Open Source Scavenging Transport Protocols

Dario Rossi
Telecom ParisTech, Paris, France
dario.rossi@enst.fr

***Abstract*—In this work we report on our experience in developing lower-than-best-effort congestion control disciplines for the TCP/IP Linux kernel stack. Specifically, we offer an open source implementation of NICE (and, as an ongoing effort, of LEDBAT), that we compare to other TCP flavors with experiments in controlled testbeds and in the wild Internet.**

I. Introduction

The growing collection of physically interconnected devices and the rapidly varying landscape of software applications are at the root of the Internet success and grandeur. Initially, the Internet was envisioned as a collection of diverse applications, globally interconnected through a common suite of protocols (i.e., TCP/IP), over a rich diversity of physical media and transmission technologies (wired, wireless, underwater, acoustic, etc.). TCP/IP was henceforth referred to as the "thin waist" of the so called "hourglass" model, providing a unifying glue between the rich upper-world of applications and under-world of physical interconnections. Truth is, over the years TCP became, over and over, a rather wide protocol family.

To testify this diversity, consider that Cubic is used by default in Linux kernels since version 2.6.19. Compound has been widely deployed with Microsoft Windows Vista and Windows Server 2008. MacOS and FreeBSD uses NewReno as the default algorithm. Yet, many more TCP congestion control flavors have been proposed over the years – even before the very same definition of novel TCP flavors was outsourced to computer programs [1]. As the network link bandwidth steadily arose in the last years, most TCP proposals targeted higher than best effort performance [2], where best effort refers to the default TCP version. A more narrow choice is instead available for what concerns protocols aiming at a *lower than best effort* priority [3].

Yet, applications that require bulky and non-interactive transfers are numerous. These not only include P2P filesharing applications, but also virtually any Cloud service requiring constant or periodic synchronization. These low-priority transfers are already largely popular nowadays: consider for instance Microsoft Background Intelligent Transfer Service (BITS), or the Picasa background transfer option, or the BitTorrent Low Extra Delay Background Transport (LEDBAT) feature. More generally, we argue that any data intensive application such as synchronization toward the Cloud, backup, etc. would benefit of low priority scavenging capabilities.

In principle, lower than best effort priority could be simply achieved by traffic shaping at the user access: generally speaking, an application can decide to throttle (e.g., with a token bucket shaper) its maximum rate to $r < C$, with C the physical capacity (due to asymmetric link capacities, uplink and downlink rates can be separately shaped). This is indeed already quite popular today, and usually done to avoid the user access link to become fully saturated, as otherwise the relatively large buffer sizes in front of relatively slow ADSL and cable connections raise problems to interactive applications – with queuing delay growing to multiple seconds, a phenomenon known as bufferbloat [4].

Yet, we argue that lower than best effort capabilities should better be offered by the Operating System (OS) as part of the common TCP/IP networking stack: indeed, application-layer bandwidth limits have three major drawbacks. First, as not all the available bandwidth is utilized, application-layer shaping causes large inefficiencies in terms of user experience. To make a point, consider for instance advised settings[1] by Vuze, a largely popular BitTorrent client. Notice that in case of C=5Mbps, Vuze suggest to set a maximum upload speed of 400kB/s (or 3.2Mbps), i.e., throttling throughput to only 64% of the available capacity. Clearly, in case these bandwidth limits were largely adopted (e.g.,hard-coded as application defaults), this would terribly slow down download for the whole swarm (as service capacity of the aggregated P2P swarm scales exponentially with capacity of individual peer[5]). Second, application-layer shaping is not sufficient to entirely solve the user bufferbloat problem: this is simply understood by considering that in case multiple applications independently set their r_i limit, nothing forbids $\sum_i r_i > C$ (which applies to both the case of multiple applications running on the same host, or on multiple hosts of the same household). Third, proprietary shaping implementations may differ or interfere with each other (e.g., one may employ a packet pacing strategy, while another may send packet bursts of packets) and they may offer few to none control.

Based on the above observation, we argue it would be a desirable practice to systematically share within the research community, not only novel ideas and results, but also their software incarnation. In this spirit, we offer to the research community[2] our open source implementations of NICE [6] (and LEDBAT [7] as part of our ongoing effort). These software implementations can be used both as Linux kernel modules or as ns2 modules (ns3 tests with nsc are ongoing). In this paper, we report on our motivation (Sec. II), on the available lower than best effort protocols (Sec. III), as on our experimental experience with scavenging protocols (Sec. IV).

[1] http://wiki.vuze.com/w/Good_settings
[2] http://www.telecom-paristech.fr/∼drossi/scavenging

II. MOTIVATION

As previously introduced, TCP proposals have been so numerous that is very cumbersome even to dress a full list [2]. Yet, if we restrict the count to those that are readily available to the savvy user as open source, (e.g., in the Linux kernel stack), this task becomes unfortunately significantly simpler.

Consider for instance one of the first low-priority TCP flavors, i.e., NICE[6]. Despite the original NICE code is still available[3], it however dates back to 2002, when Linux kernel 2.3 was a development release. It follows that the original code is, at least in our experience, non easily portable to more recent kernels.

This naturally follows from the fact that, while the research process often involves production of software prototypes, these are only rarely "products": as research software does not generate direct revenue, it does not generate the funding necessary to cover the cost of its long-term maintenance. Additionally, since research interests (and funding) shift, maintaining over time even own software packages can be a daunting task: we thus believe that sharing this burden within the community can be a more productive model, already proven by Linux and other open source initiatives. We therefore offer our implementation of the original[4] NICE design, interoperable with recent kernels (tested under 2.6 to 3.3), to the open source community.

Our own motivation in developing NICE (and LEDBAT) was to study application performance under several congestion control protocols. In particular, our focus was on BitTorrent: despite a known interplay between application-layer policies and lower-layer transport dynamics [8], it was unclear how the different flavor of the TCP stack would affect protocol performance at a finer grain. When BitTorrent replaced TCP swarming with LEDBAT, a low priority protocol implemented at the application layer over an UDP framing, it was implicitly confirming TCP to be an important piece of the puzzle. In particular, the need for LEDABT was motivated by the excessive buffering delay [4] self-inflicted by BitTorrent users – interfering with more interactive activities of the same users such as gaming, VoIP and Web browsing. While we initially resorted to the BitTorrent ns2 simulator [8] to carry on this analysis, it quickly become clear that an experimental approach would allow a more realistic performance assessment, additionally offering the ability to include popular (but closed-source) BitTorrent clients (such as uTorrent) in the comparison.

We have extensively tested the lower priority protocols implementations with both simulations and experiments, with scenarios ranging from on simple backlogged workloads to full-scale BitTorrent swarms, possibly in presence of active queue management techniques. It is out of the scope of this paper to fully report our findings: rather, we briefly share a few important highlights of our experience, and especially provide the implementations to the community[5].

III. SCAVENGING PROTOCOLS

Scavenging protocols date back to early 2000, with NICE[6] and TCP-LP[9] among the very first proposals, and LEDBAT[7] among the most recent ones. The above list is by no means exhaustive: hence, we refer the interested reader to [3] for a more complete survey of lower than best effort proposals. Similarly, we just cover the main lower than best effort protocol features, referring the reader to to [6], [9], [7] for protocol details.

A. Delay variations.

Recall that best effort TCP employs *packet losses* as primary signs of congestion: depending on the gravity of these signs (e.g, reinforcement of the loss signal via duplicated acknowledgements, or absence of explicit signals after a timeout), best effort TCP then takes appropriate reactions (e.g., halving the congestion window and recover losses through FastRecovery, or resetting the congestion to 1 segment and performing Slow-Start).

In order to avoid interfering with best effort TCP, lower than best effort protocols rely on more timely signs of congestion. Specifically, such protocols all employs *delay variations* to infer that queuing is building up, even prior that losses occur: on such early signs of congestions, lower than best effort protocols promptly shrink their congestion window, resulting in a more conservative, and consequently less intrusive, congestion control algorithm with respect to best effort TCP.

B. OWD vs RTT delay

Early congestion inference is based on delay variation. Intuitively, since (i) packet transmission time is constant, (ii) propagation delay is constant, (iii) packet processing time is usually negligible, it follows that variations in the end-to-end delay relate to *queuing delay* induced by the transmission of packet bursts. While delay variation is the common knob used by all protocols, a fundamental difference is that some designs opt for *One Way Delay* (OWD) measurement, whereas others leverage *Round Trip Time* (RTT) measurement.

Both techniques have pros and cons. First, it could be argued that measuring OWD in the uncontrolled Internet with non-synchronized clocks is a notoriously difficult task. Yet, protocols rely on correctly estimating OWD *variations*, which is a much easier task: indeed, skew between sender and receiver clock cancels in the difference operation, whereas clock drift has a less dramatic impact over a timescale of minutes (and can be corrected over larger timescales).

Conversely, it could be argued that measuring RTT variations couples the state of the queues in the forward and backward directions: it follows that, as RTT variations relate to the sum of queuing delays in both directions, interfering traffic

[3] http://www.cs.utexas.edu/users/dahlin/software/2002-nice.html

[4] To make the matter worse, we additionally point out that while more recent implementation effort exists (e.g., http://wand.net.nz/publications/congestion-control-advancements-linux) that share the same name (i.e., NICE) and goal (i.e., low priority), they are however based on more simplistic design, that have furthemore been less thoroughly tested with respect to [6].

[5] A more structured account of our findings is available along with the code at http://www.telecom-paristech.fr/~drossi/scavenging

on the reverse direction could impact non-congested traffic on the forward direction. Yet, we also notice that merely slowing down the reception of OWD congestion signals (e.g., by cross traffic in the reverse directions) can have a similar effect [10].

C. Congestion inference

All protocols define similar, though slightly different, congestion signals. In particular, denote with D_i the istantaneous OWD sample, and with $D_{min} = \min_{j \le i} D_j$ and $D_{max} = \max_{j \le i} D_j$ the extremal values obseved.

TCP-LP maintains an exponentially weighted average of the OWD delay (i.e., $\overline{D}_i = (1-\alpha)\overline{D}_{i-1} + \alpha D_i$, with $\alpha = 1/8$ by default), whereas LEDBAT measures the distance (i.e., $D_i - D_{min}$) of the current OWD delay sample with respect to the minimum observed value (likely obtained by packets traversing an empty queue) and finally NICE exploits individual RTT samples.

Then TCP-LP, infers congestion whenever the average OWD exceeds a given threshold (specifically, $D_{min}+(D_{max}-D_{min})\delta$, with $\delta = 0.15$), whereas LEDBAT infer queuing whenever current OWD exceeds the minimum ever observed (i.e., $D_t - D_{min} > 0$) and finally NICE detects congestion whenever a significant fraction of RTT samples during the same window exceeds a given threshold (whenever at least half of the packets experience an RTT delay greater than $RTT_{min} + (RTT_{max} - RTT_{max})\delta$, with $\delta = 0.2$).

D. Early (and late) reactions

In absence of congestion indication, TCP-LP behaves like best effort TCP in congestion avoidance, i.e., it additively increases the congestion window. Conversely, NICE and LEDBAT have a more gentle behavior, avoiding additive increase and rather trying stabilizing the congestion window as a function of the measured RTT and OWD delay variations respectively.

Once congestion is inferred, proper reactions are taken: in case of late congestion signals (i.e., losses), all protocols behave like standard TCP (i.e., FastRecovery or Slow-Start as early introduced); reaction to early congestion signals instead differs from protocol to protocol.

In particular, whenever an early congestion is detected, NICE simply halves its congestion window (possibly supporting fractional values by sending one packet after waiting for the appropriate number of RTTs). In case of early congestion, TCP-LP halves the congestion window and enters an inference phase, during which it inhibit additive increase of the congestion window. In case congestion persists at the end of this phase, TCP-LP resets the window to 1 segment and enters a Congestion Avoidance phase. Finally, LEDBAT continuously tune the congestion window proportionally to the distance from a target delay parameter τ: whenever $\tau - (D_i - D_{min})$ is negative, this means that queuing delay exceeds the target, so that the congestion window reduces proportionally to this excess.

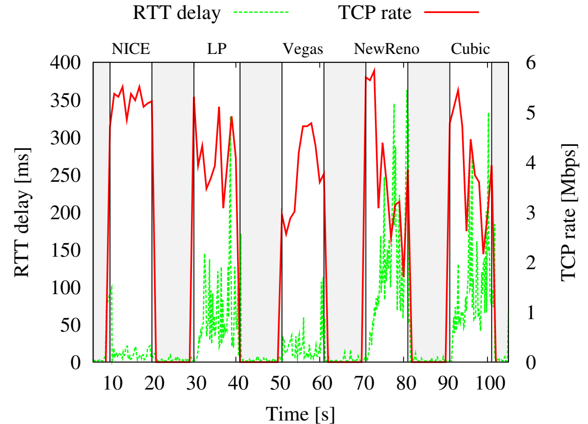

Fig. 1. Temporal evolution of rate (solid line) and RTT delay (dotted line) of different congestion control flavors operating alone on a bottleneck link (WiFi).

IV. Experience

We now report on simple experiments to validate the soundness of our implementation, and describe the current state of low priority protocols in the networking stack of the Linux kernel. In part, our spirit is similar to the comparison of lower than best effort protocols we have carried on by means of ns2 simulation in [11]. Differently from [11], we however employ an experimental methodology, and aim at gathering results that are representative of the great diversity of network environment that constitute nowadays Internet. Additionally, we not only relatively weight the low-priority level of the different protocols, but we also correlate these performance to the expected implication on user experience. As we aim at gathering realistic performance, as opposite to the best possible performance, we avoid a systematic sensitivity analysis over multiple parameters, and rather employ default protocol configuration.

In terms of network environments, we carry on both controlled testbed (emulating network conditions over the loopback interface) as well as on the wild Internet (ranging from a 1Gbps LAN, to an open WiFi network, and a 1Mbps ADSL). As for traffic generation, we either mimick bulk data transfers with `iperf` (that from version 2.0.4 allows to select the congestion control flavor), or generate more realistic traffic patterns through P2P applications. For `iperf` receivers, we use two public servers, one in the same country (`debit.info`) and one in a neighboring country (`eltelnet.info`).

In terms of TCP flavors, we consider Cubic, NewReno for MIMD and AIMD versions of effort TCP respectively. We instead consider TCP-LP and NICE as lower than best effort protocols but exclude LEDBAT for the time being[6]. We finally

[6]While our LEDBAT implementation works flawlessly in ns2, it seldom experiences starvation in the real world, which we are currently debugging: we have observed spurious values of the TCP TimeStamp options as a possible source of the problem (which possibly also affect TCP-LP behavior, as we shall see).

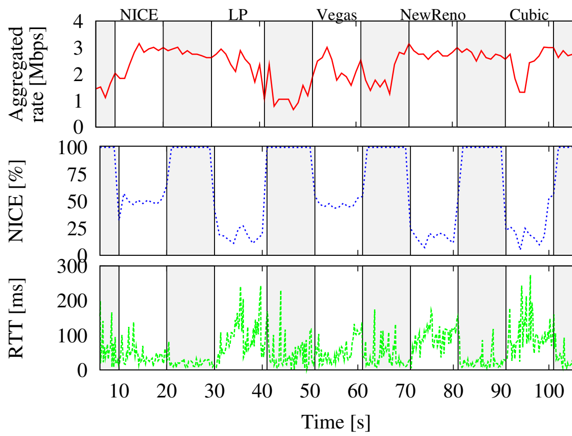

Fig. 2. Temporal evolution of aggregated rate (top plot), breakdown (middle plot) and RTT delay (bottom plot) of different congestion control flavors sharing a bottleneck link (WiFi) with a background backlogged NICE source.

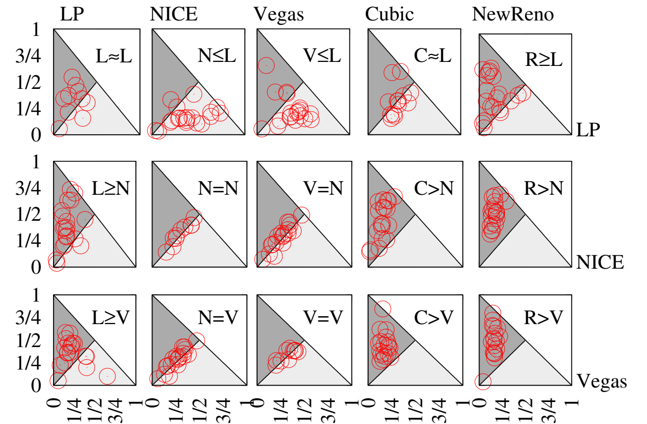

Fig. 3. Scatter plot of relative protocol comparison, reporting normalized throughput for (L)P, (N)ICE, (V)egas, (C)ubic and New(R)eno.

include Vegas that was designed as a delay-based replacement of best effort TCP, though it is known to be less aggressive than AIMD/MIMD protocols and could thus be included in the second category.

A. Temporal evolution

Fig. 1 shows, at a glance, performance of the different congestion control flavor. We start, periodically, a 10-second backlogged transfer changing the TCP flavor at each new period, and measure the instantaneous rate at a 1Hz frequency (solid line). In between each period, we pause transfers for 5 seconds (shaded gray regions). During the whole experiment, we also measure the RTT delay by means of ICMP echo request at a 5Hz frequency (dashed line). Experiments in Fig. 1 are performed between two PCs interconnected through a WiFi AP, explaining variability in throughput.

As expected, delay based congestion control flavors such as Vegas and NICE, introduce the lowest additional queuing delay. At the same time, we also notice that TCP-LP, though delay based, is less effective in limiting queuing delay due to AIMD. A qualitatively similar behavior holds for ADSL lines and loopback interface (with emulated bandwidth limits), though queuing delay in this case can grow up to several seconds[4].

We perform a second set of experiments by running a background backlogged NICE source, and periodically starting 10-second backlogged transfers with different foreground TCP flavors at each new period. As before, we measure instantaneous rates of the two flows and RTT delay. Fig. 2 depicts the aggregated rate of NICE and the second source (top plot), the percent bottleneck share of the background NICE traffic (middle plot) and RTT delay (bottom plot). By construction, between two foreground TCP transfers, only background NICE traffic is present (share is 100% in the shaded gray region). It can be seen that NICE shares equally the traffic with Vegas, but is otherwise lower priority with respect to any other protocol.

B. Relative comparison

While temporal plots convey immediate information, they serve an illustrative purpose, but are not otherwise statistically significant. Hence, we report on a more thorough battery of tests, where we compare all transport protocols. We again focus on the WiFi case, which is the more challenging as capacity is varying due to interference on the same channel.

In each test, we start two backlogged connections using TCP flavors X and Y, and let them compete for bottleneck resources. We repeat each experiment 20 times, and measure the average throughput (x, y) normalized over the capacity C achieved by both protocols in each experiment. We report the most significant comparison adopting a scatter plot representation in Fig. 3, where multiple plots are arranged as a matrix, with each columns and rows representing a specific TCP flavor.

Intuitively, the $x = y$ bisector of the scatter plot is the *fairness* line, i.e., where both protocol achieves the same throughput. The $y = 1 - x$ line instead represents the *efficiency* region, i.e., where the aggregate throughput equals the normalized capacity $x + y = 1$. Hence, we can individuate three regions in each scatter plot: a white non-admissible region in the top left region where $x + y > 1$, a dark-gray region where $x \leq y$ and a light-gray where $x \geq y$. To simplify the interpretation of the plots, we additionally identify each protocol with a single letter, and write the relationship between x and y in the white region.

From the two bottom rows, we notice that while NICE and Vegas are lower priority than Cubic and Reno, they are additionally fair among each other (with the all-Vegas case slightly more efficient than the all-NICE case). Conversely, the top rows clearly indicate that not only LP is more aggressive than NICE and Vegas, but that it is also possibly as aggressive as NewReno (notice indeed that while most points fall in the dark-gray region where NewReno is more aggressive, some points fall along the $x = y$ line) and even more aggressive than Cubic (most points fall along and some below $x = y$). Finally, we also notice that intra-protocol fairness reduces in

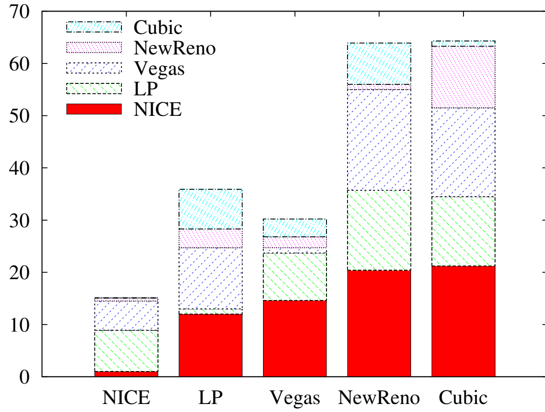

Fig. 4. Relative level of priority. Percentage of times protocol X achieve a higher throughput than its competitors Y (detailed in the bar breakdown).

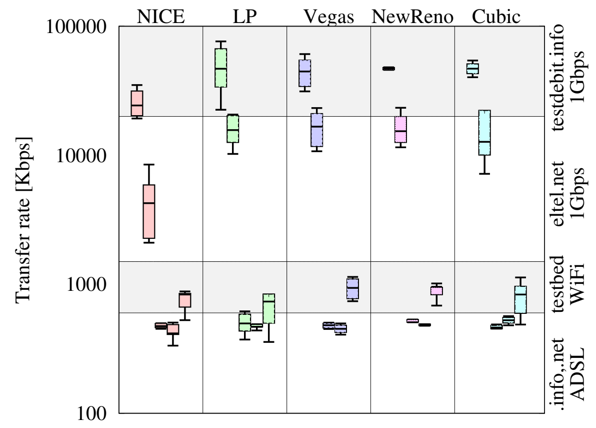

Fig. 5. Boxplots of queuing delay during BitTorrent transfers (mainline client) using different TCP congestion control flavors.

the all-LP case.

These (easily reproducible) results are somewhat unexpected, as they contradict TCP-LP intended behavior[7]. As TCP-LP was designed long before Cubic, it is thus likely that comprehensive regression tests[8] have not been performed since the introduction of the latter.

Finally, we relatively rank priorities of the different TCP flavors in Fig. 4. To define an intuitive order, we compute the fraction of experiments in which protocol X achieves a higher throughput than Y, i.e., $x > y$, and exclude experiments in which a protocol competes against itself. Fractions are normalized to each protocol X, and Fig. 4 additionally reports a visual breakdown of this fraction among the protocols Y that achieve a lower throughput than X.

For instance, considering the case of NICE, it can be seen that it achieves a higher throughput than other protocols only about 15% of the times (or equivalently, that its throughput is lower in 85% of the experiments). It can also be seen that these 15% of the times are roughly equally split among Vegas and TCP-LP: in other words, NICE was never found to be more aggressive than Cubic or NewReno, and only seldom more aggressive than Vegas or TCP-LP. The picture again confirms that TCP-LP is actually more aggressive than Vegas, and that especially it seems to inherits from NewReno its aggressiveness against Cubic.

C. User viewpoint

We now adopt a user-centric viewpoint, to gauge the expected level of her satisfaction. To do so, we point out that user experience depends on the level of interactivity of her applications: bulk data transfers benefit of high throughput, while interactive and multimedia applications are delay-sensitive. As background transfers may be in parallel to some interactive foreground application, they should not generate excessive delay. At the same time, as background transfers may be alone from time to time, they should be able to efficiently use the bottleneck capacity.

This lead us to consider both data rate (Fig. 5) as well as queuing delay (Fig. 6) performance. We estimate data rate, as before, during backlogged transfers (e.g., upload of large videos/music/pictures to Cloud servers) under a variety of access links. We instead estimate queuing delay during chunked-transmissions (e.g., BitTorrent background application) by sampling at high frequency the kernel level queue in controlled experiments as in [12].

Data rates are reported in Fig. 5, for a number of test scenarios. As rates span a rather large capacity range on purpose, we employ a logarithmic scale. From top to bottom, we experiment with a 1Gbps access to `testdebit.info` (receiver located in France, as our sender probe) and to `eltel.net` (receiver located in Belgium), within a local testbed with WiFi access, and to the `testdebit.info` and `eltel.net` servers from an ADSL line. In these experiments, we run two simultaneous connections with homogeneous congestion control to the `iperf` server, and measure raw performance of each connection. Each experiment is repeated 10 times, and we depict boxplots of the data rate distributions. It can be seen that, whereas for low capacities differences between congestion control protocols are less remarkable (e.g., in ADSL all protocols are close to saturating the nominal bottleneck capacity), performance discrepancies start to appear under WiFi (where latency variability plays against NICE and Vegas) and are more evident at 1Gbps access speed (where NICE achieves less than half the average throughput of the other TCP flavors). Hence, the low-priority properties of NICE yield to a potential performance penalty in terms of the achievable data rates (especially when data rates are in excess of 5Mbps).

[7]Further research is needed to fully understand the root causes of these observations. We however point out that, as our LEDBAT implementation, TCP-LP uses TCP TimeStamp to measure OWD delay variations: the spurious TimeStamp values we observed to starve LEDBAT, could be a tied to the unexpected behavior of the TCP-LP as well.

[8]Regression tests are admittedly costly: see the recent ”TCP ex Machina” and ”TCP experiments” threads on end-to-end mailing list http://www.postel.org/e2e.htm. To further hinder the chances of systematic regression tests, we remark that recent Linux kernels are shipped with a default `net.ipv4.tcp_allowed_congestion_control` configuration that inhibits TCP-LP.

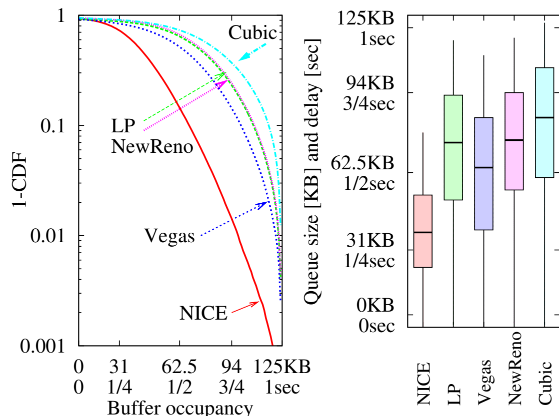

Fig. 6. Inverse cumulative distribution (left) and boxplots (right) of queuing delay during BitTorrent transfers over different TCP congestion control flavors.

Queuing delay characteristics are expressed in terms of the inverse cumulative distribution (left plot of Fig. 5) especially apt at showing large delays. NICE is lowest (approaching 1second less 1/1000 of the times), followed by Vegas, LP Reno and Cubic (that all can grow to 1sec). Quartiles of the distribution, along with 5-th and 95-th percentiles are reported in the boxplots (right of Fig. 5), where the thick line represent median delay. It can be seen that whereas NICE keeps median delay to slightly above 250ms (already possibly hurting interactive traffic), median delay under BitTorrent traffic is consistently above 500ms for all other TCP flavors. Hence, the slight data rate penalty suffered by backlogged background transfers seems a necessary price to pay in order to successfully limit the delay experienced by user during their other, more interactive, activities.

V. Aftermath

Arguing that code development (and maintenance) should be a shared burden, we offer our NICE implementation to the scientific community. We additionally share our experience by performing a broad experimental evaluation of low priority protocols (such as NICE and TCP-LP) against other available flavor in the TCP/IP Linux kernel suite (such as Vegas, NewReno and Cubic). In agrement with ns2 simulation results[11] we find that as expected NICE has the lowest priority in the mix (followed by Vegas).

Surprisingly, we also find that TCP-LP appears to correctly work in low priority only when competing with NewReno (and, even in this case, the low-priority behavior holds most of the times, but not always). More stricking, TCP-LP appears to be quite aggressive against Cubic (a behavior inherited from NewReno, that however requires further investigation).

We also find that, NICE limits the delay significantly more than the other protocol (including Vegas), though it may incur in some data rate inefficiency (unlike Vegas) especially on very high capacity bottleneck links. Overall, the slight data rate penalty suffered by backlogged background transfers under NICE seems a necessary price to pay in order to succesfully limit the delay experienced by user during their other, more interactive, activities.

Indeed, as shown in [4], [10] a high delay can lead to instable behavior also for a single backlogged flow, as the congestion signals are not timely received. Additionally, as shown [12], reducing delay may be beneficial also for more complex applications involving multiple flows: as the same physical channel is shared by both data and control plane, delay due to the former can negatively affect the latter, with potentially negative impact on the overall user experience.

In case successful low-priority protocols were largely available at OS level, they could be easily leveraged at application level. For instance, applications could be simply configured with a list of protocol flavors, ordered by decreasing preference, to provide a fallback in case of unavailability of the former. In case of new flavors[1], changes to the application would require only minor configuration modification. Overall, this approach would prove more efficient in case of regression tests (focusing on few common flavors, rather than a scattering the effort among multiple, and rapidly-evolving, changes at application level) allowing to gather a better understanding (and hopefully control) over the traffic flowing in the network.

VI. Acknowledgement

This work has been carried out at LINCS http://www.lincs.fr, and the LEDBAT implementation cited throughout this work has been carried out by Silvio Valenti. The research leading to these results has received funding from the European Union under the FP7 Grant Agreement n. 318627 (Integrated Project ”mPlane”)

References

[1] K. Winstein and H. Balakrishnan, “Tcp ex machina: Computer-generated congestion control,” in *ACM SIGCOMM*, 2013.
[2] S. Lar and X. Liao, “An initiative for a classified bibliography on tcp/ip congestion control,” *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 126 – 133, 2013.
[3] D. Ros and M. Welzl, “Less-than-best-effort service: A survey of end-to-end approaches,” *IEEE Commuication Surveys & Tutorial*, 2013.
[4] J. Gettys and K. Nichols, “Bufferbloat: Dark buffers in the internet,” *Communications of the ACM*, vol. 55, no. 1, pp. 57–65, 2012.
[5] X. Yang and G. De Veciana, “Service capacity of peer to peer networks,” in *INFOCOM 2004*, vol. 4. IEEE, 2004, pp. 2242–2252.
[6] A. Venkataramani, R. Kokku, and M. Dahlin, “TCP Nice: A mechanism for background transfers,” *USENIX OSDI*, vol. 36, 2002.
[7] S. Shalunov, G. Hazel, J. Iyengar, and M. Kuehlewind, “Low Extra Delay Background Transport (LEDBAT),” IETF RFC 6817, Dec. 2012.
[8] K. Eger, T. Hoßfeld, A. Binzenhofer, and G. Kunzmann, “Efficient simulation of large-scale p2p networks: packet-level vs. flow-level simulations,” in *ACM UPGRADE-CN*, Monterey, CA, Jun. 2007.
[9] A. Kuzmanovic and E. Knightly, “TCP-LP: low-priority service via end-point congestion control,” *IEEE/ACM Transactions on Networking (TON)*, vol. 14, no. 4, p. 752, 2006.
[10] D. Rossi, C. Testa, and S. Valenti, “Yes, we LEDBAT: Playing with the new BitTorrent congestion control algorithm,” in *PAM'10*, Zurich, Switzerland, Apr 2010.
[11] G. Carofiglio, L. Muscariello, D. Rossi, and C. Testa, “A hands-on Assessment of Transport Protocols with Lower than Best Effort Priority,” in *IEEE LCN'10*, Denver, CO, Oct. 2010.
[12] C. Testa, D. Rossi, A. Rao, and A. Legout, “Data plane throughput vs control plane delay: Experimental study of bittorrent performance,” in *IEEE P2P'XIII*, Trento, Italy, Sep. 2013.