TCP Libra: balancing flows over heterogeneous propagation scenarios

–Submission to SigMetrics 2006–

Anonymous due to Double Blind Submission

ABSTRACT

The majority of Internet traffic relies on the Transmission Control Protocol (TCP[1]) devised in the early 1970s to provide a reliable and fair data transfer across the ARPANET. Users that download large multimedia files from remote servers, say, are expected to receive the same share of "bandwidth" (i.e., same transfer rate) if they share the same bottleneck. Unfortunately, with current TCP, fairness does not apply when the round trip delay RTT is very different among sessions. 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 can easily imagine several other applications where RTT-fairness is a must. Here we propose a new version of TCP, namely TCP Libra, which guarantees fair sharing regardless of RTT. "Libra" in Latin means "scale", thus indicating good balance among competing 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 that compensates for RTT difference. 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. This leads to a solution that provides fairness among TCP flows that share the same bottleneck link regardless of RTT. Remarkably, TCP Libra achieves fairness while still maintaining throughput efficiency and friendliness with respect to TCP New Reno. Moreover, TCP Libra is a "sender-side-only" modification of TCP, thus greatly simplifying its deployment. In the paper, fairness and stability properties are proved analytically and are extensively tested via simulation using popular benchmarks. The sensitivity to various design parameters is carefully analyzed. A comparison is also carried out with other "RTT-fair" TCP versions reported in the literature.

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

Categories and Subject Descriptors

C.2 [**Computer Systems Organisation**]: Computer-Communication Networks; C.2.2 [**Computer Systems Organisation**]: Computer-Communication Networks—*Network Protocols*; C.2.6 [**Computer Systems Organisation**]: Computer-Communication Networks—*Internetworking (C.2.2.)*

General Terms

TCP, Congestion Control, Optimization, Performance Evaluation

1. INTRODUCTION

The Internet success is based on the ability to provide a reliable medium for information exchange. In the current Internet the traffic control functionalities are provided by the Transmission Control Protocol (TCP) in an end-to-end fashion. TCP has been designed to provide a connection oriented, reliable, and fair service in the ARPANET [48], initially, and in the Internet later. Reliability and congestion control embody the two major issues addressed by TCP [20]. To achieve the second goal, TCP adapts the sending rate to avoid network overflow or fall 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 [11]).

The congestion control scheme implemented by TCP falls in the AIMD (Additive Increase, Multiplicative Decrease) class of algorithms. The currently deployed scheme considers packet loss as an indicator of network congestion and reacts with a drastic reduction of the sending rate [2]. The loss rate experienced over the Internet can be well beyond the optimal operating conditions for TCP thus leading to poor network utilization [39]. TCP New Reno and TCP Sack partially correct this problem and improve efficiency in the loss recovery phase. However, the overall performance is still far from optimal on large pipe sizes, paths where the bandwidth-delay product is large. Practical approaches to "big pipe" scalability are the fine-tuning of New Reno parameters (i.e. receiver advertised windows and network buffers), the use of jumbo packets (i.e. 8Kbyte packets) and the opening of multiple TCP sessions in parallel.

All of these approaches were directed to enhance utilization. None of them directly addresses the issues of fairness and friendliness - the latter referring to the *behavior* towards legacy competing flows. One particular type of unfairness which is intrinsic in legacy TCP is RTT unfairness. We recall that in congestion avoidance the congestion window increases linearly with RTT. As a consequence,

small RTT sessions, with a faster dynamics in the window increase, get the lion share of bottleneck bandwidth. 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.

The RTT-unfairness has been recognized since early TCP protocol enhancement efforts. In [13], the authors of TCP New Reno proposed a solution to the protocol's RTT-unfairness. Unfortunately, that solution was shown to be unstable in [19]. The research community has never lost interest in the RTT-fairness problem, as it is witnessed by recent schemes that try to address it in their design [57] [24].

A different, proactive rather than reactive, approach to congestion control is exploited by Delay-based Congestion Avoidance algorithms (DCA) first introduced by Jain in [21]. DCA algorithms monitor the RTT experienced by the packets and react to its increase in the attempt to avoid network congestion before it occurs. Two examples of DCA algorithms are TCP Vegas and TCP DUAL [6] [27]. These schemes introduced a trade off between RTT fairness and efficiency; fairness can be achieved at the cost of throughput reduction [39]. DCA algorithms have re-gained popularity in the last couple of years [24]. The limitation of DCA algorithms is that they highly depend on the strong correlation between packet loss events and RTT increase prior to the loss event. This dependence has been proven to be weak as shown in [16] [39] [49]. In particular, the RTT probe performed by TCP is too coarse to correctly foresee congestion events [23].

In this paper we introduce TCP Libra [3], a sender side TCP variation designed to cope with RTT-unfairness in heterogeneous best-effort networks. TCP Libra controls the window based on RTT measurements, like the aforementioned DCA schemes. However, it is in a class of its own, that can be referred to as Delay-enhanced Congestion Control (DCC). A DCC algorithm takes into account the delay information it receives from the network, in the sense that it limits the aggressiveness of its congestion window increase based on RTT information, but in a milder way than DCA algorithms. The key point in the distinction between DCC and DCA algorithms is that DCC algorithms do not avoid congestion, they rather delay the point at which congestion will occur, while DCA algorithms try to avoid congestion in the network. As seen in [39] [49] the delay information is not consistent to give, by itself, a good prediction of congestion events. For this reason we believe that a DCC approach represents a better approach to the congestion reaction problem.

We will here analyze TCP libra in relation to its features of RTT-Fairness,efficiency and friendliness to TCP New Reno, thus being an ideal candidate for incremental deployment.

TCP Libra follows a holistic approach and features a modular design where each component is thought to cope with a single issue. We here validate TCP Libra top-down approach through analysis and simulation. In particular we present: (i) a component-wise analysis to determine how design choices impact the protocol behavior, (ii) a capacity sensitivity analysis aimed at the evaluating the impact of capacity estimation errors to TCP Libra behavior, (iii) a simulative performance study under realistic traffic scenarios aimed at determine TCP Libra's ability to deal with heterogeneous and changing network conditions.

The reminder of the paper is organized as follows. In Section 2 we briefly present TCP Libra algorithm and architecture while in Section 3 we formally introduce the network model. The model validation is outlined by Section 4. Section 5 discusses the tuning strategies and impact for Libra's components. Our sensitivity analysis is described in Section 6. In section 7 we study TCP Libra's performance as well as as several competing protocols in a realistic network conditions. A literature survey is reported in Section 8, while the paper is finally concluded by section 9.

2. TCP LIBRA: ARCHITECTURE AND ALGORITHM

TCP Libra has been designed to achieve the best possible trade-off among utilization, fairness and friendliness to TCP New Reno. These objectives have been pursued by designing single components studied to perform specific tasks. The main components of TCP Libra are:

1. *Capacity estimation*. The *capacity estimation* component estimates the capacity of the bottleneck link at the beginning of a new session. We will not describe in depth the mechanism by which the algorithm performs this task, but we will analyze what are the consequences of small errors in the estimation result on the stable behavior of the protocol.
2. *Fairness control*. The *fairness control* implements the fairness functionality of Libra, equalizing the throughput of heterogeneous round-trip time flows among each other. This component is mainly based on what was suggested by Floyd and Jacobsen in [13]. We add a further component to it that lowers the variance of throughput.
3. *Scalability control*. The *scalability control* receives as an input the capacity of the bottleneck link from the *capacity estimation* component and sets the slope of the window increase accordingly. We can justify intuitively this choice by observing that a higher bottleneck link capacity requires a greater speed in convergence to the fair share. We are here trying to avoid the well known drawback of TCP New Reno, i.e. slow convergence with large pipe sizes paths.
4. *Stability control*. The *stability control* makes sure that, taking as an input the share of buffer occupancy, the protocol operates in its stable region (we will later more formally define what we mean as *stable region*). This control acts as a gauge on the *scalability control*, which may be too aggressive in scenarios where a great number of flows share the same bottleneck link.
5. *Burstiness control*. This component determines when packets may be sent, making sure that the network is not injected with a heavy burst of traffic all at once. The reason is to avoid synchronization of losses and multiple reductions of the congestion window due to buffer overflows. In order to also avoid losses due to synchronized pacing strategies between flows that share the same bottleneck, we have implemented a randomized pacing strategy, that should prevent the problem.

In the following subsection we will present the algorithm and highlight the components we just enumerated.

2.1 Algorithm

TCP Libra falls in the class of the AIMD (additive increase, multiplicative decrease) algorithms and implements a sender side modification to the New Reno congestion control algorithm. TCP Libra reacts to queueing by diminishing the increment of the congestion window when the buffer occupancy increases. If packet loss is detected, TCP Libra decreases the congestion window by a variable multiplicative decrease factor. Here follows the 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).

where $window_n$ is the window size at time n, T_n the end-to-end round-trip time measured at time n, T_0, T_1 are fixed parameters and α_n is defined as:

$$\alpha_n = k_1 c e^{-k_2 \frac{T_n - T_{min}}{T_{max} - T_{min}}} \quad (1)$$

where k_1 and k_2 are fixed parameters, c is the capacity of the bottleneck link seen by the source, T_{min} and T_{max} are the minimum and the maximum round-trip times, respectively, experienced during the connection up to time n.

We can now map the architectural components of TCP Libra at an algorithmic level.

We may identify the *scalability control* as $k_1 C$, where k_1 is a fixed parameter that must be set taking into account the requirement of responsiveness of the algorithm, as well as the necessity to operate in a stable region.

The *fairness control* is implemented by the $\frac{T_n^2}{T_0+T_n}$ factor in the increase portion and the $\frac{1}{T_0+T_n}$ factor in the decrease portion. The T_n^2 term in the increase portion is a well known factor, that has already been conjectured by Floyd and Jacobsen in [13] and analyzed by Henderson and Kelly in [19] [18] [28]. The term $\frac{1}{T_0+T_n}$ gives us the control over the range of RTTs for which we are interested to equalize throughput. In cases in which $T_n << T_0$ the influence of T_n over the increase portion will be minimal, so we'll have that $\frac{T_n^2}{T_0+T_n} \approx \frac{T_n^2}{T_0}$. In cases where T_n is at least comparable to T_0, we'll have that $\frac{T_n^2}{T_0+T_n} \approx \frac{T_n}{2}$. We are somewhat penalizing the flows that exceed a certain maximum RTT that we may set. This may be a further control in case we want to penalize flows that exceed a certain RTT limit. The reason to do this is that a flow experiencing a RTT value above a certain threshold may be understood as experiencing some problem on the path. Since transmissions RTTs are bounded by upper limits of light propagation and buffering times, this may be a reasonable tradeoff: paths that are not performing properly should not be overflowed with packets.

The *stability control* section of the algorithm is represented by the term $e^{-k_2 \frac{T_n - T_{min}}{T_{max} - T_{min}}}$. This control function ensures that the algorithm slows down with the window increase rate when links become congested. Indeed, the term $\frac{T_n - T_{min}}{T_{max} - T_{min}}$ may be interpreted as the share of buffer occupancy. When this approaches unity, it has the effect to slow down the increase in the congestion window. The substantial difference with the schemes analyzed in [39], is that the TCP Libra algorithm does not strongly penalize the window dynamics when buffer build up is sensed. In both DUAL and Vegas [6] [27], when RTT grows beyond a threshold, the congestion window is decreased. Here we do not decrease the window, rather we reduce the increase of the window. We also argue that this is the correct strategy in order to achieve $fair$ friendliness to the legacy TCP implementation, since the reaction to queuing delay (which the legacy protocol ignores in its dynamics) does not heavily compromise the performance of TCP Libra when competing with TCP New Reno. k_2 is a fixed parameter that sets the responsiveness to queue build up. A greater k_2 implies that TCP Libra will be functioning with a larger stability region, utilize its share more efficiently but, on the other hand, suffer higher throughput loss when competing against legacy protocol flows.

3. PROBLEM FORMULATION

3.1 Network Model

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:

$$R_{lr} = \begin{cases} 1,\ if\ link\ l \in L\ is\ utilized\ by\ source\ r \in S \\ 0,\ otherwise \end{cases} \quad (2)$$

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:

$$y_l(t) = \sum_r R_{lr} x_r(t - \tau_{lr}^f) \quad (3)$$

where τ_{lr}^f is the forward delay from source r to link l. We also define the backward delay in the feedback path from link l to source r as τ_{lr}^r. We have that the round-trip time may then be written as $T_r = \tau_{lr}^f + \tau_{lr}^r$. 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 queuing delay. TCP Libra is primarily sensitive to packet loss, while it adapts to buffer occupancy while increasing its window. The aggregate price, in terms of packet loss, seen by source r is:

$$\lambda_r(t) = 1 - \prod_{l \in r}(1 - \mu_l(t - \tau_{lr}^r)) \quad (4)$$

where $\mu_j(t)$ is defined as:

$$\mu_l(t) = f_l(\sum_{r:l\in r} x_r(t-\tau_{lr}^f)) \quad (5)$$

where $f_l(y(t))$ is the price signal sent by link l at time t. The price, $\mu_l(t)$, also known as $barrier$ [5] is a function of the link flow. The price perceived by the source depends on the aggregate price signals sent by all the resources on route r, as seen in (4).

3.2 Queue Models

In this paper we will focus on routers implementing either droptail, which is the type of queue implemented in the great majority of routers on the Internet, or RED [14] which is the most popular active queue management technique. We can model the probability of loss deriving from the droptail queue strategy as follows:

$$\mu_l(t) = \begin{cases} 1 \ \ if \ q_l(t) > B \\ 0 \ \ otherwise \end{cases} \quad (6)$$

where $q(t)$, queuing time at time t, is calculated as:

$$q_l(t) = \int [y(t) - c]^+ dt \quad (7)$$

The dynamics of RED is captured, at a fluid flow level, by the following:

$$\mu_l(t) = \begin{cases} 1 \ \ if \ q_l(t) > B \\ \rho_l(q_l(t) - b) \ \ if \ b \le q_l(t) \le B \\ 0 \ \ if \ 0 \le q_l(t) < b \end{cases} \quad (8)$$

with:

$$\rho_l = \frac{\mu_{max}}{B-b} \quad (9)$$

and $q_l(t)$ is the average queue length, which is calculated with an exponential averaging filter.

4. MODEL VALIDATION

4.1 Fluid Flow Model and Network Stability Analysis

We carry out and complete here the analysis of TCP Libra, which had began in [3]. This time we will refer to the network case, where the system will be studied in its linear approximation around the equilibrium point and conditions for local stability will be shown. This analysis derives directly from the analysis by Vinnicombe [54] and Kelly [28] for TCP New Reno, where the general case of a non-linear increase and decrease algorithm is considered. We will also consider two different dual strategies in the network, namely droptail and RED [14] [9], seeing how they affect the primal algorithm operation. We can here recall the fluid flow non-linear expression of TCP Libra [3], for the r-th flow, is:

$$\frac{dx_r(t)}{dt} = \frac{x_r(t-\tilde{T}_r)}{\tilde{T}_r}\left(\frac{\tilde{\alpha}_r\tilde{T}_r}{x_r(t)(\tilde{T}_r+1)}(1-\lambda_r(t)) - \frac{2}{3}\frac{x_r(t)\tilde{T}_r\lambda_r(t)}{\tilde{T}_r+1}\right) \quad (10)$$

We may linearize the above around its equilibrium points $(\tilde{x}_r, \tilde{\lambda}_r)$, where the average throughput may be expressed as:

$$\tilde{x}_r = \sqrt{\frac{a_r}{b_r}\frac{\alpha_r}{T_1}\frac{1-\tilde{\lambda_r}}{\tilde{\lambda_r}}} \quad (11)$$

and may write the linearized system as:

$$\delta\dot{x}_r(t) = -2\frac{b_r\tilde{\lambda_r}}{T_0+\tilde{T}_r}\sqrt{\frac{\tilde{\alpha}_r a_r}{T_1 b_r}\left(\frac{1-\tilde{\lambda_r}}{\tilde{\lambda_r}}\right)}\delta x_r(t) - \frac{\tilde{\alpha}_r a_r}{(T_0+\tilde{T}_r)\tilde{\lambda}_r}\delta\lambda_r(t) \quad (12)$$

Recalling the result by Vinnicombe [54], we may state the following:

THEOREM 4.1. *Consider the TCP Libra congestion control algorithm (43). Lets suppose the congestion control algorithm is responding to the input prices given by a single bottleneck link l. We may then write that:*

$$\lambda_r(t) = \mu_l(t-\tau_{lr}^r) = f_l(\sum_{r:l\in r} x_r(t-\tau_{lr}^f-\tau_{lr}^r)) \quad (13)$$

Under this assumption the condition:

$$\frac{\tilde{\alpha}_r\tilde{T}_r}{(\tilde{T}_r+T_0)\tilde{x}_r}\frac{1-\tilde{\lambda}_r}{\tilde{\lambda}_r}\frac{y_l f_l'(y_l)}{1-f_l(y_l)} < \frac{\pi}{2} \quad (14)$$

is a sufficient condition for the the linearized system (12) to be asymptotically stable.

LEMMA 4.2. *If $f_l(y)$ is Lipschitz, under the condition $y_l \ge x_l > 0$, there $\exists\ \beta$ such that:*

$$y_l f_l'(y_l) \le \beta f_l(y_l) \quad (15)$$

condition (14) may be written as:

$$\frac{\tilde{\alpha}_r\tilde{T}_r}{(\tilde{T}_r+T_0)\tilde{x}_r}\frac{1-\tilde{\lambda}_r}{\tilde{\lambda}_r}\frac{y_l f_l'(y_l)}{1-f_l(y_l)} \le \frac{\tilde{\alpha}_r\tilde{T}_r}{(\tilde{T}_r+T_0)\tilde{x}_r}\frac{1-\tilde{\lambda}_r}{\tilde{\lambda}_r}\frac{\beta f_l(y_l)}{1-f_l(y_l)} \quad (16)$$

$$= \frac{\tilde{\alpha}_r\tilde{T}_r}{(\tilde{T}_r+T_0)\tilde{x}_r}\beta \quad (17)$$

$$< \frac{\pi}{2} \quad (18)$$

The above lemma, under the given assumption, confirms what we have already observed, for the single source single link case, in [3].

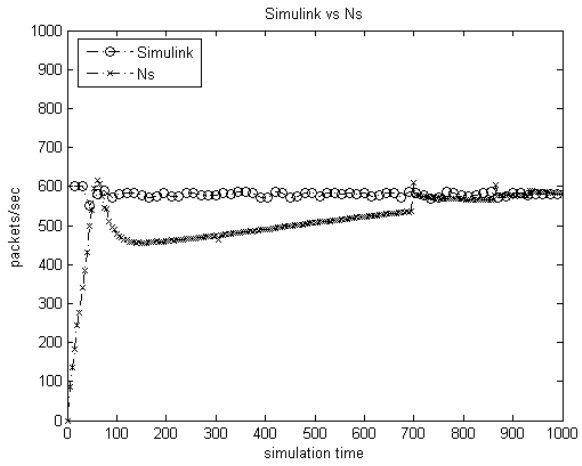

Figure 1: ***Flow 1, with RTT 10ms, competing with Flow 2 on a 15Mb bottleneck. Packet size is set to 1500 bytes. The buffer at the bottleneck link implements droptail, with size 600pkts.*** T_0 ***is set to*** 1 ***and*** $k_1 = 2$***.***

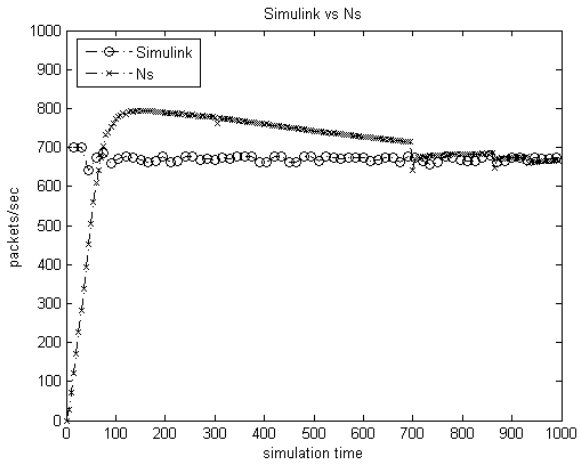

Figure 2: ***Flow 2, with RTT 82ms, competing with Flow 1 on a 15Mb bottleneck. Packet size is set to 1500 bytes. The buffer at the bottleneck link implements droptail, with size 600pkts.*** T_0 ***is set to*** 1 ***and*** $k_1 = 2$***.***

High values of k_1 lead to instability, while low values of k_1 lead to extremely slow convergence, as may be understood both intuitively (smaller k_1 represents a smaller increase rate in congestion control phase) and analytically. An opposite reasoning may be done for k_2 and T_0. We will validate these arguments in the following section by simulation means.

In Fig. 1 and 2 we compare the result of a NS-2 simulation against the fluid flow model implemented in Simulink. We obtained the average values in (12) from the simulation in NS-2 and plugged them in the simulink model. The queue has been modeled as an integrator (please refer to Appendix II for the details), while the droptail behavior has been implemented as a step, on-off non-linearity (i.e. depending on the queue size congestion information is fed back to the transmitters). The simulink graphs in Fig. 1 and 2 are the result of a perturbation, Δx_1 and Δx_2, with respect to the stable values $\tilde{x}_1$ and $\tilde{x}_2$.

5. COMPONENT TUNING

TCP Libra performance, other than the "prices" and accuracy of the capacity estimate, depends on the parameters' tuning. We analyze here the influence of the parameters tuning and the correspondence between the expected values as determined by the analysis and the actual behavior of the algorithm. We also want to investigate what are the consequences of choosing a value, for each parameter, instead of another and which is the influence of parameters on each component of TCP Libra. We, hence, focus the simulative investigation, via Matlab and NS-2, on effects of tuning over the variance of instantaneous throughput. Other aspects of parameter tuning that concern stability and efficiency, have been already treated by simulative means in [3].

5.1 Fairness Control

We have "captured" the $fairness\ control$ component in the term $\frac{T_n^2}{T_0+T_n}$ in the increase portion of the algorithm and in the term $\frac{T_1}{T_0+T_n}$ in the decrease portion of the algorithm. We can control the variance in the throughput and the convergence speed by setting the parameter T_0. An increase in T_0 implements a reduction in throughput variance at the expense of convergence speed. For this reason, an increase in T_0 should be followed by an increase in k_1. We here produce three examples, implemented in Matlab and in NS-2, where we increase both T_0 and k_1. As we can see from Fig. 3-8, as expected, an increase in T_0 results in a more regular pattern of the curve representing the instantaneous throughput.

5.2 Stability Control

Stability is one of the key contributions of TCP Libra together with scalability and fairness. The *stability control* portion of the algorithm, which is reflected in the additive phase of the algorithm by the coefficient $e^{-k_2 \frac{T_n - T_{min}}{T_{max} - T_{min}}}$, is the control that enables TCP Libra to operate with the desired tradeoff between stability and efficiency. This control fixes the problem that has been identified in [13] [19], where the effect of setting the increase coefficient c, that multiplies $\frac{T_n^2}{window_n}$ cannot be precisely determined. In fact, a high value of c will drive to instability problems at some point, while low values will give a good stability but lead to a very slow convergence. The "right" operational value of coeffcient c cannot be determined a priori, without introducing adaptivity to network conditions.

5.3 Burstiness Control

TCP Libra's burstiness control has been designed with the objective of avoiding two specific problems: synchronization of loss events and failure to collect significant samples of the round-trip time experienced by the flow. From previous studies [1], we have learned that pacing has a de-synchronizing effect that leads to higher efficiency in steady state. We improve the cyclic pacing scheme by introducing randomness. Namely, packets are uniformily, but randomly sent within a round-trip time.

We argue that our pacing scheme will help obtaining significant RTT measurements. The under-sampling problem reported in [49] remains, but since our scheme implements a DCC, and not a DCA algorithm, the influence of innaccurate sample measurements is less disruptive.

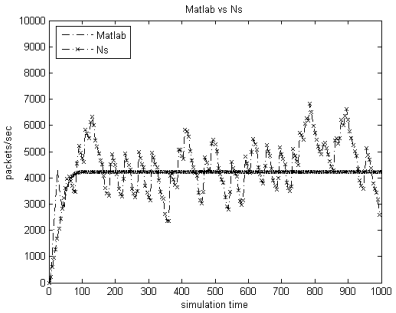

Figure 3: ***Flow 1, with RTT 10ms, competing with Flow 2 on a 100Mb bottleneck.*** T_0 ***is set to*** 1 ***and*** $k_1 = 2$***. We set the slow start threshold to*** 1 ***in NS2. Packet size is 1500 bytes. The bottleneck router implements RED. The RED parameters are,*** μ_{max} ***= 0.1, b = 150pkts and B = 600pkts. The queue averaging weight is set to 0.002.***

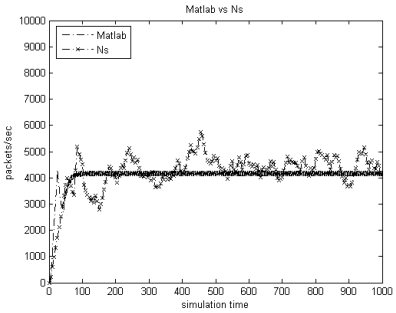

Figure 4: ***Flow 1, with RTT 10ms, competing with Flow 2 on a 100Mb bottleneck.*** T_0 ***is set to*** 4 ***and*** $k_1 = 8$***. We set the slow start threshold to*** 1 ***in NS2. Packet size is 1500 bytes. The bottleneck router implements RED. The RED parameters are,*** μ_{max} ***= 0.1, b = 150pkts and B = 600pkts. The queue averaging weight is set to 0.002.***

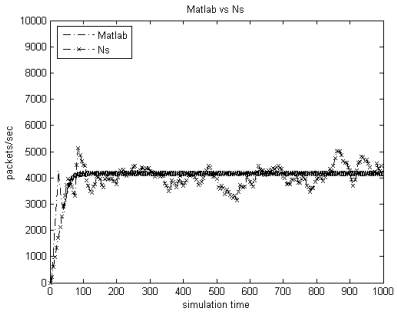

Figure 5: ***Flow 1, with RTT 10ms, competing with Flow 2 on a 100Mb bottleneck.*** T_0 ***is set to*** 8 ***and*** $k_1 = 16$***. We set the slow start threshold to*** 1 ***in NS2. Packet size is 1500 bytes. The bottleneck router implements RED. The RED parameters are,*** μ_{max} ***= 0.1, b = 150pkts and B = 600pkts. The queue averaging weight is set to 0.002.***

5.4 Scalability Control

The scalability feature of TCP Libra was designed to ensure that TCP Libra will scale to any delay bandwidth product. There will be no need to change the protocol at some point for scalability issues. It is well known from the literature that at some point of its life TCP New Reno will need to be retrofitted or replaced. The constant increase in Internet bandwidth will eventually make the protocol itself the bottleneck. TCP Libra scales its window increase rate proportionally to the bottleneck capacity. A higher value of k_1 makes the algorithm more aggressive. This value must be selected jointly with k_2.

6. SENSITIVITY ANALYSIS

The sensitivity analysis of TCP Libra aims at determining how TCP Libra is sensitive to errors in the estimate of the capacity. To this end we derive a simple linearized model, which is based on the assumption that the percentage error in the estimate is low.

6.1 Capacity Sensitivity

TCP Libra relies on the capacity estimation, which contributes to esteem the α factor. We, hence, need to investigate the impact of estimation errors on the algorithm behavior. Recalling (43) we must now introduce a new variable, the estimated capacity of the bottleneck link $c(t)$. In Appendix II we evaluate the effect of errors in the bottleneck link capacity estimation taking into account the "perturbation" applied to the estimated bottleneck link capacity at the beginning of each session. We find that the error percentage error, with respect to the throughput $\tilde{x}_r$, is given by:

$$Relative\ error = \frac{Absolute\ error}{\tilde{x}_r} = \frac{1}{2}\gamma \qquad (19)$$

The TCP Libra control mechanism cannot, of course, correct the error. However, it has the remarkable property of reducing its effect by the factor 2. In terms of fairness, if two TCP users are sharing the same bottleneck and one is overestimating capacity by 10%, it will achieve a throughput 5% higher than the user equipped with an accurate estimate. This "fairness" error is quite acceptable, especially if we consider the fact that state of the art bottleneck capacity estimators for wired Internet path are quite accurate (well within 10% errors if there is time to collect enough samples - as is the case in off line estimation).

The above sensitivity analysis was carried out on a linearized model. To verify that the $\frac{1}{2}$ *factor impact* property applies also to *large* errors, we have carried out an experiment with 50% error. Both Matlab and NS-2 simulation results are in presented Fig. 9 and 10.

7. PERFORMANCE EVALUATION

In this section we evaluate TCP Libra using the NS-2 [50] simulation platform. We also compare it to the *de facto* standard transport protocol in the Internet, namely TCP Sack, and to other TCP versions which claim some RTT-fairness. Further RTT-fairness claiming protocols were previously evaluated against TCP Libra in [3]. In that paper, a simple dumbbell shaped topology with concurrent flows but no background traffic was utilized to compare the performance of TCP Libra, TCP New Reno, TCP Vegas, CUBIC and TCP Hybla over configurations featured by including connections with different RTTs. In particular, we gathered results that demonstrate how our TCP Libra was the only one really guaranteeing a high degree of RTT-fairness, whilst remaining friendly towards TCP New Reno. More in detail we showed the advantage of utilizing TCP Libra as the transport protocol by evaluating the following parameters:

i available bandwidth utilization;

ii intra-protocol RTT-fairness;

iii inter-protocol RTT-fairness;

iv friendliness to the legacy protocol TCP New Reno;

v scalability to many flows, large capacity and large RTT.

We are now aimed at studying the behavior of TCP Libra in more complex scenarios that include heterogeneous cross traffic, i.e., concurrent long and short lasting TCP flows, and web traffic. Moreover, we compare our protocol with other RTT-fair proposals, namely: TCP Sack, BIC and TCP Fast. For TCP Sack, we have used the existing NS-2 option; for BIC and TCP Fast we have downloaded and

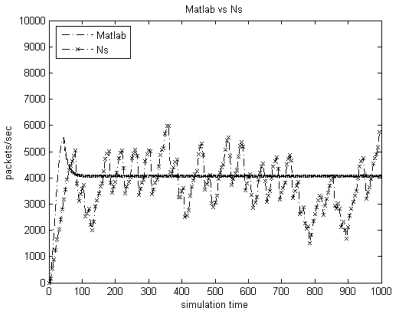

Figure 6: ***Flow 2, with RTT 83ms, competing with Flow 1 on a 100Mb bottleneck.*** T_0 ***is set to*** 1 ***and*** $k_1 = 2$***. We set the slow start threshold to*** 1 ***in NS2. Packet size is 1500 bytes. The bottleneck router implements RED. The RED parameters are,*** μ_{max} ***= 0.1, b = 150pkts and B = 600pkts. The queue averaging weight is set to 0.002.***

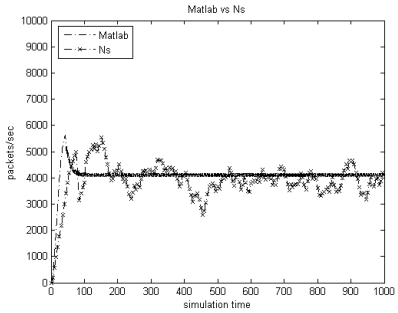

Figure 7: ***Flow 2, with RTT 83ms, competing with Flow 1 on a 100Mb bottleneck.*** T_0 ***is set to*** 4 ***and*** $k_1 = 8$***. We set the slow start threshold to*** 1 ***in NS2. Packet size is 1500 bytes. The bottleneck router implements RED. The RED parameters are,*** μ_{max} ***= 0.1, b = 150pkts and B = 600pkts. The queue averaging weight is set to 0.002.***

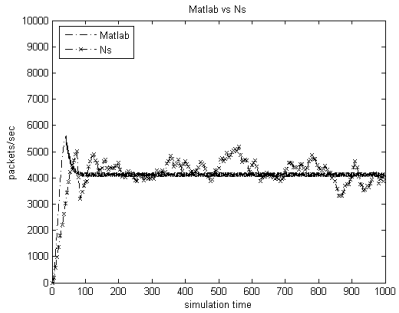

Figure 8: ***Flow 2, with RTT 83ms, competing with Flow 1 on a 100Mb bottleneck.*** T_0 ***is set to*** 8 ***and*** $k_1 = 16$***.We set the slow start threshold to*** 1 ***in NS2. Packet size is 1500 bytes. The bottleneck router implements RED. The RED parameters are,*** μ_{max} ***= 0.1, b = 150pkts and B = 600pkts. The queue averaging weight is set to 0.002.***

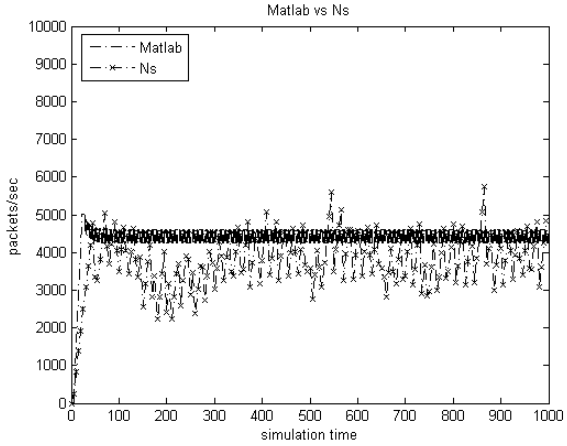

Figure 9: ***Flow 1, with RTT 82.6ms, competing with Flow 2 on a 100Mb bottleneck.*** T_0 ***is set to*** 1 ***and*** $k_1 = 2$***. All other parameters are set as before. This flow overestimates the bottleneck link capacity by 50%.***

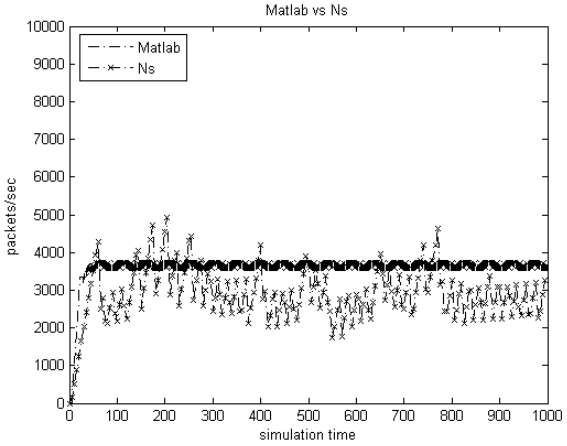

Figure 10: ***Flow 2, with RTT 83.4ms, competing with Flow 1 on a 100Mb bottleneck.*** T_0 ***is set to*** 1 ***and*** $k_1 = 2$***. All other parameters are set as before. This flow estimates correctly the bottleneck link capacity.***

installed NS-2 modules developed by the authors and available online [33] [51]. Finally, we have developed the module representing our TCP Libra. We have set parameters for BIC and TCP Fast as inspired by their simulation scripts available online and following the directions provided by their official web sites.

Each experiment was run for 1000 seconds of simulation time in order to gather results of the various protocols in their steady state. We have adopted two different topologies for our analysis and implemented several scenarios. The bottleneck was set with different values and its buffer size was equal to the pipe size corresponding, alternatively, to the bottleneck link pipe or to the longest connection pipe. Routers along the path were configured to implement both drop tail and adaptive RED queuing policies. The advertised window for each connection was set larger than the whole pipe size so as to still occasionally drop packets, even when that connection is the only active one. Finally, the packet size was set always equal to 1000 bytes.

Unfortunately, space limitation allows us to present only a limited subset of our simulations. In particular, we decided to show results only for the 100Mbps bottleneck configuration, as it well represents a general case and results coming from other configurations did not show particular differences worth to be deeply analyzed or compared with the chosen one. For this bottleneck size, the alpha and beta parameters utilized by TCP Fast were always set equal to 100. BIC parameters were instead set as in [57].

7.1 Benchmark Suite

The scientific community is feeling as more and more urgent the necessity of defining a valid benchmark suite to objectively evaluate different protocols [12] [15] [56] [8]. Indeed, whenever a new protocol is proposed, the authors independently choose the set of experiments they are going to utilize to measure performance. However, some experiment may exploit a too simplicistic configuration which does not embody the real scenario in which the protocol would be utilized, while others may be too complex to permit

precise information acquisition on the new protocols behavior.

We decided to run a large set of simulations involving different topologies, parameters configuration, typology of cross traffic, and network workload. Among these, we had then to choose a subset of information that may not exceed the page limitation of this paper but, at the same time, provide relevant information.

Inspired by [12], a set of relevant metrics have been evaluated: *fairness*, *throughput*, and *deployability*. Since TCP Libra was developed to address the RTT-unfairness of regular TCP, one of the evaluated metric had obviously to be the achieved fairness degree and, in particular, the RTT-fairness degree. The efficiency in utilizing the available bandwidth is important as well, since there would be no point in having fairness if this implied a conspicuous general reduction of transmission speed (e.g., transmitting only one packet every second would be an unpractical, even if RTT-fair, solution). Finally, deployability regards the integration feasibility of the new protocol within the current Internet. This includes the amount of modifications that has to be performed both in terms of complexity of the code and of involved nodes typology (i.e., routers, end-nodes). Furthermore, the large volume of nodes and domains composing the Internet does not allow a sudden upgrade from one version of TCP to a new one but rather a slow evolution. In order to be factually deployed over the Internet, any new protocol must hence be able to coexist with the legacy one. We deem that the deployability metric should also include a measure for friendliness toward existing protocols (i.e., TCP New Reno or TCP Sack).

We define a new TCP as friendly toward the legacy one if the former does not affect the performance of the latter and viceversa, when concurrently employed. In essence, the new TCP may even obtain better results than those achievable by concurrent traditional TCP; however, it should do this by exploiting the inefficiencies of the latter rather than aggressively overwhelming its functionalities and affecting its performance.

The most largely adopted tool for evaluating the fairness degree of TCP flows is Jain's Fairness index [22]. This index can be used also to specifically evaluate the RTT-fairness as required in our case. We have hence computed its value for every configuration adopted in our simulations. However, it has to be said that there is not a complete consensus in the scientific community about the need for reaching the RTT-fairness goal. In particular, flows that traverse different number of congested node may deserve different achievable throughput [12] [9] [13]. In point of this, we believe that RTT-fairness should be achieved whenever there is an identical amount of utilized resources, regardless of the RTTs experienced by the various concurrent flows. Our standpoint may align the diverse views on this topic as we use the term resource to indicate the number of queues (and therefore congested links) that each flow has to travel on its path from the source to the destination and not the physical length of the traversed links.

The efficiency of the protocols has been analyzed through the goodput and, in particular, through its value with respect to its maximum achievable value on the considered bottleneck link. The goodput can be distinguished from the throughput since the former represents the subset of the latter that embodies the useful traffic effectively received by the destination. To measure this metric in our simulations we have counted the number of packets acked by the receiver and compared this value with the maximum achievable amount of packets that could be sent on the considered link.

Finally, friendliness towards legacy TCP represents an important metric but also a difficult one to be evaluated. Indeed, it is not sufficient that a new protocol obtains higher (or lower) sending rate with respect to regular TCP to determine unfriendliness: the new protocol has to be proven as affecting the performance of the old one. For this reason, we have first computed the average goodput of flows when only TCP Sack is utilized for all of them. We have then substituted TCP Sack with another protocol on half of the connections and verified the average value of the goodput for the remaining TCP Sack flows. This clearly shows how (if) the new protocol impact on the legacy one.

7.2 Parking Lot Topology

A *parking lot topology* is a linear network generally used as an example for a multiple bottleneck links configuration. Its links can be featured with different bandwidth and delays and its flows may traverse it utilizing diverse hops. It is frequently exploited in scientific literature as a simplification for representing connections with different RTTs, number of hops, and cross traffic between client and server.

Fig. 11 shows a simulation network involving 8 connections. Two of these connections (i.e., flows 1 and 2) have 180ms of minimum RTT and traverse 9 hops, other two (i.e., flows 3 and 4) have 90ms of minimum RTT and traverse 9 hops, and the remaining 4 flows (i.e., flows 5-8) utilize 3 hop paths featured with 30ms of minimum RTT.

To overcome the phase effect artificially introduced by NS-2, the actual minimum RTT of the aforementioned connections was slightly modified in positive or negative with a maximum difference of 1ms [32]. Moreover, all the flows where randomly starting within the first 5 seconds of simulation.

In all the simulations we run, for every considered protocol, the Jains index corresponding to flows 5-8 hits values superior to 0.99 thus denoting a very high fairness among them. In Fig. 12, we present the Jains index values achieved during the same simulations by flows 1-4, for each different protocol, and with the buffer set equal to the bottleneck pipe size and the longest connection pipe size, respectively. As clearly showed by the chart, TCP Libra is the only protocol that is never affected by RTT differences among the various connections. TCP Sack achieves a very high Jains index when large buffers are present on routers and BIC shows acceptable values regardless of the buffer size; yet we have to analyze figures 13 14 to reconsider (in negative) their results. Fig. 13, in fact, demonstrates that BIC significantly suffers from the considered topology and that its fairness degree is mainly obtained at the cost of a very low goodput. A similar problem is experienced also by TCP Sack. Simulation results showed that the short RTT connections 5-8 largely affect the medium and long RTT connections 1-4, leaving them only a very limited share of the available bandwidth. This can be deduced also by looking at Fig. 14 where the Jains index is computed over all the 8 connections. TCP Sack, which had an acceptable Jains 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. 14 also shows that, TCP Libra results one of the fairest protocol when evaluating long and short RTT connections together, even if it does not hit optimal fairness degrees. However, this is perfectly coherent with what we declared in section 7.1: "RTT-fairness should be achieved whenever there is an identical amount of uti-

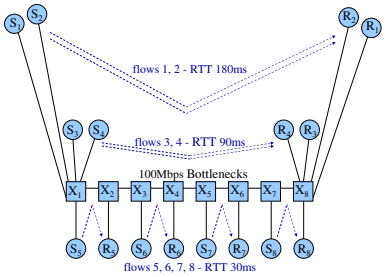

Figure 11: ***Parking Lot Topology***

.

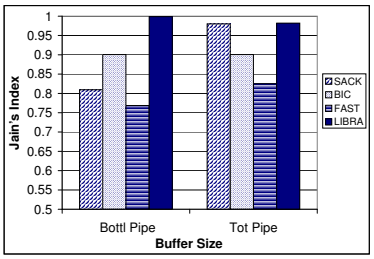

Figure 12: ***Jains index values achieved during the same simulations by flows 1-4, for each different protocol, and with the buffer set equal to the bottleneck pipe size and the longest connection pipe size, respectively.***

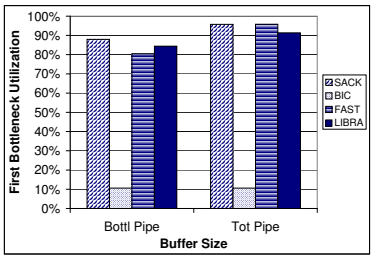

Figure 13: ***Efficiency in the Parking Lot Topology***

.

lized resources, regardless of the RTTs experienced by the various concurrent flows". In fact, optimal RTT-fairness is achieved among flows utilizing the same number of congested queues along the path between sender and receiver (i.e., flows 1-4 encounter 4 congested queues, while flows 5-8 encounters only 1).

This claim is confirmed also by Fig. 15 where the Jains index is showed for TCP Libra supporting a varying number of short RTT connections. In particular, "0 cross tr" corresponds to having no cross traffic along the backbone (i.e. only flows 1-4 are running), "1 cross tr" means that flow 5 was active, and so on until "4 cross tr" which restores the original configuration where all the 8 connections are simultaneously transmitting packets. In the "1 cross tr" case, all the 5 active flows share the unique congested link in the configuration thus achieving optimal RTT-fairness regardless of the RTTs experienced by the various flows and the number of hops on their paths. As long as we increase the number of cross traffic along the backbone, the difference in terms of number of traversed congested link augments thus generating unfairness among flows 1-4 and flows 5-8, even if maintaining the fairness within these two groups.

The deployability of the various protocols is quantitatively evaluated in Fig. 16 by measuring the goodput of TCP Sack flows when coexisting with other flows exploiting an alternative transport protocol. More in detail, the flows were evenly divided between TCP Sack and, in turn, each of the other protocols. 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). Fig. 16 presents a set of columns for each of the tested protocols; the first one corresponds to the referring case of 8 TCP Sack flows, while the other three sets represent the goodput achieved in average by TCP Sack on each of its flow classes (in terms of RTT) and as the sum of all of its driven connections.

Considering only the total goodput achieved, all the proposed protocols seems to be friendly towards TCP Sack. However, it is evident that TCP Fast annihilates the TCP Sack multihop connections characterized by 90ms and 180ms of RTT. Conversely, when BIC is employed on competing flows, TCP Sack increases its achieved goodput. As seen before, BIC suffers in this kind of topology, without reaching decent data rate, thus leaving a lot of space to concurrent protocols. The unfriendliness of BIC toward TCP Sack is hence expressed by the impossibility of coexisting efficiently with the legacy transport protocol in this scenario.

Therefore, TCP Libra is the only transport protocol that results factually friendly toward TCP Sack. The aggregate throughput of the concurrent TCP Sack, in fact, diminishes of only about 11% with a desirable, even if narrow, redistribution of the goodput from the 30ms RTT flows to the 90ms and 180ms RTT ones. We could hence say that TCP Libra also helps coexisting TCP Sack flows to (slightly) improve their fairness degree. Furthermore, this result is achieved whilst preserving the inter-protocol RTT fairness among its driven flows.

7.3 Dumbbell Topology

The *dumbbell topology* is widely adopted for network simulations. Its aim is that of simply representing a set of connections sharing the same bottleneck. We have used it to analyze how concurrent heterogeneous traffic may affect the performance of the proposed protocols.

For our experiments, we have adapted the simulation scripts utilized by [57] and available online. Fig. 17 shows with more details our implementation of the dumbbell topology. Similar to the parking lot topology, each link is configured so as to have different RTTs and different starting times in order to avoid the phase effect. In particular, the minimum RTT is around 40ms for the short connections (from S_2 to R_2 and between B_i and C_i) and around 240ms for the longest one (from S_1 to R_1).

These simulation scripts are able to generate a significant amount of concurrent traffic; in particular, two flows, one going from S_1 to R_1, the other one going from S_2 to R_2, featured with different RTTs and relying on the evaluated transport protocol compete for the bottleneck with other traffic. Concurrently, other flows go from B_i to C_i, or from C_j to B_j and, specifically, i) 4 regular long-lived TCP Sack flows traveling forward, ii) 4 regular long-lived TCP Sack flows traveling backward, iii) 25 small TCP flows having their advertised window limited to 64, and iv) an amount of web traffic in both directions able to occupy from 20% to 50% of the available bottleneck bandwidth when no other flows are present.

For the presented results, the buffer size at the bottleneck was alter-

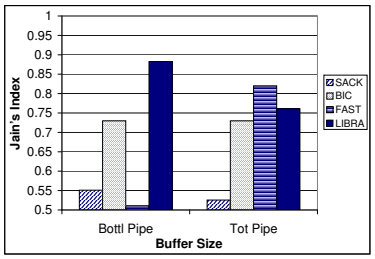

Figure 14: ***Jains index is computed over all the 8 connections. TCP Sack, which had an acceptable Jains index for connections 1-4 and an optimal one for connections 5-8, obtains a very poor value when considering all the 8 connections together.***

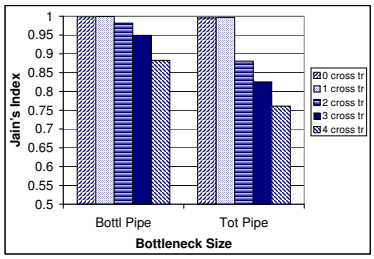

Figure 15: ***Jains index: TCP Libra is employed for a varying number of short RTT connections. In particular, 0 cross tr corresponds to having no cross traffic along the backbone (i.e. only flows 1-4 are running), 1 cross tr means that flow 5 was active, and so on until 4 cross tr which restores the original configuration where all the 8 connections are simultaneously transmitting packets.***

.

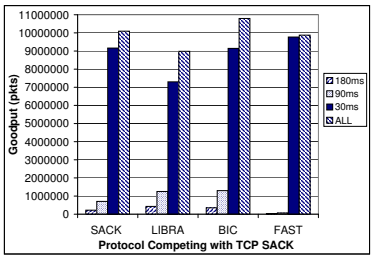

Figure 16: ***Friendliness: TCP Sack was utilized on flows 2 (180ms of RTT), 4 (90ms of RTT), 6 and 8 (30 ms of RTT each), while the evaluated alternative protocol was utilized on flows 1 (180ms of RTT), 3 (90ms of RTT), 5 and 7 (30 ms of RTT each) — Parking Lot Topology***

.

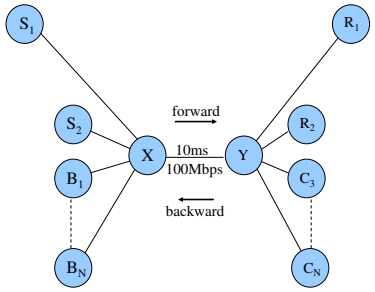

Figure 17: ***Dumbbell Lot Topology***

.

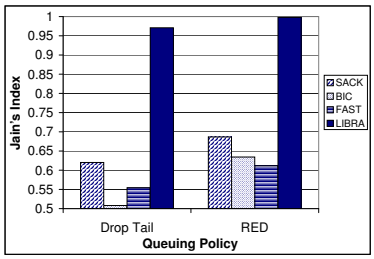

Figure 18: ***Jains index values achieved by the evaluated protocol while competing with a large amount of concurrent traffic. The concurrent connections were both short and long lived TCP session in both directions.***

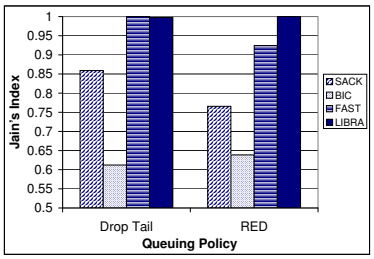

Figure 19: ***Jains index values achieved by the evaluated protocol while competing with a large amount of concurrent traffic. The concurrent connections were both short and long lived TCP session in both directions. - Buffer= longest connection pipe size***

natively set equal to the bottleneck link pipe size or to the longest connection pipe size. Results for the former case are reported in Fig. 18 while results for the latter are in Fig. 19. As it is easily observable, TCP Libra generally overwhelm the other protocols in terms of achieved fairness. When small buffers are employed, this result is even more evident since all the other protocols performs very poorly in terms of fairness.

8. 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 [13]. They also proposed a solution for the RTT-bias based on a constant increase algorithm 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 [9] [10]. 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 [14]. Henderson in [18] [19] 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 [43] and [44]. Additionally, in [42] 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.[2]

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 [31]. In [30] and [17] it was shown that TCP stability can be achieved in the aforementioned network model if the TCP utility function is concave [28] [29]. Kelly's results have been extended by several researchers. In particular, Vinnicombe, Massoulie, Johari and Tan addressed the stability in network congestion control schemes [26]. 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 [53] [55] [41]. Additionally Vinnicombe and Massoulie derived the stability condition for scenarios with heterogeneous delays [54] [40]. 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 [35] [45] [47] [46] [37] [34]. 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 [24, 25] [4] [36].

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 BIC and CUBIC [57] [52] feature a linear RTT-fairnes, TCP Hybla [7] enhances the solution proposed by Floyd in [10] and provides RTT-fairness under a certain stability bound, TCP Vegas and FAST [6] [24] provide good RTT-fairness when implementing 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, fairness 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.

[2]The fluid model presented in [42] 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.

9. CONCLUSIONS AND FUTURE WORK

In this paper we introduced and analyzed TCP Libra, a new RTT delay measurement aware protocol. The results show that TCP Libra is RTT-fair, throughput efficient and friendly to legacy TCP protocols. TCP Libra is a sender-side extension, thus allowing expedited deployment in the network. We have developed an analytic model of TCP-Libra's stability and have carefully evaluated its sensitivity to various protocol parameters. A critical parameter in TCP Libra implementation is bottleneck capacity C. The current design assumes prior knowledge of C, using source initiated estimates collected off line (e.g., CapProbe, Pathrate, etc) or during the slow start phase. An error in capacity estimate may lead to unfair sharing. Analytic and simulation results show that the resulting flow imbalance is less than the capacity error. The simulation results support the main innovative claims of TCP Libra, namely: the scheme is TCP fair over many diverse scenarios with a broad range of network speeds and RTT ratios; the scheme is friendly to legacy TCP protocols, and; the RTT fairness enforcement does not penalize efficiency.

More work is under way in several directions: incorporation of the bottleneck capacity estimation technique in the TCP Libra source only implementation; evaluation of the accuracy of the capacity estimate; extensions to mixed wired and wireless scenarios with random errors.

Acknowledgments

This work has been partially supported by the Italian Ministry for University and Research (M.I.U.R.) via the Interlink and E-Grid Projects; and by the National Science Foundation via the Whynet Project.
Moreover we wish to extend our gratitude to the Campus Academic Technology Services for providing their high performance GRID computing platform that allowed us to gather results from thousands of simulation runs in a relatively short time. Finally we wish to thank Jwei Chen for his help with Matlab.

10. REFERENCES

[1] A. Aggarwal, S. Savage, and T. Anderson. Understanding the performance of TCP pacing. In *INFOCOM (3)*, pages 1157–1165, 2000.

[2] M. Allman, V. Paxon, and W. Stevens. Rfc2581: Tcp congestion control. Technical report, Internet Engineering Task Force (IETF), 1999.

[3] A. Anonymous. Tcp libra: Exploring rtt-fairness for tcp. Technical report, Anonymous Institution, Computer Science Dept. TR-050037, 2005.

[4] S. Athuraliya, D. E. Lapsley, and S. H. Low. An enhanced random early marking algorithm for internet flow control. In *INFOCOM (3)*, pages 1425–1434, 2000.

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

[6] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson. TCP vegas: New techniques for congestion detection and avoidance. In *SIGCOMM*, pages 24–35, 1994.

[7] C. Caini and R. Ferrincelli. Tcp hybla: A tcp enhancement for heterogeneous networks. *International Journal of Satellite Communications and Networking*, 22(5):547–566, 2004.

[8] L. Cottrell, H. Bullot, and R. Hughes-Jones. Evaluation of advanced tcp stacks on fast long-distance production networks. Presentation at SLAC, Stanford, February 2004.

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

[10] S. Floyd. A proposed modification to tcp's window increase algorithm. unpublished draft, cited for acknowledgement purposes only, august 1994. Unpublished Draft, August 1994.

[11] S. Floyd. A report on recent developments in tcp congestion control. *IEEE Communications*, 39(4):84–90, 2001.

[12] S. Floyd. Metrics for the evalutaion of congestion control mechanisms. Internet - draft, Internet Egnineering Task Force, 2005.

[13] S. Floyd and V. Jacobson. On traffic phase effects in packet-switched gateways. *ACM SIGCOMM Computer Communication Review*, 21(2):26–42, 1991.

[14] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 3(3):115–156, September 1993.

[15] S. Floyd and E. Kholer. Tools for the evaluation of simulation and testbed scenarios. Internet draft, Internet Engineering Task Force, 2005.

[16] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and C. Diot. Packet-level traffic measurement from the sprint ip backbone. *IEEE Network, to appear*, 2003.

[17] R. Gibbens and F. Kelly. Resource pricing and the evolution of congestion control, 1998.

[18] T. Henderson. *Networking Over Next-Generation Satellite Systems*. PhD thesis, University of California, Berkeley, 1999.

[19] T. R. Henderson and R. H. Katz. Transport protocols for internet-compatible satellite networks. *IEEE Journal on Selected Areas in Communications*, 17(2):345–359, 1999.

[20] V. Jacobson. Congestion avoidance and control. In *Proceedings of ACM SIGCOM '88*, pages 314–329. ACM Press, 1988.

[21] R. Jain. A delay based approach for congestion avoidance in interconnected heterogeneous computer networks. *Computer Communications Review, ACM SIGCOMM*, pages 56–71, 1989.

[22] R. Jain, D. Chiu, and W. Hawe. A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. Technical Report TR-301, DEC Research Labs, 1984.

[23] H. Jiang and C. Dovrolis. Passive estimation of tcp round-trip times, 2002.

[24] C. Jin, D. Wei, and S. Low. Fast tcp: Motivation, architecture, algorithms, performance. In *Proceedings of IEEE INFOCOM*, March 2004.

[25] 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*, 19(1):4–11, February 2005.

[26] R. Johari and D. K. H. Tan. End-to-end congestion control for the internet: Delays and stability. *IEEE/ACM Trans. Netw.*, 9(6):818–832, 2001.

[27] Z. W. Jon. Eliminating periodic packet losses in the 4.3-tahoe bsd tcp congestion control algorithm.

[28] F. Kelly. Mathematical modelling of the internet. In *Bjorn Engquist and Wilfried Schmid (Eds.), Mathematics Unlimited – 2001 and Beyond@ Springer*. Academic Press, 2001.

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

[30] 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*, volume 49, 1998.

[31] 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*, 49:237–252, 1998.

[32] E. Kholer and S. Floyd. Internet needs better models.

[33] S. Low. Tcp fast web page, November 2005.

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

[35] S. Low, L. Peterson, and L. Wang. Understanding vegas: A duality model. *Journal of ACM*, 49(207-235), 2002.

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

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

[38] M. Malek-Zavarei and M. Jamshidi. *Time-Delay Systems*. Hardbound, 1987.

[39] J. Martin, A. A. Nilsson, and I. Rhee. Delay-based congestion avoidance for tcp. *IEEE/ACM Transactions on Networking*, 11(No. 3):356–369, June 2003.

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

[41] M. Mathis, J. Semke, and J. Mahdavi. The macroscopic behavior of the tcp congestion avoidance algorithm. *Computer Communications Review*, 27(3), 1997.

[42] 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*, pages 151–160, New York, NY, USA, 2000. ACM PRESS.

[43] 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*, pages 313–314, 1998.

[44] 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*, 8(2):133–145, April 2000.

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

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

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

[48] J. Postel. Rfc0793: Transmission control protocol. Technical report, Internet Engineering Task Force (IETF), 1981.

[49] R. S. Prasad, M. Jain, and C. Dovrolis. On the effectiveness of delay-based congestion avoidance. In *In Proceedings of Second International Workshop on Protocols for Fast Long-Distance Networks*, 2004.

[50] T. V. Project. *The Network Simulator - ns-2*.

[51] I. Rhee. Ic tcp — a tcp variant for high-speed long distance networks, November 2005.

[52] 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.

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

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

[55] G. Vinnicombe. Robust congestion control for the internet. Submitted to SIGCOM 02, 2002.

[56] D. X. Wei and P. C. annd Steven H. Low. Time for a tcp benchmark suite? Caltech Technical Reports.

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

APPENDIX

A. APPENDIX I

A.1 Dynamic Analysis

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

$$\frac{dx_r(t)}{dt} = \frac{x_r(t-\tilde{T}_r)}{\tilde{T}_r}\left(\frac{a_r}{x_r(t)\tilde{T}_r}(1-\lambda_r(t)) - b_r x_r(t)\tilde{T}_r\lambda_r(t)\right) \tag{20}$$

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:

$$\tilde{x}_r = \frac{1}{\tilde{T}_r}\sqrt{\frac{a_r}{b_r}\frac{1-\tilde{\lambda}_r}{\tilde{\lambda}_r}} \tag{21}$$

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:

$$\begin{cases} \dot{\underline{x}} = f(\underline{x}, \underline{u}) \\ \underline{y} = g(\underline{x}, \underline{u}) \end{cases} \tag{22}$$

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 (20) around its stable point, in order to express it as (for a strictly causal system):

$$\begin{cases} \dot{\underline{x}} = \underline{\underline{A}}\,\underline{x} + \underline{\underline{B}}\,\underline{u} \\ \underline{y} = \underline{\underline{C}}\,\underline{x} \end{cases} \tag{23}$$

In (20) we may identify $x_r(t)$ and $x_r(t-T_r)$ as state variables [38], and $\lambda_r(t)$ as the input to the system. We linearize (20) around its stable point in order to study the behavior of the system in its neighborhood. In equations (24) to (26) we show the results of computing the derivatives of (20) with respect to the state and input variables, and we substitute the value of (21) for $x_r(t)$ and $x_r(t-\tilde{T}_r)$.

$$\left(\frac{df}{dx_r(t)}\right)_{eq} = -2\frac{b_r\tilde{\lambda}_r}{\tilde{T}_r}\sqrt{\frac{a_r}{b_r}\left(\frac{1-\tilde{\lambda}_r}{\tilde{\lambda}_r}\right)} \tag{24}$$

$$\left(\frac{df}{dx_r(t-\tilde{T}_r)}\right)_{eq} = 0 \tag{25}$$

$$\left(\frac{df}{d\lambda_r(t)}\right)_{eq} = -\frac{a_r}{\tilde{T}_r^2\tilde{\lambda}_r} \tag{26}$$

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:

$$\underline{\underline{A}} = diag\left\{-2\frac{b_r\tilde{\lambda}_r}{\tilde{T}_r}\sqrt{\frac{a_r}{b_r}\left(\frac{1-\tilde{\lambda}_r}{\tilde{\lambda}_r}\right)}\right\} \tag{27}$$

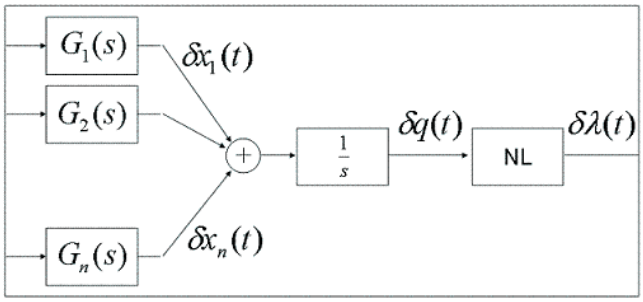

Figure 20: ***Fluid flow model of a network with n competing flows over a single bottleneck link. We have used this modeling approach in our time-continuos simulations in Simulink. In particular, we derived the $\tilde{\lambda}$, $\tilde{T}$, $\tilde{x}$ values from NS-2 long run simulations and studied the behavior of the fluid flow system for little perturbatins. The Simulink results are consistent to the NS-2 results and to the modeling intuition.***

$$\underline{\underline{B}} = diag\left\{-\frac{a_r}{\tilde{T}_r^2\tilde{\lambda}_r}\right\} \tag{28}$$

The linearized form of (20), for the r-th flow, is then :

$$\delta\dot{x}_r(t) = -2\frac{b_r\tilde{\lambda_r}}{\tilde{T}_r}\sqrt{\frac{a_r}{b_r}(\frac{1-\tilde{\lambda_r}}{\tilde{\lambda_r}})}\delta x_r(t) - \frac{a_r}{\tilde{T}_r^2\tilde{\lambda}_r}\delta\lambda_r(t) \tag{29}$$

The Laplace transform of the transfer function results to be (here we have that $\underline{\underline{C}} = \underline{\underline{I}}$, the identity matrix, in (23), since the output $\underline{y}$ is equal to state variable $\underline{x}$):

$$\underline{\underline{G}}(s) = \underline{\underline{C}}(s\underline{\underline{I}} - \underline{\underline{A}})^{-1}\underline{\underline{B}} \tag{30}$$

$$= diag\left\{-\frac{a_r}{\tilde{T}_r^2\tilde{\lambda}_r}(\frac{1}{s + 2\frac{b_r\tilde{\lambda_r}}{\tilde{T}_r}\sqrt{\frac{a_r}{b_r}(\frac{1-\tilde{\lambda_r}}{\tilde{\lambda_r}})}})\right\} \tag{31}$$

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:

$$\hat{a}_r \leftarrow \frac{\alpha_r\tilde{T}_r^2}{T_0 + \tilde{T}_r}a_r \tag{32}$$

$$\hat{b}_r \leftarrow \frac{T_1}{T_0 + \tilde{T}_r}b_r \tag{33}$$

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:

$$\tilde{x}_r = \sqrt{\frac{a_r}{b_r}\frac{\alpha_r}{T_1}\frac{1-\tilde{\lambda_r}}{\tilde{\lambda_r}}} \tag{34}$$

We now have that (27) and (28) become:

$$\underline{\underline{A}} = diag\left\{-\frac{2T_1\tilde{\lambda}_r b_r}{T_0 + \tilde{T}_r}\sqrt{\frac{\alpha_r a_r}{T_1 b_r}\frac{1-\tilde{\lambda_r}}{\tilde{\lambda_r}}}\right\} \tag{35}$$

$$\underline{\underline{B}} = diag\left\{-\frac{\alpha_r a_r}{(\tilde{T}_r + T_0)\tilde{\lambda}_r}\right\} \tag{36}$$

With these transformations, recalling (31), we have that the new Laplace transform of the *r*-th system becomes:

$$G_r(s) = -\frac{\tilde{\alpha}_r a_r}{(\tilde{T}_r + T_0)\tilde{\lambda}_r}\frac{1}{s + \frac{2T_1\tilde{\lambda}_r b_r}{T_0+\tilde{T}_r}\sqrt{\frac{\alpha_r a_r}{T_1 b_r}\frac{1-\tilde{\lambda_r}}{\tilde{\lambda_r}}}} \tag{37}$$

We are at this point still missing an important part of the system, the modeling of the network portion. In the section on queue models we have have that the queue dynamics at link *l* may be modelled as:

$$\dot{q}_l(t) = (y_l(t) - c_l)^+ \tag{38}$$

from which we can derive, assuming that these quantities vary slowly in the neighborhood of their equilibrium point:

$$\delta\dot{q}_l(t) = \delta y_l(t) \tag{39}$$

In the Laplace domain we find that the above expression may be written as $\delta Q_l(s) = \delta\frac{Y_l(s)}{s}$.

The droptail buffer is a non-linear element in our system. Droptail acts as a step non-linearity, if the queuing is below a threshold, i.e. the maximum buffer capacity, then no congestion information, i.e. loss, is feedbacked to the sender. In case the queue overshoots buffer capacity, information is fed back in order to have the transmitters lower their sending rate.

B. APPENDIX II

In this Appendix we evaluate the impact of capacity estimation errors on the equilibrium solution, more precisely the value of throughput at equilibrium. We begin by linearizing the system at the equilibrium point. We recall that at equilibrium:

$$(\frac{df}{dc_r(t)})_{eq} = \frac{\tilde{\alpha'}_r(1-\tilde{\lambda}_r)}{\tilde{T}_r + T_0} \tag{40}$$

where:

$$f(x_r(t), x_r(t-\tilde{T}_r), \tilde{\lambda}_r(t)) = \tag{41}$$

(42)

$$\frac{x_r(t-\tilde{T}_r)}{\tilde{T}_r}(\frac{\tilde{\alpha}_r\tilde{T}_r}{x_r(t)(\tilde{T}_r+1)}(1-\lambda_r(t)) - \frac{2}{3}\frac{x_r(t)\tilde{T}_r\lambda_r(t)}{\tilde{T}_r+1}) \tag{43}$$

$\tilde{T}_r$, $\tilde{\lambda}_r$ are the equilibrium round-trip time and probability of loss, while T_0 is one of Libra's parameters. We also remind that $\alpha_r(t) = k_1 c_r e^{-k_2\frac{T_r(t)-T_{min}}{T_{max}-T_{min}}}$, with c_r the bottleneck capacity, $T_r(t)$ the instantaneous round-trip time, T_{min} the minimum round-trip time and T_{max} the maximum round-trip time seen by the *r*-th flow, and k_1 and k_2 Libra's parameters. Since we are assuming that c_r isn't

anymore set to the bottleneck capacity, due to an error in estimation, we can then set $\alpha_r(t) = \alpha_r^{'}(t)c(t)$. We assume the equilibrium values $\tilde{T}_r$, $\tilde{\lambda}_r$, $\tilde{x}_r$ and $\tilde{\alpha}_r$ to be the same as before and that the perturbations in capacity are very small. We can now write the linearized model, with the superposition of the second input $c_r(t) = \tilde{c}_r + \delta c_r(t)$:

$$\delta\dot{x}_r(t) = -2\frac{b_r\tilde{\lambda_r}}{T_0+\tilde{T_r}}\sqrt{\frac{\tilde{\alpha}_r a_r}{T_1 b_r}(\frac{1-\tilde{\lambda_r}}{\tilde{\lambda_r}})}\delta x_r(t) - \frac{\tilde{\alpha}_r a_r}{(T_0+\tilde{T_r})\tilde{\lambda}_r}\delta\lambda_r(t) + \frac{a_r\tilde{\alpha'}_r(1-\tilde{\lambda}_r)}{\tilde{T_r}+T_0}\delta c(t) \quad (44)$$

we now have one more input to the system. The new input is not caused by feedback; it is a disturbance applied directly to the system's input. We want to estimate the impact on throughput. Using Laplace tranforms, we have:

$$\frac{X_r(s)}{C_r(s)}_{\Lambda(s)=0} = \frac{\frac{\tilde{\alpha'}_r(1-\tilde{\lambda}_r)}{\tilde{T_r}+T_0}}{s + 2\frac{b_r\tilde{\lambda_r}}{T_0+\tilde{T_r}}\sqrt{\frac{\tilde{\alpha}_r a_r}{T_1 b_r}(\frac{1-\tilde{\lambda_r}}{\tilde{\lambda_r}})}} \quad (45)$$

$$= \frac{\tilde{\alpha'}_r(1-\tilde{\lambda}_r)}{(\tilde{T_r}+T_0)s + 2b_r\tilde{\lambda_r}\sqrt{\frac{\tilde{\alpha}_r a_r}{T_1 b_r}(\frac{1-\tilde{\lambda_r}}{\tilde{\lambda_r}})}} \quad (46)$$

We assume capacity is measured off line, before the start of the TCP session, or during the slow start phase. Thus the error is constant throughout the life of the connection. The input disturbance may be written as $\delta c(t) = \hat{c}$. In the Laplace domain this becomes $\hat{C}(s) = \frac{\hat{c}}{s}$. To evaluate the impact on throughput $x_r(t)$, we set $x_r(t) = \tilde{x}_r + x_r'(t)$, and derive:

$$lim_{t\to\infty}x_r'(t) = lim_{s\to 0}sX_r'(s) \quad (47)$$

$$= lim_{s\to 0}s\frac{\hat{c}}{s}\frac{\tilde{\alpha'}_r(1-\tilde{\lambda}_r)}{(\tilde{T_r}+T_0)s + 2b_r\tilde{\lambda_r}\sqrt{\frac{\tilde{\alpha}_r a_r}{T_1 b_r}(\frac{1-\tilde{\lambda_r}}{\tilde{\lambda_r}})}} \quad (48)$$

$$= \frac{\tilde{\alpha'}_r(1-\tilde{\lambda}_r)\hat{c}}{2b_r\tilde{\lambda_r}\sqrt{\frac{\tilde{\alpha}_r a_r}{T_1 b_r}(\frac{1-\tilde{\lambda_r}}{\tilde{\lambda_r}})}} \quad (49)$$

$$= \frac{\tilde{\alpha'}_r(1-\tilde{\lambda}_r)\hat{c}}{2b_r\tilde{\lambda_r}\tilde{x}_r} \quad (50)$$

$$= \frac{\tilde{\alpha'}_r(1-\tilde{\lambda}_r)\hat{c}}{2b_r\tilde{\lambda_r}\tilde{x}_r}\frac{\tilde{C}_r}{\tilde{C}_r} \quad (51)$$

$$= \frac{1}{2}\frac{\tilde{x}_r^2\hat{c}}{\tilde{x}_r\tilde{C}_r} \quad (52)$$

$$= \frac{1}{2}\gamma\tilde{x}_r \quad (53)$$

The percentage error, with respect to $\tilde{x}_r$ is given by :

$$Relative\ error = \frac{Absolute\ error}{\tilde{x}_r} = \frac{1}{2}\gamma \quad (54)$$

The system wont dump the error to zero, but will alleviate its effect on the output. This is an acceptable behavior, considering we are assuming the capacity percentage estimation error small. This error depends strongly on the estimation method that is implemented. More work has to be done on the understanding of the accuracy of the existent estimation techniques and on understanding how the system will vary for greater percentage errors in estimates.