TCP *Libra*: Exploring RTT-Fairness for TCP

Gustavo Marfia[1], Claudio Palazzi[1,2], Giovanni Pau[1], Mario Gerla[1], M.Y. Sanadidi[1], and Marco Roccetti[2,⋆]

[1] Computer Science Department
University of California Los Angeles, CA 90095
{gmarfia,cpalazzi,gpau,gerla,medy}@cs.ucla.edu
[2] Dipartimento di Scienze dell'Informazione, University of Bologna
Italy 40126
roccetti@cs.unibo.it

Abstract. The majority of Internet users rely on the Transmission Control Protocol (TCP[1]) to download large multimedia files from remote servers (e.g. P2P file sharing). TCP has been advertised as a fair-share protocol. However, when session round-trip-times (RTTs) radically differ from each other, the share (of the bottleneck link) may be anything but fair. This motivates us to explore a new TCP, TCP Libra[2], that guarantees fair sharing regardless of RTT.

The key element of TCP Libra is the unique window adjustment algorithm that provably leads to RTT-independent throughput, yet converging to the fair share. We position TCP Libra in a non-linear optimization framework, proving that it provides fairness (in the sense of minimum potential delay fairness) among TCP flows that share the same bottleneck link. Equally important are the friendliness of Libra towards legacy TCP and the throughput efficiency.

TCP Libra is source only based and thus easy to deploy. Via analytic modeling and simulations we show that TCP Libra achieves fairness while maintaining efficiency and friendliness to TCP New Reno. A comparison with other TCP versions that have been reported as *RTT-fair* in the literature is also carried out.

1 Introduction

TCP was initially designed to provide a connection-oriented reliable service in the ARPANET environment, which later became the Internet. TCP addresses three major issues: reliability, flow control and congestion control [3]. To achieve the third goal,

⋆ This material is based upon work partially supported by the National Science Foundation under Grant No. 0520332; by the Italian Ministry for Research via the ICTP/E-Grid Initiative and the Interlink Initiative, by the Italian Ministero Affari Esteri under the Initiative Laboratorio Congiunto and by the UC-Micro Grant MICRO 04-05 private sponsor STMicroelectronics. Libra is the Latin word for "scale". Reference Author: Giovanni Pau, Computer Science Department, University of California Los Angeles, CA 90095, e-mail:gpau@cs.ucla.edu. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or any of the other funding agencies.

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

[2] *Libra* in Latin means *scale*, thus indicating a balance between the flows.

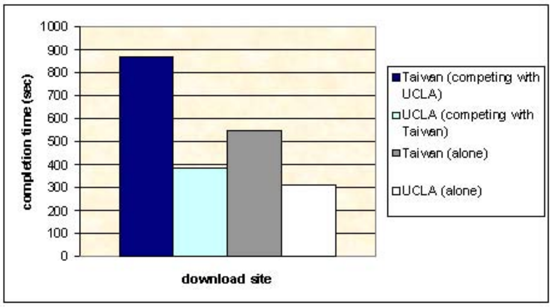

Fig. 1. With TCP New Reno, RTT imbalance typically leads to uneven share of bandwidth between two competing flows. In this experiment one server is at UCLA (with RTT $<$ 1ms) and the other in Taiwan (with RTT = 200ms). Two clients repeatedly download a 26MB file from both servers: both clients are at UCLA. The clients share a 1Mbps wireless LAN Access to the Internet. We show the respective transfer times averaged over more than 100 trials. If clients download from Taiwan and UCLA separately, they complete the downloads in 540 and 305 sec respectively (with equivalent rates .68Mbps and .38Mbps). When the clients share the access link, the UCLA client completes the download in less than 400 sec (equivalent to .52Mbps net rate). The client in Taiwan completes the download in almost 900 sec. Thus, the download from Taiwan achieves the solo rate of .38Mbps only AFTER the UCLA client completes. While the two clients share (during the first 400 sec), the UCLA download achieves .52Mbps, while the Taiwan download achieves only .04Mbps rate! This is a very severe example of unfairness induced by TCP New Reno.

the sending rate is dynamically adjusted to avoid both service starvation and network overflow. The most widely deployed version, TCP New Reno, implements a congestion control algorithm, known as AIMD (Additive Increase, Multiplicative Decrease). The very basic concept can be summarized as follows (for a detailed description, please refer to [2]):

- 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.

The data-sending rate of TCP[3] is determined by the rate of incoming acknowledgments (ACKs). At steady state, it equals the arrival rate of ACKs. This behavior has been referred to as TCP's "self-clocking behavior". This self-regulation procedure, however, may be unfair. Competing TCP senders with different end-to-end propagation delays will receive ACKs at different rates and likewise will increase their sending window at different rates. This property is know as RTT-bias (of TCP New Reno). It was analytically derived in [4]. As a rule of thumb competing TCP throughputs are inversely proportional to round-trip-times. An example of this phenomenon, from a real experiment, can be appreciated in Fig. 1.

[3] We will interchangeably use the terms congestion window, sending window, and data-sending rate.

Because of RTT-bias, competing users with larger RTTs will experience higher file download latency. This problem affects big users (e.g., supercomputer researchers), as well as small users. The RTT induced throughput imbalance is often compensated by content providers by deploying a content delivery network with a fine-grain geographic distribution. Users can then download content from nearby caches instead of the remote server.

TCP congestion control has been extensively studied during the last 30 years, leading to several TCP variants, each providing some added features (along with possible drawbacks). In this context we introduce TCP Libra, a sender-side-only variation to legacy TCP. The goal of TCP Libra is to maintain fairness in the face of uneven RTT values. A similar line of work was followed in [6] and [7], where the authors were inspired by intuition and heuristics. The TCP Libra design was inspired by the analysis of a generalized TCP model. By properly optimizing the model parameters we were able to obtain an algorithm that can be rigorously shown to converge to the RTT fair share. Moreover, the TCP Libra is not only RTT-fair. It also preserves link efficiency and scalability to large link capacities, being friendly to legacy TCP. Another important property of TCP Libra is the sender-side-only modification - non reliant on router active queue management and thus easy to deploy.

We have modeled, simulated and implemented TCP Libra (on Linux 2.6.15). Analytical and simulation results are presented in the paper. The remainder of the paper is organized as follows. The congestion control algorithm is introduced in Section 2, along with the key control parameters/function and with the utility model interpretation. Experimental results are reported in Section 3. Conclusions and future work are in Section 4.

2 The TCP Libra Algorithm

TCP Libra behaves exactly like TCP New Reno except for congestion window management. In fact, Libra differs from TCP New Reno in the following details:

- $window_{n+1} \leftarrow window_n + \frac{1}{window_n}\frac{\alpha_n T_n^2}{T_n+T_0}$ 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 3 DUPACK (and the threshold is set accordingly).

where $window_n$ is the congestion window at step n, α_n a unique control function described in the next section, T_0 and T_1 are fixed parameters and T_n is the RTT at step n. T_1 is the parameter that sets the multiplicative decrease term. T_0 is the parameter that sets the sensitivity of the protocol to the RTT. We can see that the window increase is driven, for $T_n << T_0$ (the typical case), by the α factor and by the square of round-trip-time. In this case, RTT-fairness is enforced (as we will show later) and the algorithm helps large bandwidth-delay-product flows, by letting their windows grow much faster than in TCP New Reno. If instead $T_n >> T_0$ (a rather rare event), the window increase is driven by the α factor and round-trip-time. RTT-fairness is not preserved in this case, but it is weighted as the inverse square root of round-trip-time. This last property of TCP Libra ensures that flows with pathological problems on their paths get a lower sending rate.

2.1 The α Control Function

In the previous section we introduced the new α control function. The design of α 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.[4]

For the above reasons, the α factor is expressed as the product of two components, namely:

$\alpha = S * P$, where

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

We achieve scalability by adjusting the *scalability* factor to the capacity of the narrow link. To compute the latter, we use packet pair techniques that are run embedded into the algorithm. We will further expand this point in the following subsection. In particular, we set:

$$S = \ k_1 C_r \tag{1}$$

where k_1 is a constant and C_r, expressed in Mbps, is the capacity of the narrow link seen by the r-th source.

The *penalty* factor P has been designed in order to adapt the source sending rate increase to the network congestion. As usual, congestion is measured by connection backlog time (i.e. the difference between RTT and minimum RTT). One may use different expressions of penalty functions here. A possible option for P is $e^{-k_2 \frac{T_r(t) - T_r^{min}}{T_r^{max} - T_r^{min}}}$, 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. This function tends to *penalize* the growth of the congestion window when the current round-trip-time approaches to the historic maximum round-trip-time experienced during connection lifetime.

In a later section we will show the simulation results for TCP Libra. The simulations have been obtained by substituting $T_0 = 1$, $T_1 = 1$, $k_1 = 2$ and $k_2 = 2$. While a higher value of k_2, in the penalty function, would have improved the link utilization by keeping the window at its maximum for a longer time, we have noticed that a higher k_2 generates an excessively timid behavior of TCP Libra toward TCP New Reno. We adjusted this value as a trade off between utilization and friendliness. The parameter k_1 is adjusted accordingly to k_2. T_0 is set to 1 (i.e. 1 second), since in the great majority of cases this will result in $T_r << T_0$ [10], which gives a diminished sensitivity to T_r, without excessively penalizing the algorithm's stepsize. T_1 is set to 1, which means that the window decrease will mainly be driven by legacy TCP's decrease rate.

Before moving to comparisons and results, we will discuss in the remainder of this section two other important TCP Libra design components, namely, capacity estimation and the utility function that the Libra algorithm attempts to maximize.

[4] Here we mean stability in terms of the local asymptotic stability.

2.2 Capacity Estimation

TCP Libra relies on CapProbe [11] for capacity estimation. CapProbe is an accurate and fast converging estimation algorithm that may be implemented passively by any TCP scheme. In brief, CapProbe relies on packet pairs and on the concept of minimum delay sum of a packet pair. When a packet pair is sent, if any of the two packets experiences queuing, the sum of RTTs will increase. By monitoring packet pairs and selecting the pair that has the minimum delay sum it is possible, with a high probability to recognize the pair that did not experience any queuing (please refer to [12] for more information regarding this algorithm and on useful ideas on how to incorporate it into a TCP scheme). By isolating one pair that did not experience any queuing and from the knowledge of the interval between the two ACKs for that packet pair, it is possible to compute the capacity of the narrow link of the path. The capacity information is used in eq. (1).

2.3 TCP Libra Fairness: A Utility Function Interpretation

We begin by deriving the TCP Libra fluid flow equation for any one of the several TCP flows sharing the same bottleneck. In the following $w(t)$, $x(t)$, $T(t)$ and $\lambda(t)$ represent, respectively, the instant window size, instant rate, round-trip-time and probability of packet drop. Libra increments the window by $\frac{1}{w(t)}\frac{\alpha(t)T^2(t)}{T(t)+T_0}$ per each successful ACK, hence the window increase per unit time = $\frac{x(t)}{w(t)}\frac{\alpha(t)T^2(t)}{T(t)+T_0}(1-\lambda(t))$.

When a packet is dropped, causing 3 DUPACK the window decreases by $w(t) - \frac{T_1 w(t)}{2(T(t)+T_0)}$. The rate of this event is $x(t)\lambda(t)$. The window decrease per unit time = $x(t)\lambda(t)(w(t) - \frac{T_1 w(t)}{2(T(t)+T_0)})$. We now have all the ingredients to write the fluid model of TCP Libra:

$$\dot{w}(t) = \frac{\alpha(t)T(t)}{T(t)+T_0}(1-\lambda(t)) - x(t)\lambda(t)(w(t) - \frac{T_1 w(t)}{2(T(t)+T_0)}) \tag{2}$$

By setting $T(t) = \tilde{T}$, $w(t) = x(t)\tilde{T}$ and $\alpha(t) = \tilde{\alpha}$, we have:

$$\dot{x}(t) = \frac{\tilde{\alpha}}{\tilde{T}+T_0}(1-\lambda(t)) - x(t)\lambda(t)(x(t) - \frac{T_1 x(t)}{2(\tilde{T}+T_0)}) \tag{3}$$

which (after setting $T_0 = T_1 = 1$ and assuming $T_0 >> \tilde{T}$) may be rewritten as:

$$\dot{x}(t) = \frac{x^2(t)/2+\tilde{\alpha}}{\tilde{T}+1}(\frac{1}{\frac{1}{2\tilde{\alpha}}x^2(t)+1} - \lambda(t)) \tag{4}$$

Eq. (4) is expressed in the form of a gradient algorithm (please refer to [1] [9] for a rigorous treatment of this topic), where $\frac{1}{\frac{1}{\tilde{\alpha}}x^2(t)+1}$ represents the marginal utility function (i.e. the first derivative of the utility function) the algorithm attempts to optimize, $\lambda(t)$ the aggregate price and $\frac{x^2(t)+\tilde{\alpha}}{\tilde{T}+1}$ a non negative, non decreasing function that acts as a gradient step amplifier. By equating marginal utility and price, we nullify the gradient and find the equilibrium solution for rate $x(t)$. The equilibrium point is unique because the derivative of the objective function = marginal utility price (see eq. (4)), decreases with x, and thus the objective function is concave.

$$\tilde{x} = \sqrt{\tilde{\alpha}\frac{1-\tilde{\lambda}}{\tilde{\lambda}}} \tag{5}$$

It will be noticed that the equilibrium TCP Libra throughput $\tilde{x}$ does not depend on RTT under the assumption that $T_0 >> \tilde{T}$. Thus, our design of window increase and decrease algorithm, coupled with the careful setting of parameter T_0, will guarantee fairness among all flows sharing the same bottleneck. Indeed, this fairness property is not a coincidence: we reverse- engineered TCP Libra to make it happen!

3 Performance Evaluation

We evaluate TCP Libra using the NS-2 [13] simulation platform. We compare it to TCP Sack and to other TCP versions that claim RTT-fairness, namely, BIC [8] and Fast TCP [5]. For the latter schemes, we have selected the parameters made available online in simulation scripts by the authors themselves.

Each experiment was run for 1000 seconds in order to reach steady state. Two different topologies are used, each featuring different scenarios. The bottleneck link can take different values. The bottleneck buffer can take two different values: (a) the number of packets that fill the bottleneck link, or; (b) the packets that fill the longest path.

Routers along the path were configured to implement either drop tail or an adaptive RED queuing policy. The advertised window for each connection was set larger than the corresponding pipe size so that occasional packets may be dropped, even when that connection is the only active connection. Finally, the packet size was set equal to 1000 bytes.

Space limitations allow us to present only a subset of the obtained simulation results. We report only 100Mbps bottleneck results since other values did not show significant difference. The Fast TCP parameters *alpha* and *beta* were set to 100, as found in literature. BIC TCP parameters were set as recommended in [8].

3.1 Parking Lot Topology

Fig. 2 shows the so called parking lot topology with 8 end to end flows. Flows 1 and 2 have 180ms of minimum RTT and traverse 9 links; flows 3 and 4 have 90ms of minimum RTT and also traverse 9 links. The remaining flows, 5 through 8, are short flows. They utilize 3 link paths with 30ms minimum RTT. With these values, the bottleneck buffer choices are 375 pkts (bottleneck link pipe) and 2,250 pkts (longest path pipe). To overcome phase effects, flows were started at random times within the first 5 seconds of simulation.

In Fig. 3, for each TCP variant we report the Jain's index values for flows 1-4. TCP Libra provides good fairness with both buffer sizes. The *penalty* factor in Libra adapts the window increase slope to the relative backlog time, thus reducing sensitivity to buffer size.

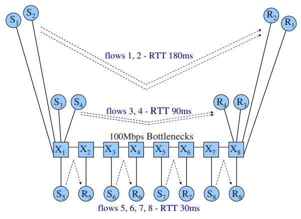

Fig. 2. Parking Lot Topology

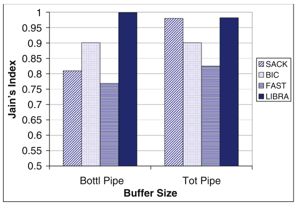

Fig. 3. Jain's index values achieved during the same simulations by flows 1-4, for each different protocol. The buffer is set equal to the bottleneck link pipe size and the longest path pipe size, respectively.

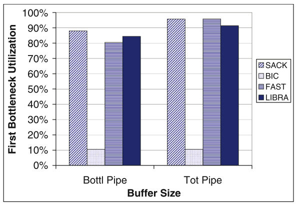

Fig. 4. Efficiency in the Parking Lot Topology

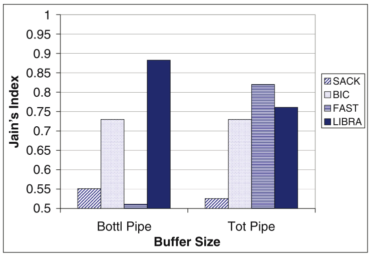

Fig. 5. Jain's index is computed over all the 8 connections. TCP Sack, which had an acceptable Jain's index for connections 1-4 and an optimal one for connections 5-8, obtains a very poor value when considering all the 8 connections together.

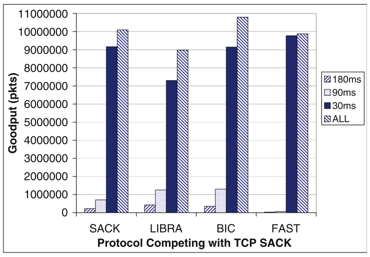

Fig. 6. Friendliness: TCP Sack was utilized on flows 2 (180ms of RTT), 4 (90ms of RTT), 6 and 8 (30 ms of RTT each), while the evaluated alternative protocol was utilized on flows 1 (180ms of RTT), 3 (90ms of RTT), 5 and 7 (30 ms of RTT each) —Parking Lot Topology

Throughput efficiency is considered in Fig. 4. TCP Libra's utilization is slightly lower than that of TCP Sack. As expected, utilization for all protocols increases with buffer size. The particular behavior of short RTT connections 5-8 and longer RTT connections 1-4 is revealed by Fig. 5 where a strong fluctuation of fairness index is noted. Fast TCP performance visibly changes with buffer size. In the small buffer case, Fast TCP's alpha parameter is not optimally tuned and hence a Fast TCP flow tries to keep too many packets in the bottleneck buffer (Note: the alpha parameter in Fast controls the number of packets a flow maintains in the bottleneck link).

3.2 Dumbbell Topology

The coexistence of legacy TCP (i.e. TCP Sack) and the new protocols is evaluated in Fig. 6. The bar chart in Fig. 6 presents the relative TCP Sack throughputs 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 (180ms of RTT), 4 (90ms of RTT), 6 and 8 (30 ms of RTT each), while the new protocol was used for flows 1 (180ms of RTT), 3 (90ms of RTT), 5 and 7 (30 ms of RTT each). Fast TCP's unfriendliness towards TCP Sack is again due to an incorrect value of the alpha parameter. When coexisting with BIC TCP, TCP Sack achieves a slight increase in its achieved goodput. TCP Libra shows a friendly, balanced behavior toward TCP Sack. The aggregate throughput of the coexisting TCP Sack flows diminishes only by 11%. There is a desirable, even if limited, redistribution of the TCP Sack goodput from the 30ms RTT flows to the 90ms and 180ms RTT ones. In this respect, TCP Libra seems to help the coexisting TCP Sack flows by (slightly) improving their fairness degree.

The dumbbell topology is the classic topology used for TCP fairness evaluation. We adapted the simulation scripts utilized by [8]. Each link has a different RTT. The one way propagation delay on the links is 21ms for the short connections (from S_2 to R_2 and between B_i and C_i) and 119ms for the longest one (from S_1 to R_1).

Background traffic flows between B_i to C_i and C_j to B_j. This 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 flows are present.

We here only examine the case in which the bottleneck link is small (i.e. equal to bottleneck link pipe size), the most demanding case for all protocols. Two different queuing policies are tested: drop tail and RED (Random Early Detection). The results are reported in Fig. 8. RED can considerably improve fairness, as expected, since RED was designed to prevent capture by aggressive flows.

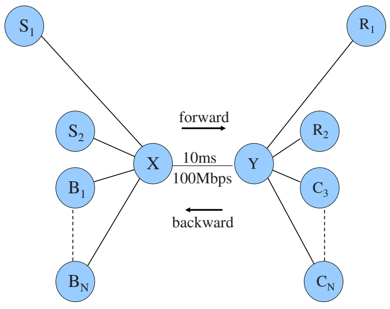

Fig. 7. Dumbbell Topology

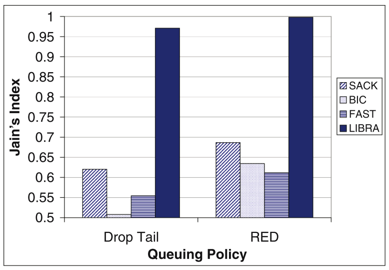

Fig. 8. Jain's index values achieved by the evaluated protocol while competing with a large amount of concurrent traffic. The concurrent connections were both short and long lived TCP sessions in both directions.

4 Conclusions and Future Work

The very interesting properties exhibited by TCP Libra strongly motivate us to continue refine the scheme using the feedback from simulation and early experiments. In particular, we will study TCP Libra performance in a larger number of heterogeneous network scenarios. Following the success of the early lab tests and eager to explore the practical impact on real Internet applications, we are now deploying TCP Libra in controlled Internet testbeds such as PLANET Lab. At the same time, we are preparing to run Internet experiments in increasingly complex and demanding scenarios with several collaborators at other institutions around the world.

References

1. R. Srikant, *The Mathematics of Internet Congestion Control*. A Birkhauser book, 2004.
2. J. Postel, "Rfc0793: Transmission control protocol," Internet Engineering Task Force (IETF), Tech. Rep., 1981.
3. V. Jacobson, "Congestion avoidance and control," in *Proceedings of ACM SIGCOM '88*. ACM Press, 1988.
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*, 1998.
5. C. Jin, D. Wei, and S. Low, "Fast tcp: Motivation, architecture, algorithms, performance," in *Proceedings of IEEE INFOCOM*, 2004.
6. S. Floyd and V. Jacobson, "Traffic phase effects in packet-switched gateways," *ACM SIGCOMM Computer Communication Review*, 1991.
7. T. R. Henderson and R. H. Katz, "Transport protocols for internet-compatible satellite networks," *IEEE Journal on Selected Areas in Communications*, 1999.
8. L. Xu, K. Harfous, and I. Rhee, "Bic tcp," in *Proceedings of the IEEE INFOCOM, 2004*, 2004.
9. D. P. Bertsekas, *Nonlinear Programming*. Athena Scientific, 1999.
10. 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.
11. R. Kapoor, L.-J. Chen, L. Lao, M. Gerla, and M. Y. Sanadidi, "Capprobe: a simple and accurate capacity estimation technique," in *SIGCOMM '04*. New York, NY, USA: ACM Press, 2004.
12. C. Marcondes, A. Persson, L. Chen, M. Y. Sanadidi, and M. Gerla, "Tcp probe: A tcp with built-in path capacity estimation." in *he 8th IEEE Global Internet Symposium (in conjunction with IEEE Infocom.05)*, Miami, USA, 2005.
13. T. V. Project, *The Network Simulator - ns-2*.