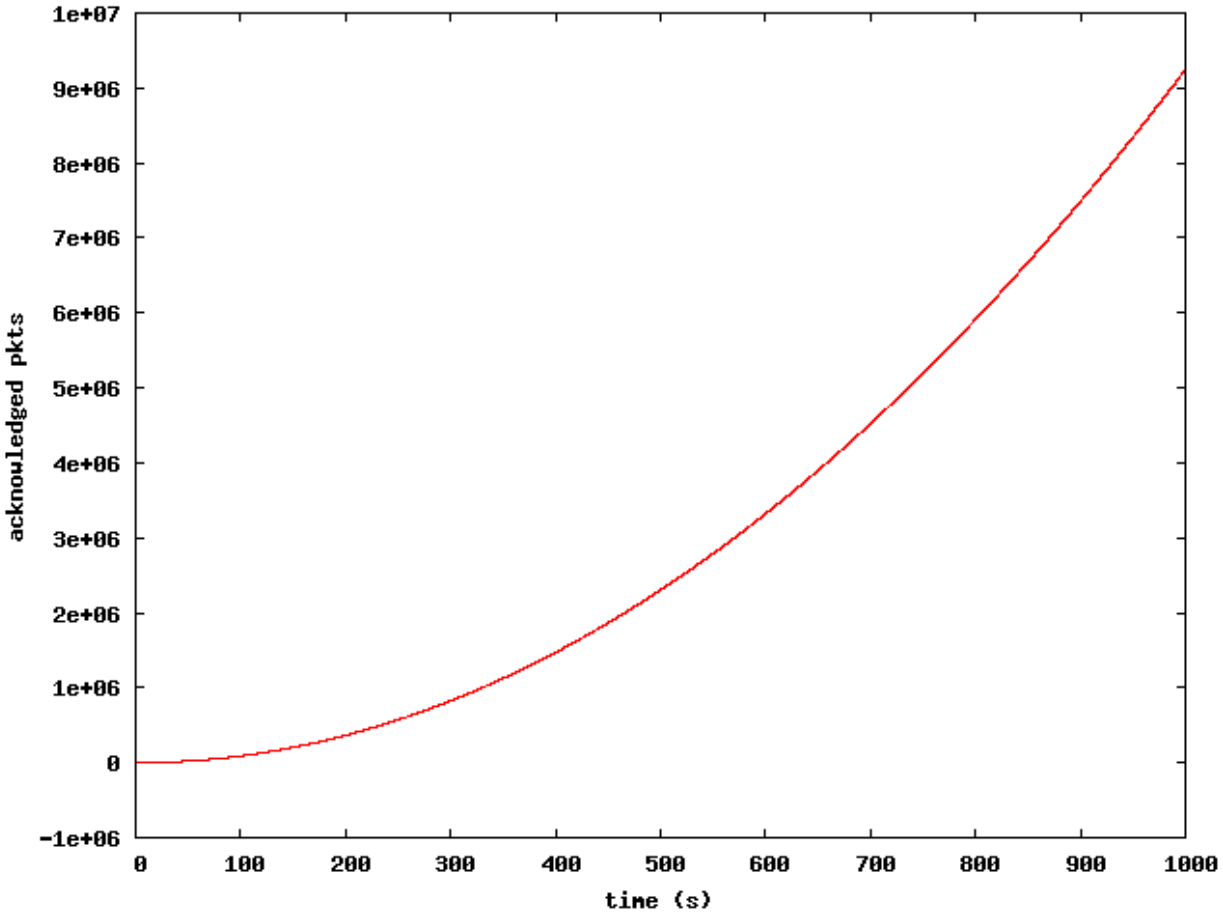

Fig. 9. TCP New Reno. Acknowledged packets vs.time. Bottleneck link bandwidth equal to Oc24(i.e., 1.244Gbps), RTT set to 160ms. The TCP New Reno link utilization is about 9% Buffer size at the bottleneck has been set equal to the bottleneck link pipe size.

particular, the stability bound in equation (59) suggests the following considerations: (a) in this experiment we are on the border of the stability region for TCP Libra and by increasing k_2 we safely return to the stability region and achieve a good RTT-fairness (see Fig. 12); (b) with relatively small buffers (compared to the pipe size) the penalty function has a lower effect thus reducing the RTT-fairness and increasing the friendliness to TCP New Reno; (c) the *cluster* effect in Fig. 12 suggests that in this condition with $k_2 = 2$, TCP Libra still attenuates the RTT-unfairness even though aggregates the flows in regions due to the fact that is on the border of its stability area. In summary, especially with small buffers, a careful selection of TCP Libra's parameters is needed to balance the algorithm's RTT-farness (or effectiveness area) with its friendliness to TCP New Reno as suggested by the TCP Libra's Model.

3) Flows with large RTT spread and staggered start times: In this experiment four different TCP Libra flows (Fig. 13) with RTTs 20, 100, 160, 200, 400 ms are started in sequence at different times. Bottleneck link bandwidth is equal to 100 Mbps. The buffer size at the bottleneck has been set equal to the longest pipe size. Fig. 12 shows the acknowledged packets versus time. It is worth noting that the slopes are the same across the flows. Thus, even in the case of a ratio of 1:20 in RTT, TCP Libra allows all the flows to achieve their fair share. In contrast, Fig. 13 shows a similar experiment with TCP New Reno. One can readily see that the short flow captures the bottleneck.

V. Conclusions and Future Work

This paper introduces TCP Libra, a new protocol designed to be RTT-fair while maintaining a good friendliness with TCP New Reno, thus allowing a seamless deployment in the network. We analytically studied TCP-Libra and its competitors using the primal/dual network modeling formulation and deriving the relative stability bounds. Our study was completed with a simulation analysis performed using NS2 that confirmed TCP Libra characteristics. In particular, we discovered that TCP Libra, when used within its stability region, is able provide RTT-fairness and

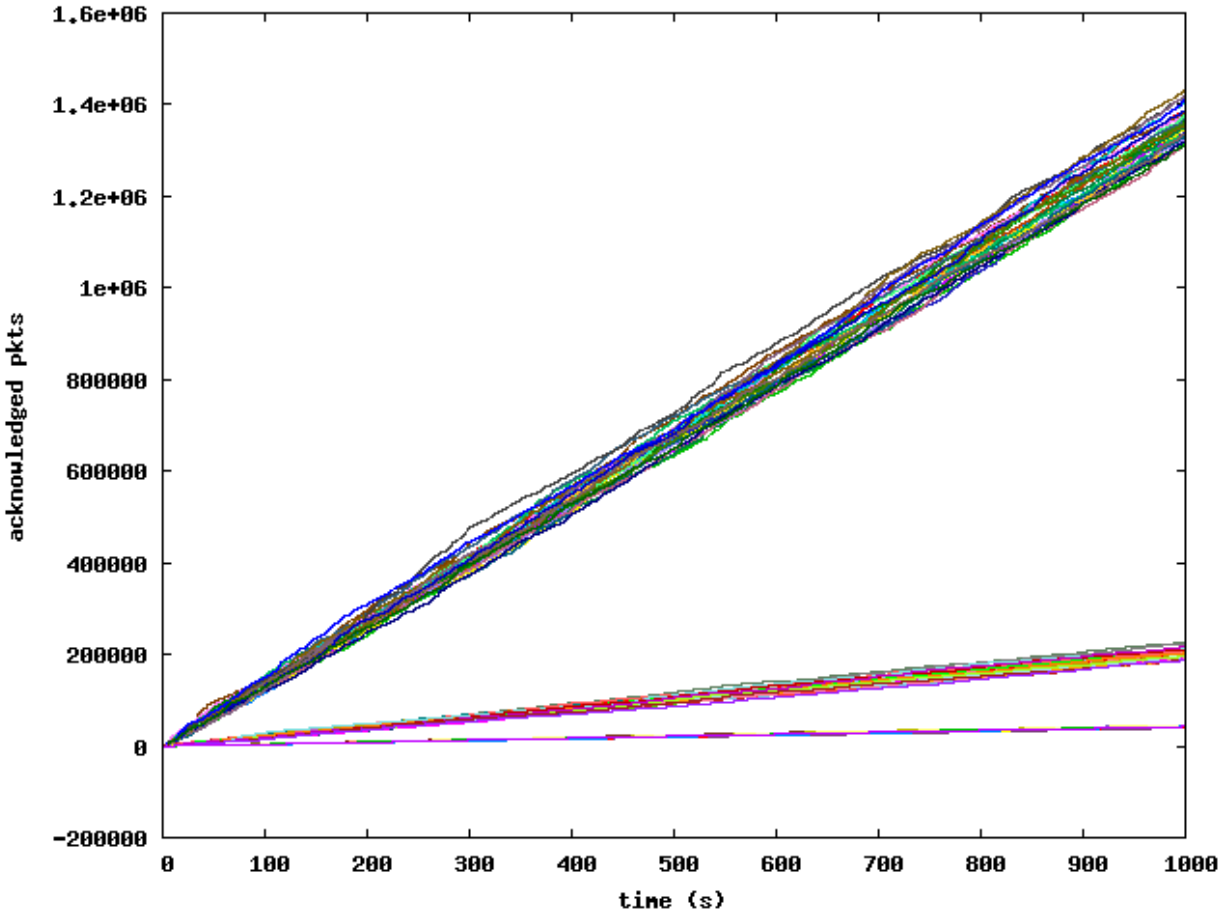

Fig. 10. TCP New Reno. Acknowledged packets vs. time. Bottleneck link of 622 Mbps (Oc12), 100 Flows; 30 Flows with RTT = 16 ms, 60 Flows with RTT = 60 ms, and 20 Flows with RTT = 180 ms. Buffer size at the bottleneck has been set equal to *500 packets*, the suggested default value for high speed core routers in the CISCO Systems configuration manual(s) [49].

maintain good asymmetry index and Jain's index values. TCP Libra requires awareness of the bottleneck capacity and a careful setting of its parameters (i.e., k_1, and k_2) in order to perform at optimum.

In the next several months we will refine TCP Libra and study its performance in a larger number of network scenarios. In particular, we are: (a) developing an efficient capacity estimator, (b) designing a *plug-n-play* parameter optimization technique to make the protocol easy do deploy, (c) implementing TCP Libra's Linux code to perform actual measurements in our high-speed test-bed, (d) developing a more extensive modeling that includes an extensive comparison with other RTT-fair proposals, and (e) studying a slow start enhancement for TCP Libra. Finally we will perform a more extensive set of simulations aimed at studying TCP Libra behavior in heterogeneous scenarios that include mice flows such web those related to web traffic.

NOTES

The NS-2 Implementation of TCP Libra, along with the scripts used in this paper can be found at `http://netlab.cs.ucla.edu/TCP-Libra/`

ACKNOWLEDGMENTS

We wish to thank Rosario Ferrincelli —University of Bologna, for his help with TCP Hybla's analysis and evaluation.

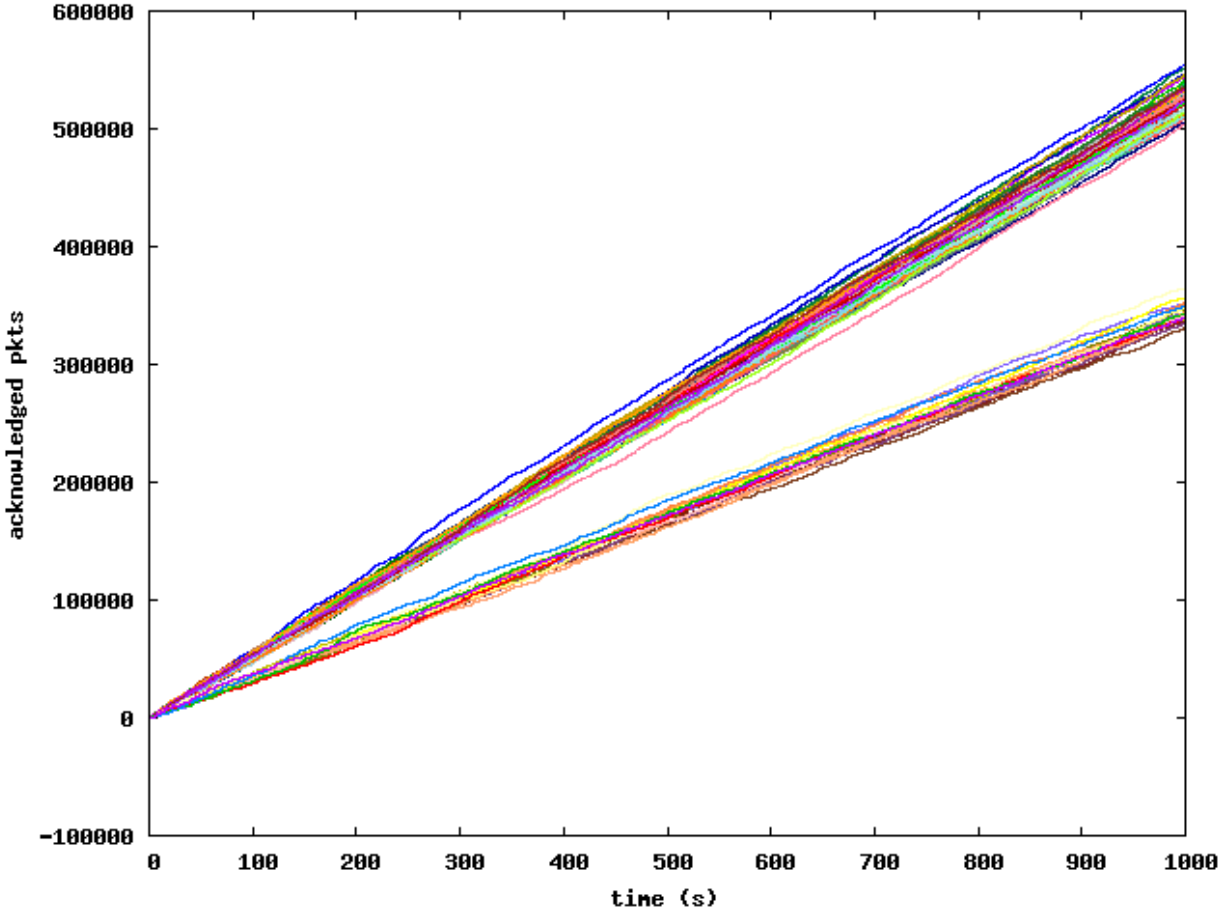

Fig. 11. TCP Libra with $k_2 = 2$. Acknowledged packets vs. time. Bottleneck link of 622 Mbps (Oc12), 100 Flows; 30 Flows with RTT = 16 ms, 60 Flows with RTT = 60 ms, and 20 Flows with RTT = 180 ms. Buffer size at the bottleneck has been set equal to *500 packets*, the suggested default value for high speed core routers in the CISCO Systems configuration manual(s) [49].

REFERENCES

[1] J. Postel, "Rfc0793: Transmission control protocol," Internet Engineering Task Force (IETF), Tech. Rep., 1981. [Online]. Available: tp://ftp.rfc-editor.org/in-notes/std/std7.tx

[2] V. Jacobson, "Congestion avoidance and control," in *Proceedings of ACM SIGCOM '88*. ACM Press, 1988, pp. 314–329.

[3] S. Floyd, "A report on recent developments in tcp congestion control," *IEEE Communications*, vol. 39, no. 4, pp. 84–90, 2001.

[4] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling tcp throughput: A simple model and its empirical validation," in *Proceedings of the ACM SIGCOMM '98 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, 1998, pp. 313–314.

[5] S. Floyd and V. Jacobson, "Traffic phase effects in packet-switched gateways," *ACM SIGCOMM Computer Communication Review*, vol. 21, no. 2, pp. 26–42, 1991.

[6] T. R. Henderson and R. H. Katz, "Transport protocols for internet-compatible satellite networks," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 2, pp. 345–359, 1999.

[7] F. Kelly, "Fairness and stability of end-to-end congestion control," *European Journal of Control*, vol. 9, pp. 159–176, 2003.

[8] S. Floyd, "Connections with multiple congested gateways in packet-switched networks, part 2: Two-way traffic," 1991, unpublished draft.

[9] ——, "A proposed modification to tcp's window increase algorithm. unpublished draft, cited for acknowledgement purposes only, august 1994." August 1994, unpublished Draft.

[10] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance." *IEEE/ACM Transactions on Networking*, vol. 3, no. 3, pp. 115–156, September 1993.

[11] T. Henderson, "Networking over next-generation satellite systems," Ph.D. dissertation, University of California, Berkeley, 1999.

[12] J. Padhye, V. Firoiu, D. F. Towsley, and J. F. Kurose, "Modeling tcp reno performance: A simple model and its empirical validation," *IEEE/ACM Transaction on Networking*, vol. 8, no. 2, pp. 133–145, April 2000.

[13] V. Misra, W.-B. Gong, and D. Towsley, "Fluid-based analysis of a network of aqm routers supporting tcp flows with an application to red," in *SIGCOMM '00: Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. New York, NY, USA: ACM PRESS, 2000, pp. 151–160.

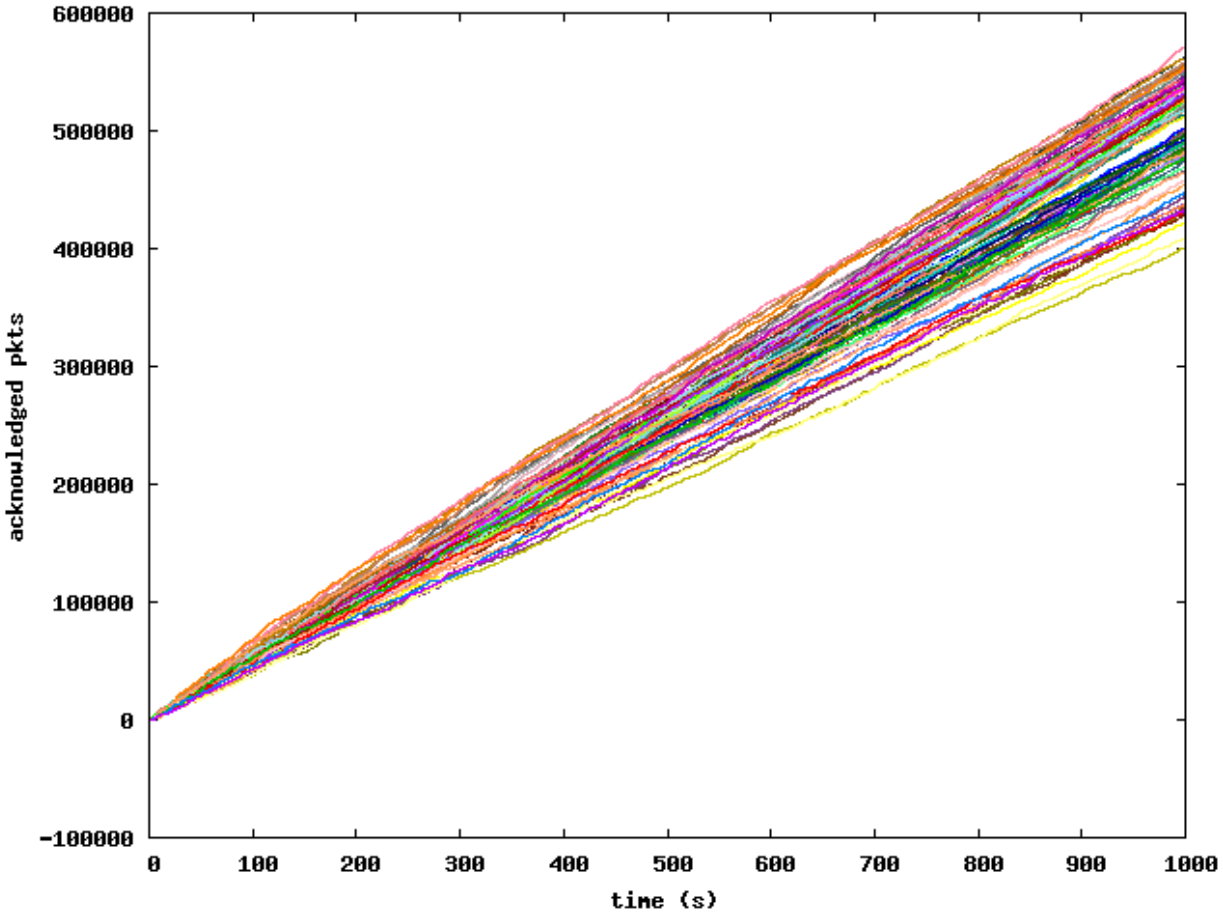

Fig. 12. TCP Libra with $k_2 = 12$. Acknowledged packets vs. time. Bottleneck link of 622 Mbps (Oc12), 100 Flows; 30 Flows with RTT = 16 ms, 60 Flows with RTT = 60 ms, and 20 Flows with RTT = 180 ms. Buffer size at the bottleneck has been set equal to *500 packets*, the suggested default value for high speed core routers in the CISCO Systems configuration manual(s) [49].

[14] F. P. Kelly, A. K. Maulloo, and D. K. H. Tan, "Rate control in communication networks: Shadow prices, proportional fairness, and stability," *Journal of the Operational Research Society*, vol. 49, pp. 237–252, 1998.

[15] F. Kelly, A. Maulloo, and D. Tan, "Rate control in communication networks: Shadow prices, proportional fairness and stability," in *Journal of the Operational Research Society*, vol. 49, 1998. [Online]. Available: citeseer.csail.mit.edu/kelly98rate.html

[16] R. Gibbens and F. Kelly, "Resource pricing and the evolution of congestion control," 1998. [Online]. Available: citeseer.ist.psu.edu/gibbens98resource.html

[17] F. Kelly, "Mathematical modelling of the internet," in *Bjorn Engquist and Wilfried Schmid (Eds.), Mathematics Unlimited – 2001 and Beyond@ Springer*, 2001. [Online]. Available: citeseer.ist.psu.edu/kelly99mathematical.html

[18] R. Johari and D. K. H. Tan, "End-to-end congestion control for the internet: Delays and stability," *IEEE/ACM Trans. Netw.*, vol. 9, no. 6, pp. 818–832, 2001.

[19] G. Vinnicombe, "On the stability of end-to-end congestion control for the internet." Cambridge University Engineering Department, Technical Report CUED/F-INFENG/TR.398. 2000, 2000.

[20] ——, "Robust congestion control for the internet," 2002, submitted to SIGCOM 02.

[21] M. Mathis, J. Semke, and J. Mahdavi, "The macroscopic behavior of the tcp congestion avoidance algorithm," *Computer Communications Review*, vol. 27, no. 3, 1997. [Online]. Available: citeseer.ist.psu.edu/mathis97macroscopic.html

[22] G. Vinnicombe, "On the stability of networks operating tcp-like congestion control," in *Proceedings of IFAC02 World Conference*, 2002.

[23] L. Massoulie, "Stability of distributed congestion control with heterogeneous feedback delays," Microsoft Research, Technical Report MSR-TR-2000-111, 2000.

[24] S. Low, L. Peterson, and L. Wang, "Understanding vegas: A duality model," *Journal of ACM*, vol. 49, no. 207-235, 2002.

[25] F. Paganini, "A global stability result in network flow control," *Systems and Control Letters*, vol. 46, pp. 165–172, 2002.

[26] F. Paganini, Z. Wang, S. Low, and J. C. Doyle. A new tcp/aqm for stable operation in fast networks.

[27] F. Paganini, J. Doyle, and S. Low, "Scalable laws for stable network congestion control," in *Proceedings of IEEE CDC*. Orlando, FL: IEEE, December 2001.

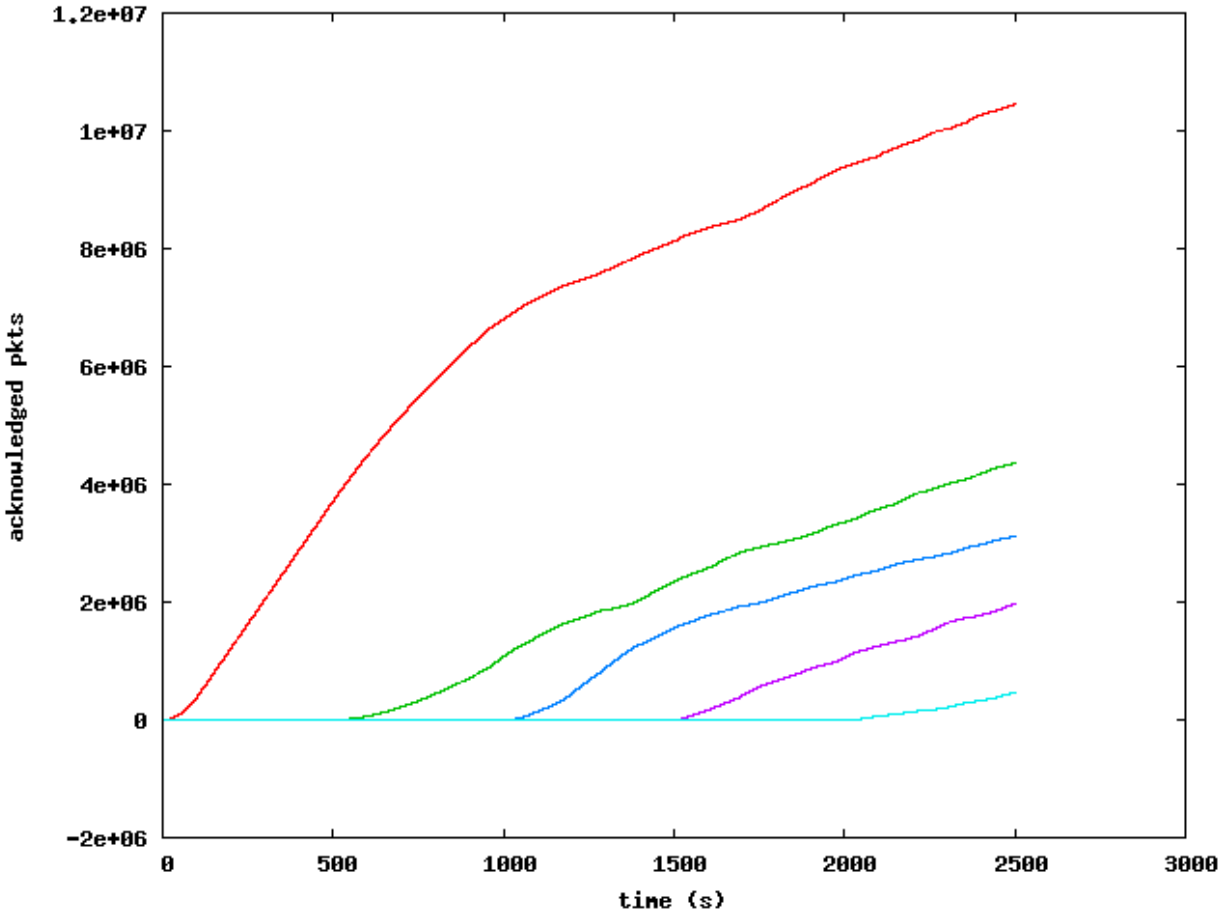

Fig. 13. TCP Libra. Acknowledged packets vs. time. Four different flows with RTTs 20, 100, 160, 200, 400 ms, started in sequence at different times. Bottleneck link bandwidth equal to 100 Mbps. It is worth noting that even in this case TCP Libra allows all the flows to achieve their fair share. Buffer size at the bottleneck has been set equal to the longest pipe size.

[28] S. H. Low and D. E. Lapsley, "Optimization flow control i: Basic algorithm and convergence," *IEEE/ACM Transactions on Networking*, vol. 7, pp. 861–875, 1999.

[29] S. Low, F. Paganini, and J. C. Doyle, "Internet congestion control," *IEEE Control Systems Magazine*, vol. 22, pp. 28–43, 2002.

[30] C. Jin, D. Wei, and S. Low, "Fast tcp: Motivation, architecture, algorithms, performance," in *Proceedings of IEEE INFOCOM*, March 2004. [Online]. Available: citeseer.ist.psu.edu/jin04fast.html

[31] 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, and S. Singh, "Fast tcp: From background theory to experiments," *IEEE Network*, vol. 19, no. 1, pp. 4–11, February 2005. [Online]. Available: citeseer.ist.psu.edu/article/jin03fast.html

[32] S. Athuraliya, D. E. Lapsley, and S. H. Low, "An enhanced random early marking algorithm for internet flow control," in *INFOCOM (3)*, 2000, pp. 1425–1434. [Online]. Available: citeseer.ist.psu.edu/athuraliya99enhanced.html

[33] S. Low and R. Srikant, "A mathematical framework for designing a low-loss, low-delay internet," *Networks and Spatial Economics*, 2003.

[34] I. Rhee and L. Xu, "Cubic: A new tcp-friendly high-speed tcp variant," in *Proceedings of Third International Workshop on Protocols for Fast Long-Distance Networks*, 2005.

[35] C. Caini and R. Ferrincelli, "Tcp hybla: A tcp enhancement for heterogeneous networks," *International Journal of Satellite Communications and Networking*, vol. 22, no. 5, pp. 547–566, 2004.

[36] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "TCP vegas: New techniques for congestion detection and avoidance," in *SIGCOMM*, 1994, pp. 24–35. [Online]. Available: citeseer.ist.psu.edu/article/brakmo94tcp.html

[37] D. P. Bertsekas, *Nonlinear Programming*. Athena Scientific, 1999.

[38] R. Srikant, *The Mathematics of Internet Congestion Control*. A Birkhauser book, 2004.

[39] R. Karp, E. Koutsoupias, C. Papadimitriou, and S. Shenker, "Optimization problems in congestion control," in *41st Annual Symposium on Foundations of Computer Science*, 2000, p. 66.

[40] S. Low, "A duality model of tcp and queue management algorithms," *IEEE/ACM Transactions on Networking*, vol. 11, no. 4, pp. 525–536, August 2003. [Online]. Available: citeseer.csail.mit.edu/article/low00duality.html

[41] L. Xu, K. Harfous, and I. Rhee, "Tcp bic," in *Proceedings of the IEEE INFOCOM, 2004*, 2004.

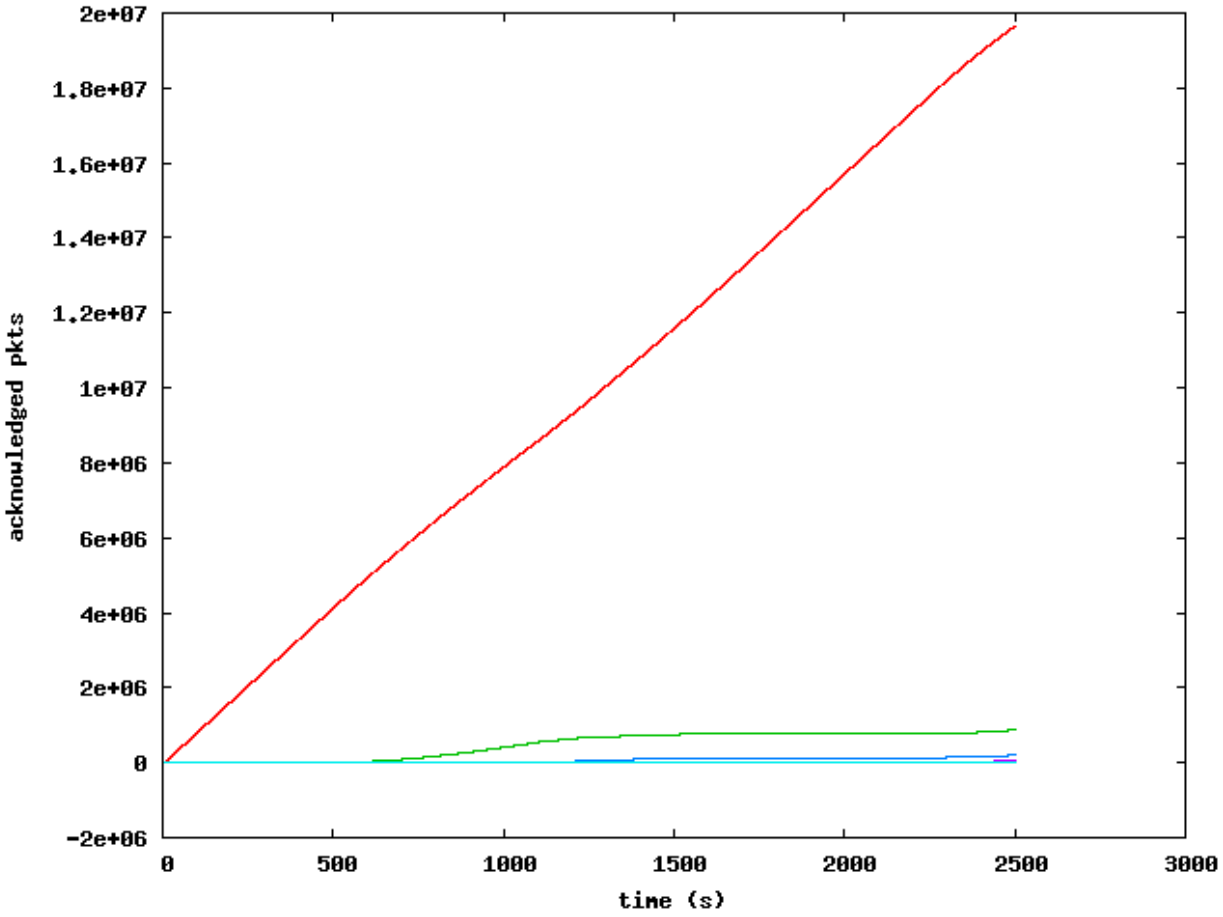

Fig. 14. TCP New Reno. Acknowledged packets vs. time. Four different flows with RTTs 20, 100, 160, 200, 400 ms, started in sequence at different times. Bottleneck link bandwidth equal to 100 Mbps. It is worth noting that even in this case TCP Libra allows all the flows to achieve their fair share. Buffer size at the bottleneck has been set equal to the longest pipe size.

[42] D. Wischik and N. McKeown, "Part i: Buffer sizes for core routers," *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 3, pp. 75–78, 2005.
[43] S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi, and R. Wang, "TCP westwood: Bandwidth estimation for enhanced transport over wireless links," in *Mobile Computing and Networking*, 2001, pp. 287–297. [Online]. Available: citeseer.ist.psu.edu/463862.html
[44] M. Malek-Zavarei and M. Jamshidi, *Time-Delay Systems*. Hardbound, 1987.
[45] M. Mathis, J. Semke, J. Mahdavi, and J. Ott, "The macroscopic behavior of the tcp congestion avoidance algorithm," *SIGCOMM Comput. Commun. Rev.*, vol. 27, no. 3, pp. 67–82, 1997.
[46] J. Aikat, J. Kaur, F. D. Smith, and K. Jeffay, "Variability in tcp round-trip times," in *IMC '03: Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement*. New York, NY, USA: ACM Press, 2003, pp. 279–284.
[47] T. V. Project, *The Network Simulator - ns-2*. [Online]. Available: http://www.isi.edu/nsnam/ns/
[48] S. H. Low, L. L. Peterson, and L. Wang, "Understanding TCP vegas: A duality model," in *SIGMETRICS/Performance*, 2001, pp. 226–235. [Online]. Available: citeseer.csail.mit.edu/article/low01understanding.html
[49] C. S. Inc. Buffer tuning for all cisco routers – document id: 15091. [Online]. Available: http://www.cisco.com/warp/public/63/buffertuning.html
[50] R. Jain, D. Chiu, and W. Hawe, "A quantitative measure of fairness and discrimination for resource allocation in shared computer systems," DEC Research Labs, Technical Report TR-301, 1984.
[51] L. Cottrell, H. Bullot, and 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]. Available: www.slac.stanford.edu/grp/scs/net/talk03/pfld-feb04.ppt
[52] R. Nitzan and B. Tierney, "Experiences with tcp/ip over an atm oc12 wan," Lawrence Berkeley National Laboratory, Technical Report LBNL-44765, 1999.