# Balancing Video on Demand Flows over Links with Heterogeneous Delays

Gustavo Marfia[1], Claudio E. Palazzi[2], Giovanni Pau[1],
Mario Gerla[1], Medy Y. Sanadidi[1], Marco Roccetti[2]

[1] Computer Science Department,
University of California Los Angeles,
Boelter Hall, Los Angeles, CA, 90095, USA
+1-310-825-4367
{gmarfia, gpau, gerla, medy}@cs.ucla.edu

[2] Dipartimento di Scienze dell'Informazione,
Università di Bologna,
Mura Anteo Zamboni 7, 40127 Bologna, Italia
+39 051 - 209 45 03
{cpalazzi, roccetti}@cs.unibo.it

## ABSTRACT
The popularity of Video on Demand (VoD) services has recently grown to unprecedented levels. Even if UDP is often considered the standard transport protocol for video streaming, TCP is often used for VoD since its reliability, congestion control, and resilience to the presence of firewalls/NATs on the link. Unfortunately, poor video quality, frequent playback pauses, and delays due to slow frame buffering are still annoying users engaged in long RTT connections, both wired and wireless, with the server. This is due to the ack-based mechanism that increases the TCP's congestion window, which leads to RTT-unfairness. As a practical consequence, if a VoD user is experiencing long RTTs while sharing the channel with another VoD user whose connection has small RTTs, the former will see a very slow progression of its video until the latter is done. In this paper, we propose the use of TCP Libra on VoD servers to resolve this RTT-unfairness issue, thus providing an efficient VoD service to any user, regardless of her/his RTTs.

## Keywords
Video on Demand, TCP, RTT-fairness.

## 1. INTRODUCTION
The popularity of Video on Demand (VoD) services, such as Google Video and YouTube, has recently reached unprecedented levels with a growing trend whose end is still not in sight. Needless to say, their domain also extended on the wireless realm, thus having VoD services enjoyed also on portable devices (e.g., PDAs, cellphones). Indeed, with the wide diffusion of handheld devices able to connect to the Internet and to show multimedia contents, it is easy to foresee an immediate future where mobile users will represent the large majority of VoD consumers.


MOBIMEDIA 2007, August 27-29, Nafpaktos, Greece

DOI 10.4108/ICST.MOBIMEDIA2007.1760

In this context, although UDP is often considered the standard transport protocol for video streaming services, commercial VoD services and personal video sharing on the web generally exploit TCP. This preference is generally due to TCP's reliability, congestion control mechanism, and resilience to the presence of firewalls/NATs between client and server.

Unfortunately, a well known (negative) property of TCP is represented by its *RTT-unfairness* [1]. Simply stated, the sending rate (i.e., the congestion window) of a TCP session increases upon receiving acks of sent packets. Therefore, the shortest the path between the server and the client is, the more acks per time unit will be received and the faster the sending rate will increase. As a consequence, TCP sessions with small RTTs will open their congestion windows faster than those with large RTTs, thus capturing most of the available bandwidth.

From a VoD point of view, this has practical implications both in case of downloading a video and streaming it. In the first scenario, users experiencing large RTTs on their connections are forced to wait endlessly to have their video downloaded (at a snail's pace), while users closer to the VoD server occupy the whole bandwidth available, downloading one or more high quality videos. With the second scenario, users on long RTT links will be able to receive only very low quality streams as most of the bandwidth is already utilized by very high quality streams on small RTT sessions.

Needless to say, this problem affects both wired and wireless connections; however, in the latter case, its negative effects are exacerbated by the combination with other wireless issues such as error losses, channel capture effects, and mobility [2] [3].

As an empirical demonstration, Fig. 1 sheds light on the entity of the RTT-unfairness problem for VoD services utilized by generic mobile users. Specifically, we have set two simultaneous downloads of a file video of 26MB. Both connections shared the same wireless bottleneck link of 1Mbps, a value comparable with domestic DSL connectivity. One of the download was from Los Angeles, US, to the same city; whereas the other one was between Los Angeles and Taipei, Taiwan. In both cases, the last hop was a standard IEEE 802.11b wireless link. The unfair bandwidth utilization of the two simultaneous video downloads is evident in Fig. 1. Indeed, even if sharing the same bottleneck, the two downloading sessions achieve very different link utilizations.

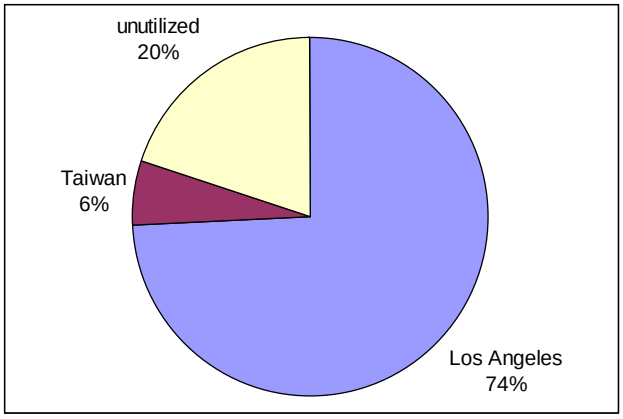


**Figure 1. Unfair utilization of the available wireless bandwidth due to different RTTs.**

This unfairness among users, whether they are paying or not for the VoD service, is clearly not acceptable. First, because users would be very disappointed by the service and refrain from using it again, and second, because VoD service provided would be forced to renounce to possible market places, or to deploy servers all around the world instead of having them located at their own convenience.

To solve this problem we addressed the real cause at its base: the TCP's RTT-unfairness. We have hence designed TCP Libra which employs a different congestion avoidance scheme to ensure RTT-fairness, while still guaranteeing bandwidth efficiency and friendliness to other TCP flows. Therefore, TCP Libra could be used on VoD servers, bringing RTT-fairness to users connected to that service, without affecting any other TCP session in the Internet. Moreover, TCP Libra requires modifications only at sender side (i.e., the server); this feature makes it very easy to be deployed as only VoD service providers would need to implement it and they will be willing to do it in order to provide a better service to their customers. Instead, VoD service users will just utilize their standard devices (e.g., laptop, PDA, smart cellphone) with standard communication capabilities/protocols and enjoy the RTT-fair service.

The rest of the paper is organized as follows. Section 2 reviews the scientific literature that embodies the background for our work. In Section 3 and Section 4 we present TCP Libra through its high level design and detailed algorithm, respectively. The experimental environment is described in Section 5, whereas results are discussed in Section 6. Finally, Section 7 concludes the paper.

## 2. TCP FOR VOD

The general belief that TCP were unsuitable for applications such as VoD has been proven wrong both by real services actually available through the Internet and by several scientific papers on this topic. For instance, [4] elaborates on how to exploit client-side buffering in multimedia streaming applications to address TCP's retransmission delays and throughput variance due to congestion control.

As an example of work in this area, a practical approach to have TCP transmitting packets in a CBR-like fashion is presented in [5]; yet, this solution unfeasibly requires feedback from routers.

Recently, [6] proposed modifications to the congestion avoidance scheme on the server aimed at stabilizing the TCP throughput around a target rate. However, the authors assumed to deal with metropolitan VoD services with very small RTTs for all the flows in their experiments, without addressing the RTT-unfairness problem.

To the best of our knowledge, [7] is the only work that considered the RTT-unfairness problem in the context of providing VoD services. The authors designed a receiver-based bandwidth sharing system for allocating the capacity of the last mile bottleneck among TCP-supported video flows according to user's preferences. Yet, the proposed mechanism is run at the receiver side thus requiring to be mounted on all the clients exploiting a certain bottleneck in order to work.

Instead, our approach is designed to run server-side. This way, any VoD provider could adopt it on its servers to enable a more fair and efficient service for all of its customers, that can hence employ the regular TCP available on their operating systems.

## 3. TCP LIBRA'S COMPONENTS

TCP Libra has been designed with the goal of being independent from RTT. It hence represents an excellent candidate to support VoD services over heterogeneous delay scenarios. Yet, any new transport protocol that will be utilized over the Internet has to take into account a lot of issues including the availability of new high speed links, the compatibility with the existing architecture and protocols, and the general end-to-end philosophy that have served so well till our days. To this aim, any proposal for modifying TCP cannot be focused on addressing just a single issue; rather, it has to follow a holistic approach that considers the performance that the new protocol will achieve within the complex Internet scenario and how the protocol will interact with existing standards.

Coherently, not only we designed TCP Libra to achieve our initial goal of RTT-unfairness, but we also added components to improve efficiency and preserve friendliness with legacy protocols. Interested readers may refer to [8] for a detailed description and analysis of TCP Libra's components. Here, for the sake of conciseness, we limit our description to the basics of the algorithm.

The main components of TCP Libra are:

1. *Fairness control*. The fairness control is the core component of TCP Libra and implements its fairness functionality by equalizing the throughput of heterogeneous RTT flows. This component takes inspiration from Floyd and Jacobson's seminal work [1]. We extend that work by adding a component that lowers the throughput variance.

2. *Capacity estimator*. This component is in charge of estimating the capacity of the bottleneck link at the beginning of a new session. The capacity of the bottleneck can be determined through an off line tool, such as CapProbe or Path Rate [9] [10], or through a mechanism that is embedded in the

TCP protocol itself, such as TCP Probe does [11]. Extensive simulation and testbed experiments of all the above schemes have shown an accuracy well within 10% in a fraction of a second. This is perfectly adequate for our purposes as, generally, the capacity is something that our algorithm needs to calculate only at the beginning of a VoD stream or download.

3. *Scalability control*. The scalability control receives in input the capacity of the bottleneck link from the capacity estimator and sets the slope of the window increase proportional to this value. We can intuitively justify this choice by observing that a larger capacity of the bottleneck link requires a greater speed to converge to the fair share; this is a choice that enables TCP Libra to scale on links of any magnitude.

4. *Stability control*. The stability control makes sure that, taking as an input the share of buffer occupancy, the protocol operates in its stability region, i.e., that interval where the expected characteristics of the protocol hold and the throughput average behaves as in the linear model [8]. 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 are factually sent, making sure that the network is not injected with a heavy burst of traffic all at once, especially when trying to scale at high bandwidth speed. Previous studies demonstrated how pacing has a de-synchronizing effect that leads to higher efficiency in steady state [12]. In TCP Libra we have implemented a randomized pacing strategy on the packet transmission functionality in order to prevent synchronized losses and multiple reductions of the TCP's window among concurrent flows [8].

In the following section we present the TCP Libra algorithm, highlighting the various components we just named and the rationale behind their functioning.

# 4. TCP LIBRA ALGORITHM

To permit a factual deployment of TCP Libra, we have designed it to be as compatible as possible with the *de facto* standard in the Internet architecture. To this aim, we have forced ourselves to limit the modifications required to implement it only at the TCP's sender side of a connection (i.e., VoD servers). Moreover, only TCP's congestion control algorithm is modified and in such a way that it still falls into the class of the AIMD algorithms, thus preserving fairness and friendliness properties towards other simultaneous TCP flows [13]. However, TCP Libra modifies both the way the window is increased after successfully receiving an ack and the window reduction in case of a packet loss (by using a *variable* multiplicative factor).

More in detail, the pseudocode fort TCP Libra's algorithm for the congestion control phase is reported in Fig. 2; regular TCP's congestion control algorithm can be found in [14].

In Fig. 2, $window_t$ and $RTT_t$ are, respectively, the congestion window size and the RTT, at time *t*. Instead, *A* and *B* are fixed parameters, whereas $\alpha_t$ is defined by (1) and (2).

$$\alpha_t = \mathrm{X} \cdot C \cdot e^{-\mathrm{Y} \cdot \beta_t} \quad (1)$$

$$\beta_t = \frac{RTT_t - RTT_{\min}}{RTT_{\max} - RTT_{\min}} \quad (2)$$

In (1) and (2), *X* and *Y* are fixed parameters, *C* is the total capacity of the bottleneck link, $RTT_{max}$ and $RTT_{min}$ are the maximum and the minimum RTTs experienced during the connection up to time *t*.

*if ( an ack is successfully received ) then*

$$window_{t+1} \leftarrow window_t + \frac{1}{window_t} \cdot \frac{\alpha_t \cdot RTT_t^2}{RTT_t + \mathrm{A}}$$

*else if ( three dupacks are received ) then*

$$window_{t+1} \leftarrow window_t - \frac{window_t \cdot \mathrm{B}}{2 \cdot (RTT_t + \mathrm{A})}$$

$$ssthresh_{t+1} \leftarrow window_{t+1}$$

*else if ( timeout expires ) then*

$$ssthresh_{t+1} \leftarrow \frac{window_t}{2}$$

$$window_{t+1} \leftarrow 1$$

*end if*

**Figure 2. TCP Libra's congestion control.**

## 4.1 The Rationale of the Algorithm

We can now map TCP Libra's architectural components discussed in Section 3 at an algorithmic level. However, as it is out of the scope of this paper to provide a detailed explanation of the setting of TCP Libra parameters (i.e., *A*, *B*, *X*, *Y*), for the sake of conciseness we just provide fundamental information to understand the functioning of the protocol and its suitability for the considered scenario of a VoD service over wired/wireless links with heterogeneous delays. However, interested readers can find a comprehensive discussion of TCP Libra design and configuration details for its general use in [8].

1. *Fairness control*. The fairness control component is the core of our algorithm and is implemented through the terms $RTT_t^2/(RTT_t + \mathrm{A})$ and $B/(RTT_t + \mathrm{A})$ of the algorithm. The former is utilized during the increase of the congestion window, when an ack has been successfully received by the sender. A similar term was already conjectured by Floyd and Jacobson in [1] to be able to provide RTT-fairness. Through analytical investigation we extended that work by adding a component that lowers the throughput variance [8]. The aim is that of penalizing those flows whose RTT exceeds a certain

threshold as they are surely experiencing severe congestion problems on the path. Indeed, in those cases where $RTT_t << \mathrm{A}$ then $\frac{RTT_t^2}{RTT_t + \mathrm{A}} \cong \frac{RTT_t^2}{\mathrm{A}}$ whereas when $RTT_t \approx \mathrm{A}$ then $\frac{RTT_t^2}{RTT_t + \mathrm{A}} \cong \frac{RTT_t}{2}$. Instead, the term $B/(RTT_t + \mathrm{A})$ gives us control over the range of RTTs for which we are interested in equalizing the throughput.

2. *Capacity estimator*. As already discussed, the capacity of the bottleneck can be precisely and quickly determined through an off line tool or an embedded mechanism within the protocol [9] [10] [11]. In our simulation experiments, the capacity was known in advance, of course. Work is now in progress to incorporate the capacity probing feature (using packet pair techniques) directly into TCP Libra. Clearly, this requires TCP packets to be sent in group of two, one immediately after the other.

3. *Scalability control*. The scalability control is embodied by the term $X \cdot C$ in (1). Thanks to it, TCP Libra is able to scale its window increase rate proportionally to the bottleneck capacity: the slope of the window increase is set proportionally to the capacity value in Mbps. The parameter $X$ is fixed and must be set taking into account the algorithm's requirement for responsiveness, as well as its necessity to operate in its stability region. Simply stated, a higher value of $X$ makes the algorithm more aggressive.

4. *Stability control*. The stability control component is represented by $e^{-\mathrm{Y} \cdot \beta_t}$. This control function ensures that the window increase rate slows down when the utilized links become congested. Indeed, the exponent $\beta_t$ may be interpreted as the utilized share of the buffer: the more its value approaches unity, the slower the increase rate of the congestion window becomes. The parameter $Y$ is a constant that sets the responsiveness to queue build up. To a greater value of $Y$ corresponds a larger stability region for TCP Libra and a more efficient utilization of its share of the available bandwidth, but also a higher throughput loss when competing against regular TCP flows.

5. *Burstiness control*. This component cannot be seen in Fig. 2 since it is not part of the window update algorithm, yet, it is an essential element of TCP Libra as it determines the dispersion of sent packets. In particular, the burstiness control has been designed with the objective of avoiding two specific problems: synchronization of loss events and failure to collect significant RTT samples. We have implemented a cyclic pacing scheme that also include randomness. Specifically, packet pairs (to enable capacity estimation) are randomly sent within a round-trip time, in such a way that they result statistically uniformly distributed. To this aim, the RTT is divided into as many intervals as half of the size in packets of the TCP's window and packets (in pairs) are sent at a time that is randomly chosen within each interval. This behaviour smoothens possible side effects related to the scalability control component by making sure that the network is not injected with a heavy burst of traffic all at once.

# 5. EXPERIMENTAL ASSESSMENT

We evaluated the ability of TCP Libra in providing a RTT-fair, efficient, and (legacy) TCP-friendly support to VoD service through the well known NS-2 simulation platform [15]. We generated the scenario depicted in Fig. 3 where short, medium, and long RTT flows are simultaneously present and where regular TCP (i.e., TCP New Reno) and TCP Libra can be utilized as the only transport protocol in the system, or together. Provided results represents average values obtained from 30 simulation runs for each configuration. Furthermore, as VoD downloading/streaming generally lasts for several minutes we have run each of our experiments for 1000s.

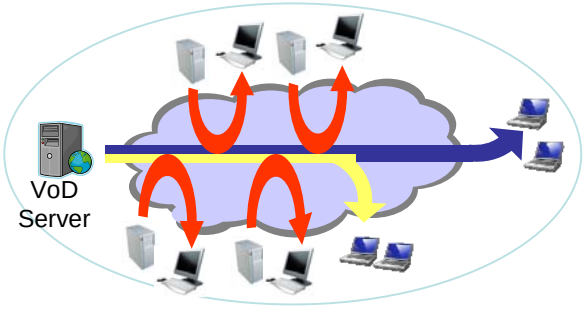


**Figure 3. Simulative Scenario.**

More in detail, a series of bottleneck links of 100Mbps is shared by two long-RTT (180ms, blue arrow in Fig. 3) and two medium-RTT (90ms, yellow arrow in Fig. 3) flows in parallel, plus other four short-RTT (30ms, red arrows in Fig. 3) flows, which do not share any links among each other. Buffers on routers along the path have been set alternatively as the pipe size corresponding to a single bottleneck link (83 packets) or to the longest overall connection (1500 packets). The actual value on commercial routers is generally something in between these two values.

Finally, parameters mentioned in Section 4 are set as follows: $A = 1$, $B = 1$, $X = 2$, $Y = 2$; interested readers may refer to [8] for a wider discussion about parameter setting.

# 6. MEASURED PERFORMANCE

The most widely adopted metric to evaluate fairness is Jain's index [16]; we have hence adopted this index to test the RTT-fairness of our protocol. The more this index approaches to 1, the more the evaluated protocol is considered to be fair. To this aim Fig. 4 and Fig. 5 show the Jain's index values achieved by regular TCP New Reno (*TCP*) and TCP Libra (*Libra*). In particular the Fig. 4 considers only medium and long RTT flows among those shown in Fig. 3, whereas Fig. 5 calculates the index by including all flows, also the small RTT ones. Moreover, in each of the charts, results are shown both for the case where buffers were set with a size in packets equal to the pipe size of a single bottleneck link (*Bottl Pipe*) or of the longest overall connection (*Tot Pipe*). As it is evident, TCP Libra outperforms regular TCP in achieving fairness, regardless of the RTT variety. Moreover, when excluding flows that have a huge difference of RTT from the others, i.e., those with 30ms of RTT, the Jain's index of TCP Libra is almost optimal.

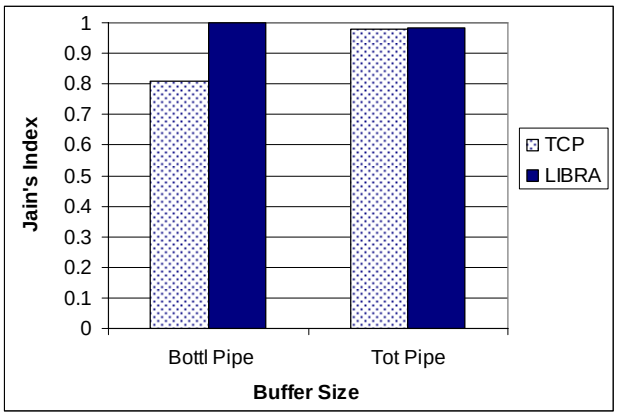


**Figure 4. Jain's fairness index calculated only on medium and long RTT flows.**

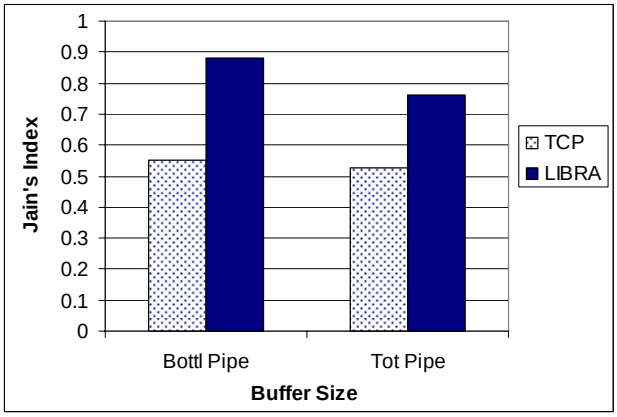


**Figure 5. Jain's fairness index calculated on all flows.**

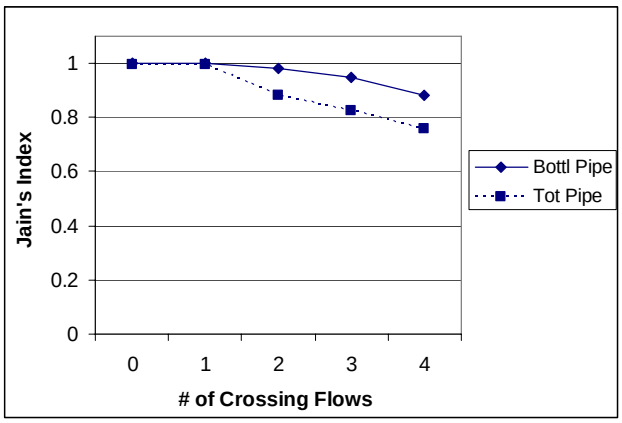


**Figure 6. Jain's fairness index for TCP Libra with different cross traffic scenarios.**

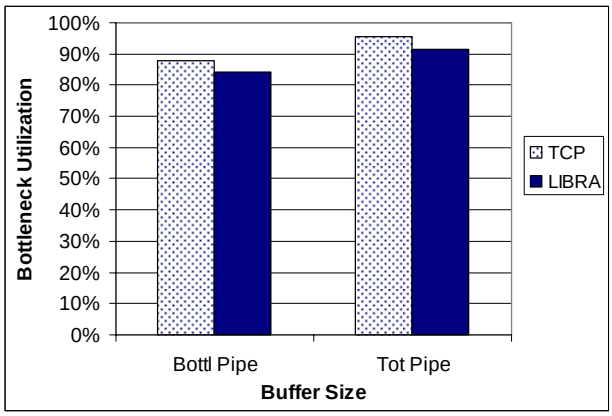


**Figure 7. Measured efficiency.**

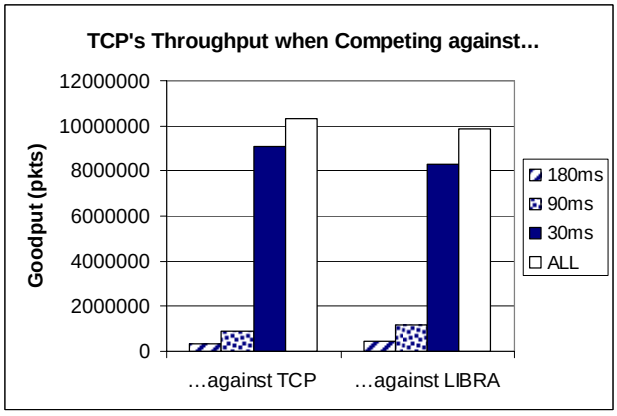


**Figure 8. Friendliness study: packets sent by regular TCP flows when competing against other TCP flows (left columns) or against Libra flows (right columns).**

For a deeper analysis of these results Fig. 6 presents values of the Jain's index as achieved by TCP Libra when the medium and long RTT flows are competing with 0, 1, 2, 3, or all 4 small RTT flows in the considered simulative scenario. Clearly, the more complex the scenario is, the harder it becomes for TCP Libra to achieve an optimal fairness. Still, results are encouraging and definitely better than those achievable by employing regular TCP; for a comparison, consider that values for the case with 4 competing small RTT flows in Fig. 6 corresponds to TCP Libra's values in Fig. 5.

Needless to say, if TCP Libra were able to provide RTT-fairness at the cost of reducing the total throughput achieved, that could not be fully considered a positive result. Conversely, as it is demonstrated by Fig. 7, TCP Libra's utilization of the available bandwidth results comparable with that of regular TCP with both buffer size configurations. Therefore, TCP Libra achieves RTT-fairness through rebalancing bandwidth resources among the various flows.

Finally, as the VoD service is deployed over the Internet and we want TCP Libra to support it, then it is crucial for TCP Libra to be able to coexist with the *de facto* standard of Internet protocols. To this aim, Fig. 8 assesses TCP Libra's friendliness toward regular TCP. It is very important to notice that Fig. 8 does not represent a fairness study; rather, it just is a study on the friendliness of TCP Libra toward regular TCP.

Specifically, we have divided flows depicted in Fig. 3 into two groups. Then we consider two cases; however, in both cases, values in Fig. 8 represents the goodput achieved by the first group of flows, which always employs regular TCP, in terms of total number of packets delivered to destination for each class of flow (small, medium, and long RTT) plus the sum of them (*ALL* in Fig. 8).

In the first case, we have regular TCP implemented in all flows of both groups and we measure the average goodput experienced by the various flows; results grouped by RTT are reported in the left columns of Fig. 8 (*TCP*). In the second case, one group of flows utilizes regular TCP whereas flows of the other group use TCP Libra. Even in this case, we measured the average goodput only of regular TCP flows so as to see how much their goodput was affected by the simultaneous presence of TCP Libra's flows. Results reported on the right columns (Libra) of Fig. 8 show that the goodput achieved by regular TCP's flows is not significantly affected by sharing the link with TCP Libra's flows. We can hence claim that not only is TCP Libra RTT-fair, but it also is friendly toward legacy TCP.

As it is evident, the utilization of TCP Libra in place of regular TCP on half of the flows does not significantly impact on the goodput achieved by the other half of flows that employs regular TCP. This clearly demonstrates TCP Libra's friendliness toward regular TCP. The network configuration utilized in Fig. 8 employed buffer sizes equal to the pipe size of one bottleneck link. However, analogous results were obtained even with larger buffer sizes.

## 7. CONCLUSION

We discussed the RTT-unfairness problem that arises when trying to provide VoD services through TCP. This problem exists for both wired and wireless connections, however, in the latter case its negative effects are worsened by the simultaneous presence of typical wireless issues such as error losses, channel capture, and mobility To this aim, we designed TCP Libra, a RTT-fair transport protocol that is also efficient in utilizing the available bandwidth and friendly toward legacy TCP.

Extensive simulation results demonstrated the ability of our protocol in being RTT-fair and efficient, while still being friendly toward simultaneous legacy TCP flows. Through it, VoD services can be hence more efficiently provided, regardless of the distance between client and server.

## 8. ACKNOWLEDGMENTS

Partial financial support for this work is provided by: the Italian MIUR (ICTP/E-Grid, MOMA, DAMASCO) and Ministero Affari Esteri (Laboratorio Congiunto); NSF (Grant No. 0520332); and STMicroelectronics (UC-Micro Grant MICRO 04-05).

## 9. REFERENCES

[1] S. Floyd, V. Jacobson. On Traffic Phase Effects in Packet-switched Gateways. *ACM SIGCOMM Computer Communication Review*, 21(2):26-42, 1991.

[2] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, R. H. Katz. A Comparison of Mechanisms for Improving TCP Performance over Wireless Links. *IEEE/ACM Transactions on Networking*, 5(6):756 – 769, 1997.

[3] K. Xu, M. Gerla, L. Qi, Y. Shu. Enhancing TCP fairness in ad hoc wireless networks using neighborhood RED. In *Proc. of MOBICOM'03*, San Diego, California, USA, pp. 16-28.

[4] Charles Krasic, Kang Li, Jonathan Wapole. The Case for Streaming Multimedia with TCP. In *Proc. of 8th International Workshop on Interactive Distributed Multimedia Systems (IDMS 2001)*, Lancaster, UK, 2001.

[5] P.-H. Hsiao, H.T. Kung, K.-S. Tan. Video over TCP with Receiver-based Delay Control. In *Proc. of ACM NOSSDAV 2001*, Port Jefferson, NY, USA, 2001, pp. 199-208.

[6] H. Shimonishi, T. Hama, T. Murase. TCP Congestion Control Enhancements for Streaming Media. In *Proc. of IEEE CCNC 2007*, Las Vegas, NV, USA, 2007.

[7] P. Mehra, C. De Vleeschouwer, A. Zakhor. Receiver-Driven Bandwidth Sharing for TCP and its Application to Video Streaming. *IEEE Trans. on Multimedia*, 7(4):740-752, 2005.

[8] G. Marfia, C. Palazzi, G. Pau, M. Gerla, M. Y. Sanadidi, M. Roccetti. *TCP Libra: Exploring RTT-fairness for TCP*. Technical Report UCLA-CSD TR-050037, UCLA, Computer Science Dept., Los Angeles, CA, 2005.

[9] R. Kapoor, L.-J. Chen, L. Lao, M. Gerla, M. Y. Sanadidi. Capprobe: a Simple and Accurate Capacity Estimation Technique. In *Proc. of SIGCOMM '04*, New York, NY, USA, 2004, pp. 67-78.

[10] C. Dovrolis, P. Ramanathan, D. Moore. Packet Dispersion Techniques and Capacity Estimation. *IEEE/ACM Transactions of Networking*, 12(6):963-977, 2005.

[11] C. Marcondes, A. Persson, L.-J. Chen, M. Y. Sanadidi, M. Gerla. TCP Probe: a TCP with Built-in Path Capacity Estimation. In *Proc. of the 8th IEEE Global Internet Symposium*, Miami, FL, USA, 2005.

[12] A. Aggarwal, S. Savage, T. Anderson. Understanding the Performance of TCP Pacing. In *Proc. of INFOCOM 2000*, Tel Aviv, Israel, 2000, pp. 1157-1165.

[13] Saverio Mascolo, Claudio Casetti, Mario Gerla, M. Y. Sanadidi, Ren Wang. TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links. In *Proc. of ACM SIGMOBILE 2001*, Rome, Italy, 2001, pp. 287-297.

[14] W. R. Stevens. *TCP/IP Illustrated, vol. 1*. Addison Wesley, Reading, MA, USA, 1994.

[15] The Network Simulator – ns-2. http://www.isi.edu/nsnam/ns/

[16] R. Jain, D. Chiu, W. Hawe. *A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer Systems*. Technical Report TR-301, DEC Research Labs, 1984.