Delay-based congestion control: flow vs. BitTorrent swarm perspectives

Claudio Testa, Dario Rossi[1]

[a]*Telecom ParisTech, 46 Rue Barrault, 75013 Paris, France*

Abstract

BitTorrent, one of the most widespread file-sharing P2P applications, recently introduced LEDBAT, a novel congestion control protocol aiming at (i) limiting the additional delay due to queuing, to reduce interference with the rest of user traffic (e.g., Web, VoIP and gaming) sharing the same access bottleneck, and (ii) efficiently using the available link capacity, to provide users with good BitTorrent performance at the same time.

In this work, we adopt two complementary perspectives: namely, a flow viewpoint to assess the Quality of Service (QoS) as in classic congestion control studies, and a BitTorrent swarm viewpoint to assess peer-to-peer users Quality of Experience (QoE). We additionally point out that congestion control literature is rich of protocols, such as VEGAS, LP, and NICE sharing similarities with LEDBAT, that is therefore mandatory to consider in the analysis. Hence, adopting the above viewpoints we both (i) contrast LEDBAT to the other protocols and (ii) provide deep understanding of the novel protocol and its implication on QoS and QoE.

Our simulation based investigation yields several insights. At flow-level, we gather LEDBAT to be lowest priority among all protocols, which follows from its design that strives to explicitly bound the queuing delay at the bottleneck link to a maximum *target* value. At the same time, we see that this very same protocol parameter can be exploited by adversaries, that can set a *higher target* to gain an unfair advantage over competitors. Interestingly, swarm-level performance exhibit an opposite trade-off, with *smaller targets* being more advantageous for QoE of BitTorrent users. This can be explained with the fact that larger delay targets slow down BitTorrent signaling task, with possibly negative effect on the swarming protocol efficiency. Additionally, we see that for the above reason, in heterogeneous swarms, any delay-based protocol (i.e., not only LEDBAT but also VEGAS or NICE) can yield a competitive QoE advantage over loss-based TCP.

Overall this tension between swarm and flow-levels suggests that, at least in current ADSL/cable access bottleneck scenarios, a safe LEDBAT operational point may be used in practice. At the same time, our results also point out that benefits similar to LEDBAT can also be gathered with other delay-based protocols such as VEGAS or NICE.

Keywords: BitTorrent; Congestion control; Lower-than best effort congestion control; LEDBAT; TCP-LP; VEGAS; NICE;

1. Introduction

Pioneered by Jain [22] in late 80s, delay-based Congestion Control (CC) has been out for a long time, with notable proposals over the years such as VEGAS [8] in late 90s, NICE [42] and LP [25] in early 2000 and more recently LEDBAT [35] in 2010.

The idea of this branch of protocols is to use the variation in the end-to-end delay transmission as *early* congestion signal: in other words, a growing delay beyond a baseline is interpreted as queuing delay building up at the bottleneck link, and the amount of data to be sent at every time frame is updated accordingly. This design choice is orthogonal to the one adopted by loss-based protocols, such as in classic TCP NewReno [18] which instead uses packet loss as a *late* congestion signal to tune the data transmission.

Since loss-based protocols forcibly fill the buffer, this can translate into rather large delays, especially at the access link where buffer sizes are relatively large compared to the narrow capacity of ADSL and cable modems. As recent work pointed out, is not uncommon that queuing delays exceed the Earth-to-Moon propagation delay [23, 13], for which the "bufferbloat" term was recently coined [12].

Clearly, such huge delays can harm the Quality of Ex-

[1]Corresponding author dario.rossi@enst.fr

perience (QoE) of interactive communication – including Voice over IP (VoIP), gaming and Web browsing. Additionally, since the bottleneck is placed at the user access link, this means that the user is self-inflicting this QoE degradation, as his own traffic is competing for the bottleneck resources. In other words, QoE degradation results from sustained uploads carried on TCP, whose loss-based Additive Increase Multiplicative Decrease (AIMD) protocol forces the buffer to fill prior to halve the congestion window due to losses.

It follows that bufferbloat can be induced by any application transferring large data volumes over TCP, such as any upload to the Cloud (e.g., Picasa, DropBox, Flickr, etc.), or peer-to-peer file-sharing (e.g., BitTorrent, eDonkey, etc.). To avoid harming contemporary interactive communication of the same user, application developers have thus the choice to exploit alternatives to the standard loss-based TCP behavior. This is precisely the choice of BitTorrent, that recently replaced loss-based TCP with delay-based LEDBAT for data transfer.

This evolution motivates our first viewpoint. As the new protocol is used in BitTorrent swarms, it is important to assess its impact on the quality of BitTorrent users experience – mainly, their completion time [26]. Interestingly though, the protocol has been normalized at the IETF under the Low Extra Delay Background Transport (LEDBAT) in late 2012 [36]. This motivates the second viewpoint: as the protocol is normalized at the IETF, its scope is wider than the BitTorrent ecosystem, and its impact on other applications has to be assessed as well.

In this work, we investigate LEDBAT by means of `ns2` simulations, and compare it to other delay-based protocols such as LP, VEGAS and NICE, from both flow vs. swarm perspective. Moreover, in case of LEDBAT we carry out a sensitivity analysis over its main parameter, namely the queuing delay target, to assess the impact of heterogeneous settings. This is an important study, since the parameter can be easily modified by legitimate end-users or legacy implementations (complying to the RFC specification) or by malicious users and developers (violating RFC specification) to possibly gain an unfair advantage. At flow-level, we study the Quality of Service (QoS) of backlogged flows, expressed as the usual network-centric metrics of congestion control studies, such as link efficiency, throughput, packet loss, etc. At swarm-level, we instead study the Quality of Experience (QoE) of BitTorrent users, expressed as the torrent completion time, that collectively depends on the performance of multiple flows, in a furthermore non-trivial way as we shall see.

Summarizing our most interesting findings, we have that heterogeneous LEDBAT target settings yield to significant unfairness, which is especially true for backlogged connections, where flows with slightly *higher delay target* can starve competing flows. Interestingly though, competitive advantage for selfish users in the swarm case are obtained for *lower delay target* – which suggests that safe LEDBAT operational points may be used in practical cases. At the same time, our results also point out that benefits similar to LEDBAT can also be obtained with other delay-based protocols such as VEGAS or NICE.

The remainder of this work is organized as follows. Related work are discussed in Sec. 2, while a detailed overview about the congestion control protocols we consider in this study is reported in Sec. 3. Flow vs. swarm perspectives are then adopted in Sec. 4 vs. Sec. 5 respectively. For both perspectives, we investigate the novel LEDBAT protocol (e.g., carrying out a detailed sensitivity analysis of the queuing delay target parameter, and especially of heterogeneous target settings) and contrast performance with that achievable under LP, VEGAS or NICE. Finally, our findings are summarized and discussed in Sec. 6.

2. Related work

While our most important findings arise in the opposite implications of LEDBAT target settings in the flow vs. swarm perspectives, we point out that, so far, all related effort has focused on either viewpoint in isolation. Hence, we separately treat the above perspectives in this section. In more details, at flow-level, we overview delay-based congestion control protocols and focus on recent work targeting LEDBAT. At swarm-level, we overview studies of BitTorrent performance, and focus on work targeting the impact of packet-level dynamics on content distribution performance.

2.1. Flow viewpoint

Congestion control is a long studied subject: as it would be out-of-scope to provide a full review of the existing literature here, we concentrate on the subset that is most relevant for our work. We present four different categories of congestion control protocols in Fig. 1. Those protocols can be classified based on the *design strategy* (loss-based vs. delay-based) and their *aggressiveness* (high-priority vs. low priority) in capturing the available bandwidth. IETF endorses TCP NewReno [18], a high-priority loss-based congestion

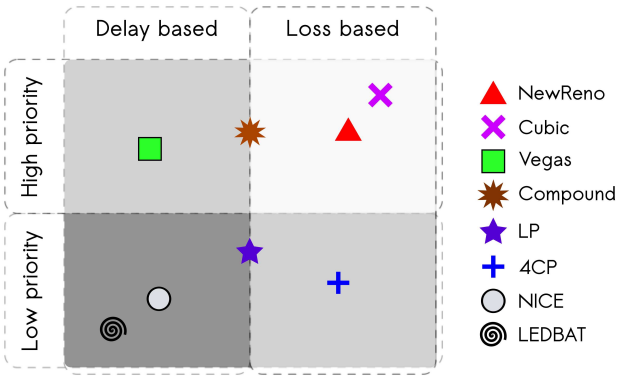


Figure 1: Congestion control design space: aggressiveness vs design strategy.

control algorithm. Recent evolutions of loss-based protocols include Cubic [33] and Compound TCP [37]. Cubic has become the default TCP flavor in Linux (since kernel version 2.6.18) and Compound TCP in Windows operating system (since Vista). As far as the delay-based and low-priority solutions are concerned, that are most interesting to our work, we will focus on LP [25], VEGAS [8], NICE [42] and LEDBAT [36], that we detail in Sec. 3.

Since its proposal, LEDBAT triggered the interest of the scientific community, that started dwelling on several aspects via either experiments [15, 31, 34, 20, 13], simulation [32, 9, 20] or analysis [10, 19]. Delay related issues are addressed in [15, 13]: in [15], BitTorrent developers detail an algorithm to correct the clock drift, while we exploit LEDBAT to gauge bufferbloat delay of remote hosts in [13]. An experimental analysis of LEDBAT performance is carried out in [31, 34]. In [31], we use a black-box approach to study initial closed-source versions of the protocol in its early stage of adoption. Authors in [34] instead study their own LEDBAT implementation in a local testbed, employing different real ADSL modems, exposing a negative interaction with Active Queue Management (AQM) in some modem. We further investigate the interaction between delay-based protocols and AQM via experiments/simulation [20] and fluid modeling [19]: as AQM reinstate fairness among protocols, it practically equalizes priorities and nullifies the differences in protocol design (recall Fig. 1). A simulative approach is instead adopted in [32, 11, 10, 9, 24]. In [32] we unveil a *latecomer issue* that affect the LEDBAT protocol, where newly arriving connections can starve already existing ones. We design and simulate [11] and analyze via fluid modeling [10] solutions to the latecomer unfairness, while authors in [24] compare alternative window decrease schemes.

In our previous work [9], that is closest related to this for what concerns the flow viewpoint, we directly compare LEDBAT to LP and NICE, to (i) relatively weight their low-priority level and (ii) provide a thorough sensitivity analysis of LEDBAT parameters (i.e., gain and target) in both an inter-protocol (i.e., against TCP) and intra-protocol (against LEDBAT itself) scenarios. Building over [9], Sec. 4 focuses on the most relevant LEDBAT parameter (namely, the target), and report the most interesting results additionally including the VEGAS protocol (not considered in [9]). Simulation performed in this work are however entirely new, so as to comply with the default LEDBAT settings, that were updated during the RFC standardization process and thus differ with respect to [9]. Our implementation of LEDBAT for `ns2` is available at [3].

2.2. Swarm viewpoint

As the most successful peer-to-peer file-sharing application, BitTorrent has become over the years a rather popular research subject, investigated with analytic [29], measurement [21, 43] simulative [6, 7, 41, 17, 10, 38] or experimental [5, 30, 39] approaches. Yet, as most studies were carried out before the proposal of LEDBAT, it follows that to date only few work studies LEDBAT impact on BitTorrent performance, via either simulation [38] or experiments [39, 40].

More precisely, among the pre-LEDBAT studies, [29] presents a fluid model of swarming performance, while [21, 43, 28] use a measurement approach to shed light on the Internet footprint of BitTorrent, either analyzing tracker logs [21] or via large-scale crawling of the entire BitTorrent ecosystem [43]. Simulation is used to study the impact of tit-for-tat [6], overlay parameters [41] and traffic locality [7] on BitTorrent performance.

However, the above work limitedly consider application dynamics but otherwise neglects the impact of packet-level dynamics, such as congestion control. A packet-level implementation of the BitTorrent protocol for `ns2` is available at [1], that is used in [17] to contrast a simplistic flow-level to a more realistic packet-level viewpoint. Finding of [17] is that transport-layer congestion control dynamics actually do interact with application-level dynamics, so that overlay-only simulation results [6, 41, 7] may report optimistic completion times.

Following [17], we consider transport-layer dynamics and assess the impact of LEDBAT on BitTorrent

completion time via simulation [38]. Yet another possibility it to perform experiments with real BitTorrent clients [30, 5, 39, 40]. In particular, authors in [30] address the problem of reproducibility and reliability testbed-driven results, that are found to be close enough to results gathered in the real Internet. In a joint work with authors of [5, 30], we employ an experimental methodology to study torrent completion time under LEDBAT [39, 40]. Interestingly, experimental results [39, 40] validates the simulation findings of [38], and suggest thus simulative approach to be worthwhile, as it allows to explore a large investigation space, as we do in this work.

Building over [38], that is closest related to this work for what concerns the swarm viewpoint, Sec. 5 employs the `ns2` simulator [17] to simulate BitTorrent swarm under a mixture of congestion control protocols. More precisely, while [38] limited considered LEDBAT, in Sec. 5 we extend the current knowledge by (i) consider a broader range of congestion control protocols, i.e., LP, NICE and VEGAS, (ii) extend the considered swarm settings (e.g., seed leave policies) to ensure the generality of our findings, (iii) perform a sensitivity analysis of heterogeneous LEDBAT target settings, that leads to novel important findings about LEDBAT operation.

3. Background

We perform a preliminary set of simulations, that assists the description of important similarities and differences of the congestion control protocols we use in this work. Fig. 2 reports the congestion window evolution of two backlogged flows sharing a bottleneck link obtained via `ns2` simulation. We consider an access bottleneck of C =10 Mbps, a RTT=50 ms, with a queue size of B=100 full-size packets (larger than the bandwidth delay product as common in practice [23]). Top plots of Fig. 2 refer to the heterogeneous protocol case where one flow employs standard loss-based TCP NewReno, while the other flow employs a delay-based protocol among (a) VEGAS, (b) LP, (c) NICE or (d) LEDBAT. Bottom plots report the bandwidth share in the homogeneous flow case, where thus both flows employ (e) VEGAS, (f) LP, (g) NICE or (h) LEDBAT. Already at first sight, it is possible to gather that VEGAS, NICE and LEDBAT have very similar behavior (with a very smooth congestion window), while LP is more aggressive (and saw-toothed congestion window profile). Notice further that while LP, NICE and LEDBAT share the same low-priority spirit, VEGAS was designed for higher efficiency but is known to be less aggressive that TCP NewReno [4].

3.1. TCP-Vegas

TCP-Vegas (or VEGAS *tout court*) exploits the simple idea that the number of bytes in transit is directly proportional to the expected throughput. VEGAS maintains an estimate RTT_{min} of the minimum measured Round Trip Times (RTT), corresponding to the RTT encountered when the bottleneck queues is empty. Then, the expected throughput is given by $Expected(t) = W(t)/RTT_{min}$, where $W(t)$ is the size of the congestion window at time t. Similarly, it calculates the current actual sending rate as $Actual(t) = W(t)/RTT(t)$, and adjusts the $cwnd$ according to the difference between $Actual(t)$ and $Expected(t)$ throughput:

$$\Delta(t) = Expected(t) - Actual(t) \tag{1}$$

For the $cwnd$ adjustment two thresholds are defined: $\alpha < \beta$. When $\Delta(t) < \alpha$, VEGAS increases $cwnd$ linearly during the next RTT, while if $\Delta(t) > \beta$, VEGAS decreases $cwnd$ linearly during the next RTT, and leave $cwnd$ unchanged otherwise:

$$cwnd(t+1) = \begin{cases} cwnd(t)+1 & \text{if } \Delta(t) < \alpha, \\ cwnd(t)-1 & \text{if } \Delta(t) > \beta, \\ cwnd(t) & \text{if } \alpha < \Delta(t) < \beta. \end{cases} \tag{2}$$

As shown in Fig. 2-(a) and (e), VEGAS is less aggressive with respect to TCP NewReno, and efficiently and fairly shares the bottleneck link in the intra-protocol case.

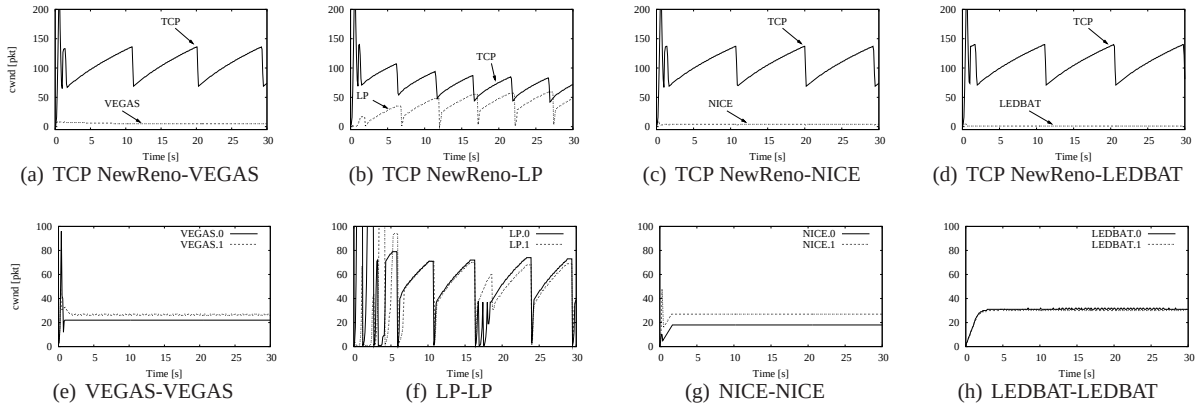
3.2. TCP-LP

TCP-LP (or LP *tout court*) measures the One-Way Delay (OWD) and employs a simple delay threshold-based method for early inference of congestion. More specifically, LP estimates the minimum OWD_{min} and maximum OWD_{max} delays, filtering the instantaneous measure $OWD(t)$ by means of an exponentially weighted moving average $\tilde{OWD}(t)$ with smoothing parameter α, updated packet-by-packet. The smoothed average $\tilde{OWD}(t)$ and the condition for early-congestion detection are:

$$\tilde{OWD}(t) = (1-\alpha)\tilde{OWD}(t-1) + \alpha OWD(t) \tag{3}$$

$$\tilde{OWD}(t) > OWD_{min} + (OWD_{max} - OWD_{min})\delta \tag{4}$$

where $\delta \in (0,1)$ is a custom threshold parameter. Throughout this paper, we use the default values $\alpha = 1/8$, $\delta = 0.15$ as in [25].



(a) TCP NewReno-VEGAS (b) TCP NewReno-LP (c) TCP NewReno-NICE (d) TCP NewReno-LEDBAT

(e) VEGAS-VEGAS (f) LP-LP (g) NICE-NICE (h) LEDBAT-LEDBAT

Figure 2: Congestion control protocols at a glance: Inter (top) and Intra (bottom) protocol interaction on a simple bottleneck (C=10 Mbps, RTT=50 ms, B=100 packets)

In the absence of early-congestion indication, LP behaves like standard TCP NewReno, i.e., performing an additive increase of $cwnd$ as shown by the saw-tooth behavior in Fig. 2-(b) and (f). Whenever an early-congestion is detected, according to the rules outlined above, LP halves $cwnd$ and enters an inference phase that last for a preconfigured time. During this period, LP only observes responses from the network and avoids increasing the congestion window. If congestion persists at the end of this phase, LP reduces $cwnd$ to 1 and restarts a slow-start phase as evident from Fig. 2-(b). Finally, in case of losses, LP behaves like TCP NewReno.

3.3. TCP-NICE

TCP-NICE (or NICE *tout court*) instead maintains minimum RTT_{min} and maximum RTT_{max} estimates of the round trip delay. Congestion is detected when more than a given fraction ϕ of packets during the same RTT experiences a delay exceeding:

$$RTT > RTT_{min} + (RTT_{max} - RTT_{min})\delta \quad (5)$$

where δ and ϕ are protocol parameters set to $\delta = 0.2$ and $\phi = 0.5$ as in [42]. Notice that (5) is the same formula of LP (3), but computed on the RTT variable, and using the fraction-trick instead of a moving average.

In the absence of congestion, NICE behaves like VEGAS. Whenever early-congestion is signaled, NICE simply halves its congestion windows and sending rate, practically anticipating the multiplicative decrease behavior. Finally, when a loss is detected NICE behaves like TCP NewReno. We point out that NICE allows $cwnd$ to be a fraction of 1 by sending one packet after waiting for the appropriate number of RTTs: the use of fractional values for $cwnd$ guarantees non-intrusiveness even in the case of many NICE flows sharing the same bottleneck.

As clearly emerges from the smooth $cwnd$ behavior in Fig. 2-(c) and (g), NICE inherits its congestion control algorithm from VEGAS (rather than from TCP NewReno as LP), so that the throughput stabilizes around the effective capacity in the intra-protocol case.

3.4. LEDBAT

Finally, LEDBAT maintains a minimum OWD estimation OWD_{min}, which is used as base delay to infer the amount of queuing delay. Each flow has a target queuing delay τ, i.e., they aim at introducing a small, fixed, amount of delay in the queue of the bottleneck buffer. The flow continuously monitors the variations of the queuing delay $OWD(t) - OWD_{min}$ to evaluate the distance $\Delta(t)$ from its target as in (6):

$$\Delta(t) = [\tau - (OWD(t) - OWD_{min})]/\tau \quad (6)$$

$$cwnd(t+1) = cwnd(t) + \gamma\Delta(t)/cwnd(t) \quad (7)$$

where γ is a parameter responsible for the steepness of the $cwnd$ update. In the absence of early-congestion indication, i.e., when the target τ has not been reached yet, $\Delta(t) > 0$ in (6) and thus $cwnd$ grows as defined by (7). As soon as the target is reached, $\Delta(t) = 0$, thus $cwnd$ settles. Values of $\Delta(t) < 0$ are perceived as early-congestion indication (i.e., other traffic is increasing the queuing delay $OWD(t) - OWD_{min} > 0$), to which LEDBAT reacts by reducing $cwnd$ proportionally to the offset from the target. Finally, in case of losses, it behaves like TCP NewReno.

Note that as per (6) the ramp-up is limited to at most match the TCP NewReno ramp-up in congestion avoidance (i.e., 1 packet per RTT), which happens when the queue is empty (i.e, $OWD(t) - OWD_{min} = 0$). Note also that $\tau > 0$ is necessary in order for the capacity to be fully exploited. At the same time, τ should be as small as possible to avoid harming interactive communication. Early version of the draft used a mandatory $\tau = 25$ ms value, though this was too small to be used in practice and the RFC later recommend $\tau \leq 100$ ms. As we will see in the following, this possibly opens an arm race between applications/implementations using LEDBAT, since flows with higher τ (even though complying to RFC specifications) have a faster ramp-up and can starve the other flows. Additionally, while system-wide options (as TCP parameters) need super-user privileges to be tweaked, the value of τ is the μTorrent implementation can be easily overridden by users (by simply modifying the `net.utp_target_delay` value in the GUI), extending the arm race to users.

Overall, LEDBAT shares similarities with, and exhibits differences from, the other protocols: (i) as LP, it relies on OWD estimation to detect congestion, but unlike LP it does not employ a smoothing average; (ii) as NICE and VEGAS its congestion window dynamics are based on the delay, but unlike them, it defines a fixed target delay. As we can see from Fig. 2, the behavior of LEDBAT is however closer to NICE and VEGAS than to LP.

4. Flow perspective

This section has three main aims. We first (i) compare the different protocols from a flow-level perspective, using different network-centric (e.g., link utilization, packet loss) and user-centric (e.g. traffic share, queuing delay) QoS metrics. We then perform a sensitivity analysis of the LEDBAT target delay parameter τ on the system performance. Sensitivity is carried out in both (ii) an inter-protocol case, where a TCP NewReno flow and a LEDBAT flow share the bottleneck and (iii) an intra-protocol case, where two LEDBAT flows compete against each other.

The aim of (i) is to precisely quantify similarities and differences among protocols, that were qualitatively shown early in Fig. 2. Then, we assess whether (ii) τ offers the chance to tune the level of aggressiveness in LEDBAT, and (iii) further aims at verifying whether unfairness may arise among legacy LEDBAT implementations (e.g., different releases of the same code, different implementations or parameter settings, etc.).

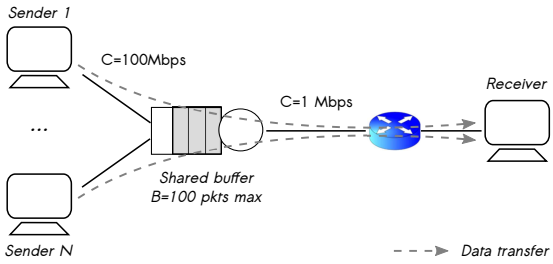


Figure 3: Synoptic of the flow-level simulations.

Performance figures are gathered via `ns2` simulations. While TCP NewReno, VEGAS and LP protocols are already implemented, we implement both NICE and LEDBAT, that we make available at [3]. As reference network scenario, we use a dumbell topology where the capacity of the bottleneck is fixed to $C = 1$ Mbps, the one-way propagation delay equals 25 ms (thus round trip delay is equal to $RTT = 50$ ms), and the buffer size is set to $B_{max} = 100$ packets[2], as depicted in Fig. 3. We consider backlogged sources[3], that use a fixed packet size equal to $S = 1500$ Bytes. All flows start simultaneously, so that we avoid potential latecomer issues [32], and last for 120 seconds. For the time being, we fix the LEDBAT target delay to $\tau =$100 ms, i.e., the maximum value compliant to RFC recommendations.

Performance are expressed in terms of the link utilization (η), i.e., the ratio between the sum of the throughput of each flow x_i over the bottleneck capacity $\eta = \sum_i x_i/C$. We quantify priority (in the inter- or intra- protocol cases) with the traffic share of a given flow f (or set of flows) $P_f = \sum_{i \in f} x_i / \sum_j x_j$. Finally, we report average buffer size $E[B]$ (correlated to the user delay) and loss probability.

We point out that a more systematic and detailed comparison in terms of scenarios (e.g., including a set of more realistic and diverse scenarios, sensitivity of more LEDBAT parameters, etc.) and metric (e.g., the short- and long-term Jain fairness index among the flows, queue size, packet loss probability, etc.) can be found in [9]. In this section though, we do report the most interesting findings, that we further extend to include the VEGAS case.

[2]Given the bottleneck capacity, this buffer size corresponds to a about 1.2 s worth of queuing delay, or equivalently to the Earth-to-Moon propagation delay early mentioned. Notice that according to [23], this scenario correspond to Internet queuing delays that are significantly lower than those observed in practice.

[3]As we consider backlogged sources only, dynamics of LEDBAT are well described by means of (7) only; in case of non-backlogged sources, the dynamics changes slightly to avoid cwnd increase indefinitely [35].

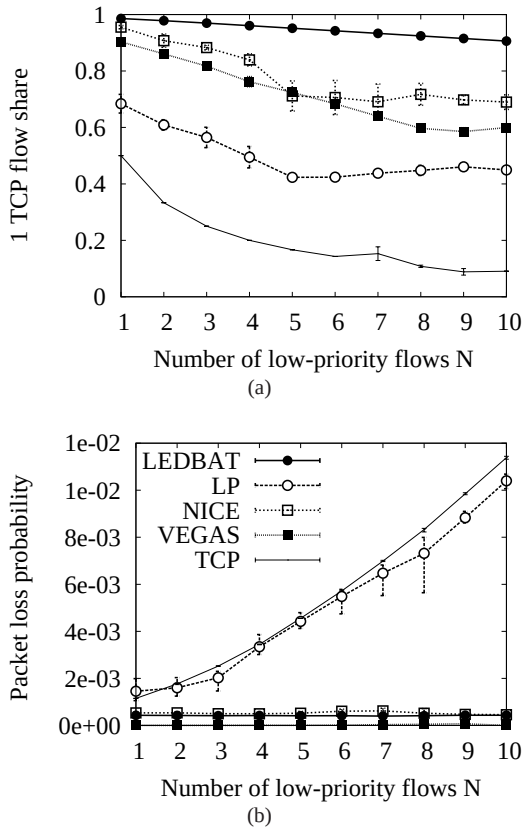


Figure 4: Relative assessment of delay-based protocols: (a) TCP share and (b) packet loss rate as a function of the number of delay-based flows.

4.1. Relative protocol assessment

We first provide a relative assessment of each delay-based protocol against TCP NewReno. We consider a typical scenario where $N \in [1, 10]$ low-priority flows (e.g., due to P2P or other Cloud services) share the same bottleneck with a single TCP NewReno connection, (representative of a generic high-priority service), for a total of $N + 1$ flows. We perform several sets of simulations separately and, for reference purpose, we also simulate the case where $N+1$ TCP NewReno flows share the same bottleneck.

The TCP share reported in Fig. 4-(a), states that e.g., in the $N = 10$ LEDBAT case, the single TCP NewReno consumes about 90% of the link capacity (since $\eta \simeq 1$), leaving thus each of the $N = 10$ LEDBAT flows a mere 1% of the capacity each. Comparing this result with NICE (about 3% each) VEGAS (about 4% each) or LP (about 5% each) under the same $N = 10$ settings, we gather that LEDBAT achieves the lowest priority, closely followed by NICE and VEGAS.

Further differences are stressed by the packet loss probability reported in Fig. 4-(b), that reflects differences among delay-based versus loss-based congestion control principles: indeed, packet loss rate increases proportionally to the number of connections for TCP NewReno and LP, but is not affected by the number of NICE, VEGAS, or LEDBAT flows, since they all try to avoid losses as much as possible (so that a non-zero loss rate is due to the single TCP NewReno flow).

4.2. LEDBAT target sensitivity

Since LEDBAT is the most recent, and thus less known among the congestion control protocols we consider in this work, we need to check correctness of operation according to its low-priority goal (Sec. 4.2.1), as well as investigate if target heterogeneity leads to unfair competitive advantages (Sec. 4.2.2).

4.2.1. Inter-protocol: LEDBAT vs TCP

We start our sensitivity analysis by considering two flows, a standard TCP NewReno and a LEDBAT one, that start simultaneously and vary the values of the $\tau \in [24, 1800]$ ms which corresponds to $T \in [2, 150]\%$ of the buffer size. Given the large span of uplink capacities, and that also the amount of available buffer space in modems spans over an order of magnitude (e.g., 34KB-384KB[16]), the same LEDBAT queuing delay target may correspond to different buffer occupancy ratios: hence, more than the *absolute* value (top x-axis) this investigation should be interpreted according to the *relative* value (bottom x-axis) with respect to the buffer size.

The importance of this analysis is motivated as follows. As previously underlined, the mandatory value for target increased from $\tau = 25$ ms (transmission time of about 1 full size packet at 500 Kbps) to $\tau \leq 100$ ms during the protocol evolution. The choice of τ is somewhat arbitrary (e.g., based on unreported experiments) or motivated by practical constraints (e.g., clock precision, etc.) so that τ is often referred to as "magic number" in LEDBAT WG discussion [2]. It is thus imperative to individuate not only those working modes that correspond to legitimate and compliant settings, but also to malicious or erroneous configurations, or similarly, to changes in the underlying hardware (e.g., modem buffer size) or hardware configuration (e.g., buffer size limited in software to alleviate bufferbloat).

Fig. 5-(a) reports the value of link utilization η and Fig. 5-(b) the share of the TCP flow as a function of the target τ. From Fig. 5-(a) we see that the efficiency η is practically unaffected by variations of target and

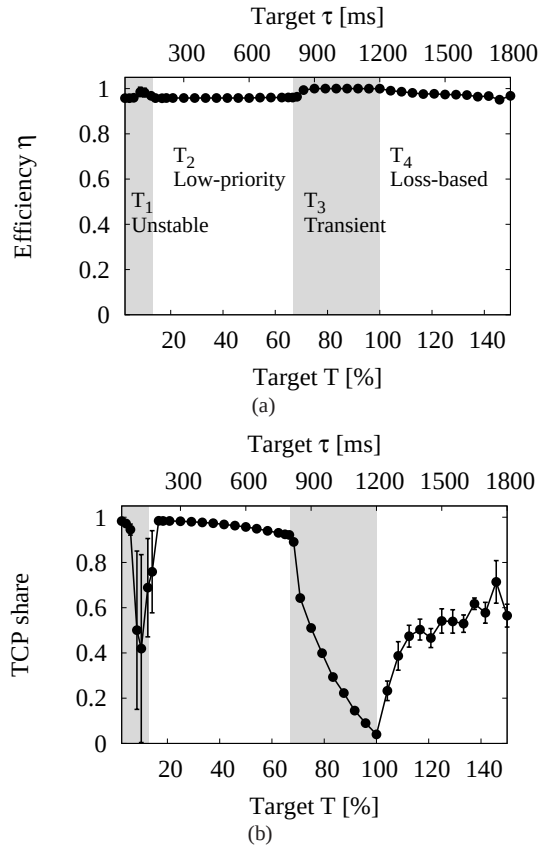


Figure 5: LEDBAT vs TCP NewReno: Inter-protocol sensitivity analysis: (a) link utilization and (b) TCP share for varying target τ.

remains always close to the total link capacity. This is a positive, though expected, finding: even if the target is misconfigured, either LEDBAT or TCP NewReno can take advantage of the unused bandwidth, which result in an overall efficient use of the link capacity.

Considering instead the TCP share reported in Fig. 5-(b), we can identify four working regions. When the target is very small $\tau \ll 100$ ms the LEDBAT protocol is not always able to reach the target delay, which leads to shaky behavior (e.g., as very low target delays may be exceeded already by having a single packet in the queue). In a second region (fairly large in terms of τ settings), LEDBAT completely yields to the TCP NewReno flows, working in low-priority mode and thus attaining its goal. In a third region where the LEDBAT target approaches (but does not exceed) the queue size $T_3 \in [65, 100]\%$, LEDBAT aggressively starts to erode bandwidth to the TCP NewReno flow: this causes losses in the TCP NewReno flow, which progressively backs off. As a consequence, the TCP share starts decreasing until LEDBAT has the monopoly of the buffer, which happens when it aims at precisely filling the buffer $T = 100\%$ and TCP NewReno starves (TCP% tends to 0%). As soon as the target exceeds the buffer size, LEDBAT revert to standard loss-based TCP NewReno behavior, and both flows compete fairly for the bottleneck resources.

Overall, the inter-protocol sensitivity analysis suggests that, although LEDBAT spans a wide range of low-priority levels (especially in the third region, that exhibits a sharp transition phase), a precise tuning is highly impractical (as slight variation of τ leads to completely different scenarios, where either LEDBAT or TCP NewReno exhibits starvation). As such, LEDBAT seems to have a single low-priority level (second region, with a fairly large range of τ values) that is furthermore the lowest among all the delay-based protocols examined in the previous section.

4.2.2. Intra-protocol: LEDBAT vs LEDBAT

The sensitivity analysis in the inter-protocol case is reported in Fig. 6. Top plot of Fig. 6-(a) depicts, as a function of the τ_1/τ_2 target ratio, the link utilization η, the $P_{\tau 2} = x_2/(x_1 + x_2)$ share of the LEDBAT flow with the smaller target (as $\tau_1 > \tau_2 = 100$ ms), and the normalized buffer length $E[B]/B_{max}$. Is immediate to see that even slight differences in the target settings may have strong consequences on the protocol fairness. Indeed a sharp transition phase on the breakdown happens as soon as $\tau_1/\tau_2 > 1$, where the share of the second flow rapidly drops to $P_{\tau 2} = 0$. As a matter of fact, if both flows start at the same time, they both measure the same base delay, and the higher-target flow converges faster to its target and stabilizes: as the amount of queuing is now larger than the one of the less aggressive flow, the latter backs off and starves. This holds irrespectively of whether the higher-target flow operates in delay-based (white shaded region) or loss-based regime (gray shaded region): under both regimes, the higher-target flow will always be advantaged prior than losses occur, and so the unfairness persists.

An example of the congestion window evolution in case $\tau_1/\tau_2 = 1.2$ is represented in Fig. 6-(b), and it can be seen that whenever $\tau_1 > \tau_2$ the first flow will starve the second. While the picture shows an homogeneous RTT scenario, actually, unfairness due to target heterogenity always dominates the unfairness due to RTT heterogenity: this is because RTT unfainess just drives the rate at which changes happen, whereas target heterogenity drives the amount of change. It follows that backlogged flows with larger targets, by design, will starve flows with smaller target (only, starvation will

take more if larger-target flows also have a larger RTT). At the same time, we point out that starvation is surely reached whenever flows are backlogged, though unfairness is less dramatic in case of short-lived flows [10] Notice indeed that in Fig. 6-(b), the second flow share reaches 0 after about 2 minutes. Thus, we expect latecomer unfairness (for which, by the way, known solution exists [11, 10]) to be a second-order detail in the case of BitTorrent experiments of Sec. 5.

Overall, we see that tuning of the protocol priority via the target parameter is highly impractical, as even small difference in that value for two flows produces an extremely unfair situation. Notice that this situation may happen also with non-malicious users with heterogeneous τ_1, τ_2 targets settings that are both complying with the LEDBAT RFC (i.e., $\tau_1 \neq \tau_2 \leq 100\,\text{ms}$). It follows that is not possible to enforce multiple, finer-grained, levels of priority among LEDBAT flows in parallel. Yet, an interesting observation is that two priority levels are possible: flows with the largest τ dominate the other flows, so that transfers of heterogeneous target flows happens sequentially (as if flows were scheduled according to their priority).

5. Swarm perspective

As before, this section has three main aims. We first compare the different delay-based protocols from a swarm-level perspective, using torrent completion time as the main metric, in a (i) *homogeneous* and (ii) *heterogeneous* peers population. We then perform (iii) a sensitivity analysis of the LEDBAT target delay parameter τ on the intra-protocol case, letting peers have heterogeneous target τ settings in all-LEDBAT swarm.

We integrate our open-source implementation of LEDBAT with the BitTorrent open-source implementations [17] for `ns2`. The [17] module runs a fully fledged BitTorrent protocol, implementing all the relevant aspects of the protocol dynamics (e.g., tit for tat reciprocation, rarest firts chunk selection, etc.). For reason of space, we assume the reader is familiar with BitTorrent (and otherwise refer the reader to [17, 5] for a detailed overview of the protocol). We now briefly describe the swarm-scenario with the help of Fig. 7. As commonly assumed, we consider the bottleneck to be represented by peer access link, which is also one of the main motivations that led to the design of LEDBAT. As access technology, we consider ADSL-like connections with $C = 1\,\text{Mbps}$ uplink capacity (homogeneous for the whole peer population) and 8 Mbps downlink capacity. Access nodes are then interconnected directly to the Internet, that we model by means of infinite capacity,

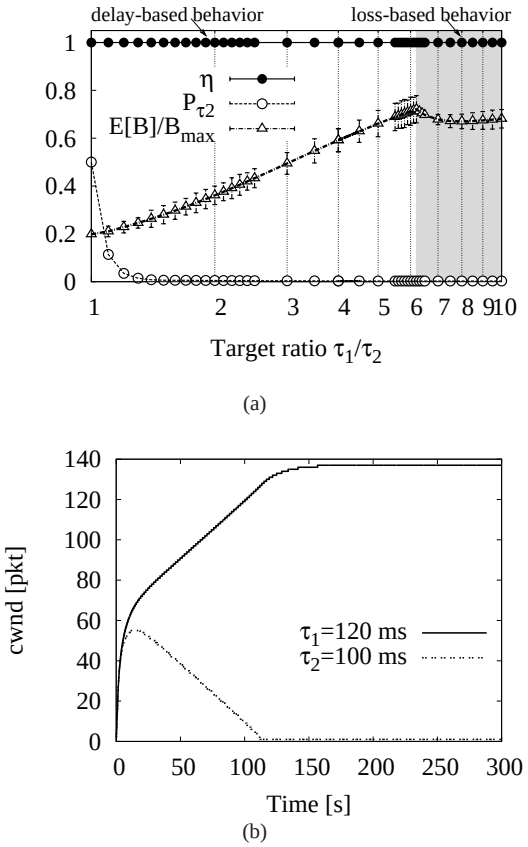


Figure 6: LEDBAT vs LEDBAT: Intra-protocol sensitivity analysis. (a) Impact of target heterogeneity on performances of two LEDBAT flows and (b) time evolution of the congestion window in case $\tau_1/\tau_2 = 1.2$. In both figures, 100ms$= \tau_2 \leq \tau_1$.

null delay links connected through an infinite switching capacity router, to which all ADSL modems are interconnected in a star topology. Unless otherwise stated, we set the buffer size to $B = 200$ full-size packets (that we have increased with respect to the buffer size in the previous flow-level simulation to better exacerbate performance difference), that given $C = 1\,\text{Mbps}$ correspond well to the range of bufferbloat delays observed via Netalyzr in real-world modems [23]. Access links also model one-way propagation delay, which is chosen uniformly at random in the interval $[0, 25]$ ms so that the average RTT delay of the swarm between two peers in the swarm is about $RTT = 25\,\text{ms}$. Unless otherwise stated, the target delay for the LEDBAT protocol is fixed at $\tau = 100\,\text{ms}$.

We implement two kind of P2P nodes, that model different application settings as far as peer preference for the congestion protocol used for *data transfer* in uplink is concerned. We assume that the lat-

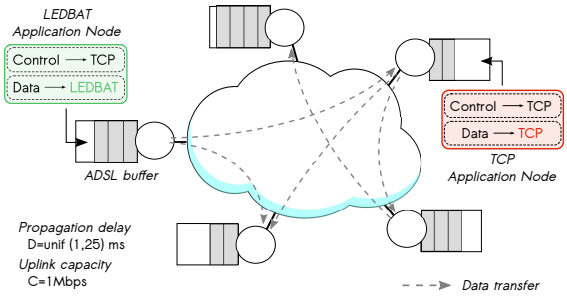


Figure 7: Synoptic of the swarm simulation model.

est application versions will by default initiate data transfers using LEDBAT on their uplink, but that they can accept incoming TCP data connections (i.e., similar to setting `bt.transp_disposition=5` in μTorrent). Older versions initiate TCP transfers, but accept incoming LEDBAT connections anyway (i.e., `bt.transp_disposition=10`). For the sake of simplicity, irrespectively of their *data transfer* settings, applications use TCP for the exchange of *control messages* (that do not send high volume of data). Hence, a different traffic mixture will possibly compete for ADSL access link capacity and buffer space.

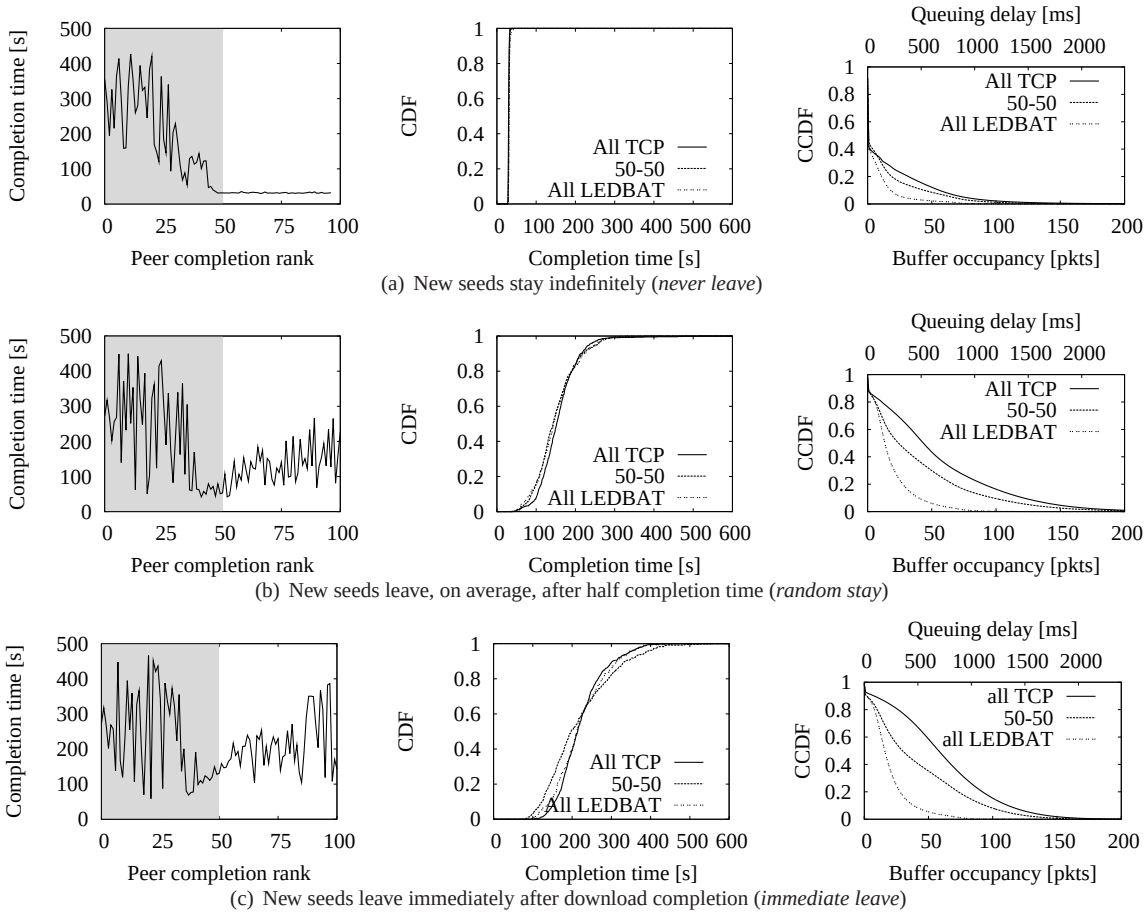
Notice that, while nodes are free to decide what congestion control flavor to use for their outbound connections, they have to comply to other peers settings as far as inbound data connections are concerned. We point out that this slightly differs from the μTorrent implementation [39], whose latest releases instead implement by default a dual-stack solution where (i) both TCP NewReno and LEDBAT connections are attempted in parallel (ii) the TCP NewReno connection is dropped in case the LEDBAT connection is successfully established, (iii) application-layer throttling of aggregated LEDBAT vs TCP bandwidth is in place. In practice, successful opening of a LEDBAT connection may depend on a number of factor (e.g., NAT traversal, different `bt.transp_disposition` configuration of the remote peer, availability of LEDBAT or legacy clients, etc.) that we prefer not to model (as their precise settings would be arbitrary and questionable anyway). Rather, we argue that this simple model, where peers have the freedom to choose the uplink protocol of their choice, is a reasonable connection management policy possibly available in the multitude of BitTorrent clients implementing LEDBAT, and need to be assessed as well. Similarly, we argue that a too detailed simulation model, e.g., including application-layer throttling of aggregated LEDBAT vs TCP bandwidth, would defeat the very same purpose of using a simulator. More faithful performance in the case of μTorrent are gathered via experiments in [39, 40], that is worth pointing out to be consistent with results we achieve in simulation.

5.1. Preliminary campaign

We carry out a preliminary campaign to further refine the P2P simulation scenario, where for simplicity, we limitedly consider TCP NewReno and LEDBAT protocol. Besides, from Sec. 4 we know LEDBAT, NICE and VEGAS to behave similarly with backlogged flows, so we can reasonably expect similarities to hold, at least to some extent, in the BitTorrent case as well. We simulate a mild flash-crowd scenario, in which at time t=0 the swarm is constituted by only one seed, and then 100 leechers join with exponentially distributed arrival times (mean rate 0.1 Hz). During each simulation, we discard the first 50 completion samples that happen during the transient period, and consider only the subsequent 50 completion times. Simulations end after the 100th user has completed its download, so that users beyond the 100th participate to swarming, but their performance are not accounted for. For each parameter settings, we repeat each simulation 10 times, so that statistics represent 500 individual torrent downloads per setting.

As far as the swarm population is concerned, we either consider *homogeneous swarms* (i.e., all TCP or all LEDBAT) or *heterogeneous swarms* where the population is equally split, on average, among LEDBAT and TCP peers (denoted in the following as 50-50). While in [39] we also explore different TCP vs LEDBAT ratios, we believe this split ratio to be the most relevant for our purpose. Indeed recent estimates of μTorrent permeation (60% in 2008 [43]) suggest that many legacy/TCP versions are still around. Additionally, a roughly equal share in terms of the traffic volume is confirmed by our measurement at multiple vantage point in Europe [10] and by Brahm Cohen own words [14]. As a side effect, as the population sets size are unbiased, the presentation of the results is also simpler.

As far as the peer and seed behavior is concerned, we consider three different scenarios, namely (i) never leave, (ii) random stay, and (iii) immediately leave. In the first one, peers *never leave* the system after becoming seeds, thus altruistically continuing to serve other leechers in a optimistic swarm configuration. In the second, more realistic, scenario newborn seeds stay in the system for a *random time*. Under this conditions, each time a peer leaves, a new leecher joins the swarm, so that the swarm size remains constant. Specifically, peers stay in the system after becoming seeds for an exponentially distributed time, with mean equal to half their



(a) New seeds stay indefinitely (*never leave*)

(b) New seeds leave, on average, after half completion time (*random stay*)

(c) New seeds leave immediately after download completion (*immediate leave*)

Figure 8: Swarm performance for different seeds holding time (never leave vs random stay vs immediate leave) and populations (homogeneous LEDBAT vs homogeneous TCP vs heterogeneous 50-50): completion time evolution and CDF, queue occupancy CCDF.

download time. On the one hand, this roughly models the BitTorrent netiquette to contribute to the system beyond the tit-for-tat; on the other hand, missing more detailed information of share ratio from real swarms, we decide to opt for a simple peer behavioral model, similar to other tricks that are common in the P2P literature[27]. Finally, a worst-case scenario is considered, in which selfish peers *immediately leave* the swarm after data completion, while new leechers join to maintain a constant population in the torrent swarm.

Simulation results of this preliminary campaign are reported in Fig. 8. Top plots of Fig. 8-(a) refer to scenarios where seeds never leave the system, middle plots of of Fig. 8-(b) to random stay scenario and bottom ones of Fig. 8-(c) to immediate leave case. Taking a single run as an example, plots on the left column depict how the completion time evolves during the simulation, ordered by peer completion rank (dark gray background represents the discarded initial transient period). If seeds stay forever, completion times shrink down to a point in which leechers are close to fully exploit their downlink capacity, which is no longer the case when seeds stay only for a finite time or leave as soon as they end the download.

Plots in the middle column of Fig. 8 report the completion time cumulative distribution (CDF) for the different populations. If peers never leave the system, no difference arises due to the congestion control algorithm: intuitively, as there is no resource hotspot, peers are able to download from many seeds at the same time, hence we do not expect congestion to play a major role in this case. Conversely, when seeds leave the system and are replaced by new leechers, resources become rare, translating into longer completion times.

Notice that our interest is especially on the *relative* completion time between peers with different congestion control flavors, more than on the torrent *absolute* completion time. Under this light, while performance

of homogeneous swarms are alike, completion time in random stay and immediate leave scenarios is affected by the specific congestion control mechanism adopted by peers in case of heterogeneous swarms.

Reasons why this happens can be better understood by looking at the queue size complementary cumulative distribution (CCDF) shown in the right column plots of Fig. 8 (gathered by sampling all uplink queues at 10 Hz). Notice indeed that uplink queue of LEDBAT peers is very similar in all scenarios, as LEDBAT tries not to exceed a target delay: deviation from target are due to TCP control connection sharing the same queue, and latecomer advantage [32]. On the contrary, TCP queues can grow long: in scenario (b), for about 20% of the cases queues exceed 100 packets, corresponding to more than a second of queuing delay considering full size packets (as reported in the top x-axis for reference). In the 50-50 case, queuing delay aggregates both LEDBAT and TCP uplink buffers, resulting in an intermediate average system queuing time (notice that in the 50-50 scenario, a further deviation from LEDBAT target is due to bursty uplink ACK traffic of TCP connections opened by other peers in the swarm). However, in heterogeneous scenarios, queues of individual peers are more influenced by their uplink protocols.

Interestingly indeed, in *heterogeneous swarms* (denoted with 50-50 in Fig. 8) the completion time behavior changes significantly according to peer congestion control preferences: more precisely, we report in Fig. 9-(a) the completion time CDF of LEDBAT vs TCP peers, gathering that LEDBAT peers have shorter download times (which holds for both the random stay and immediate leave scenarios). This is a counter-intuitive result, as we would not expect completion time to be tied to the congestion control used to handle chunks upload with the protocol of choice for the uplink. Intuitively, the completion time metric relates to the *downlink* performance of a peer, but are otherwise unrelated to the protocol a peer uses in *uplink*.

We suspect this unexpected phenomenon to arise due to (i) the coupling of the data vs control connection, associated to (ii) the very large size of ADSL buffers: while large buffers are beneficial to backlogged data connections, they can conversely harm BitTorrent signaling. Indeed, TCP control connection competes with either LEDBAT data traffic (that strive to keep a low extra delay on the access buffer) or TCP data traffic (that indiscriminately opens up the congestion window until loss occur). To prove this fact, we perform simulation with variable buffer size, whose results are reported in Fig. 9-(b) and confirm our intuition. As ADSL buffers are large, queuing delay of TCP peers can grow

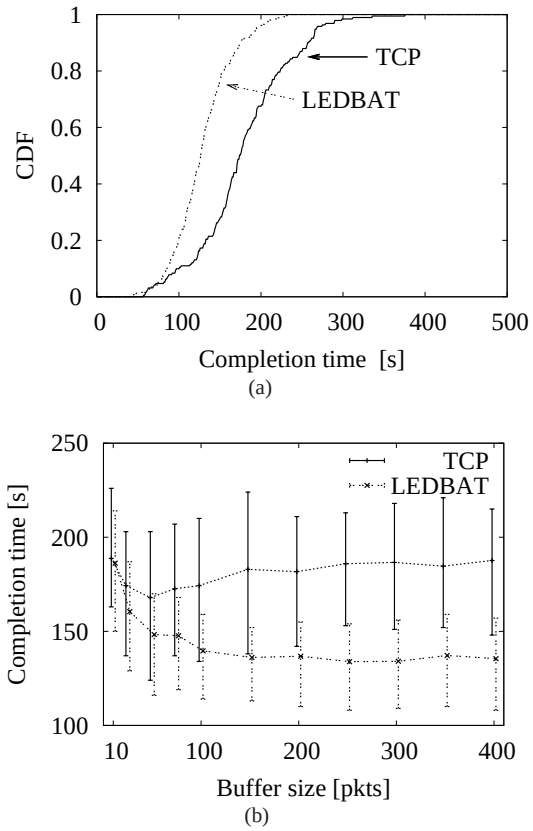


Figure 9: (a) Breakdown of completion time CDF according to the different peer population in the heterogeneous 50-50 scenario, for different seed holding times. (b) Swarm completion time (mean, 1st and 3rd quartiles) as a function of the buffer size B for the 50-50 random-stay scenario, for different peer population.

up to seconds (Fig. 8): hence, this possibly hampers the performance of TCP peers, whose control traffic is significantly slowed down by competing chunk upload to other peers and by ACK traffic of downloaded chunks. Conversely, the shorter queuing delay of LEDBAT peers lead to more responsive control connections, that opportunistically "steal" download slots from TCP peers (whose request rate is, as we just saw, slowed down due to self-induced congestion in the access link), as they are faster in filling the request buffer of other peers. This phenomenon is instead not observed in case of homogeneous TCP swarms, as all peers fairly compete against each other.

5.2. Relative protocol assessment

We now extend our investigation to other protocols than LEDBAT. For the sake of brevity, we fix the *random time* case, and consider both *homogeneous swarms*

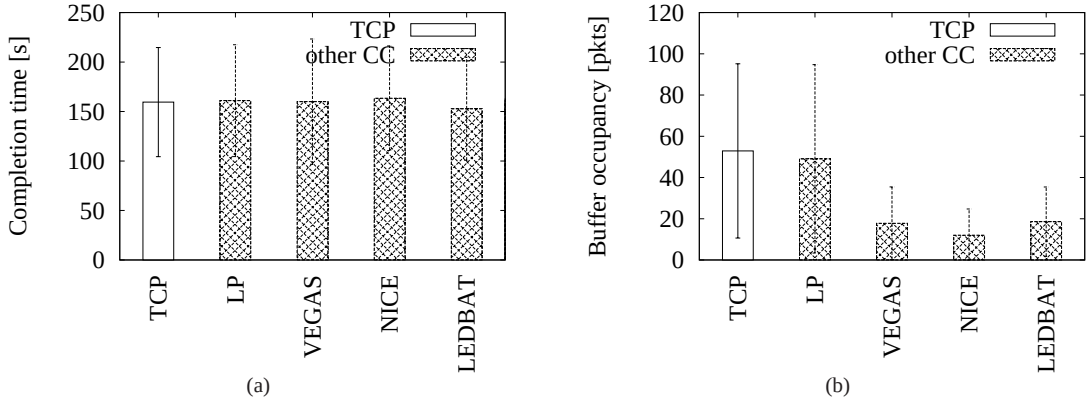


Figure 10: (a) Average and standard deviation of completion time and (b) average Buffer occupancy for different homogeneous swarms.

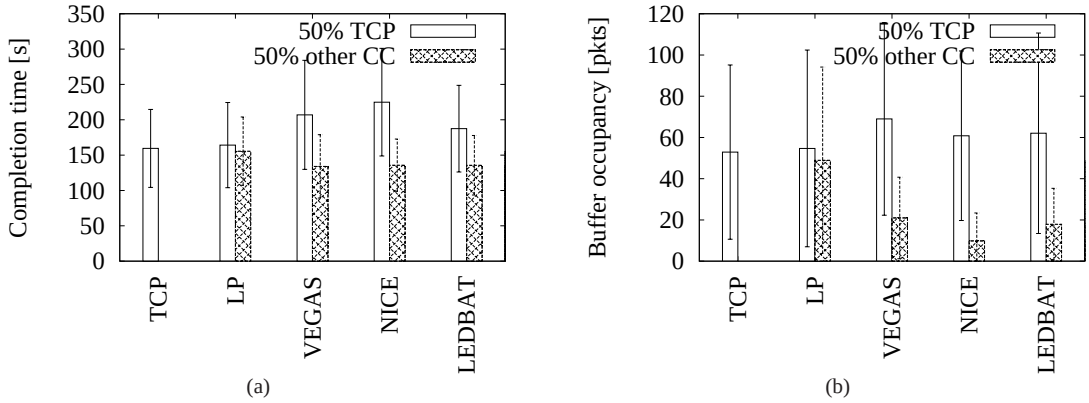


Figure 11: (a) Average completion time and (b) average Buffer occupancy for heterogeneous swarms (50%TCP-50%other CC).

(with TCP NewReno, VEGAS, LP, NICE and LEDBAT peers) and *heterogeneous swarms* (where half of the peers are TCP NewReno and half either VEGAS, LP, NICE or LEDBAT). This analysis is motivated by the fact that delay-based congestion control protocols are already available in the kernel of modern Operating Systems, so that is relatively easy for application developers to use them (i.e., TCP flavors can be specified as parameters in the socket API). Hence, in case comparable performance could be achieved under multiple protocols, this would allow to focus the development effort on other issues.

In case of homogeneous swarm, as expected from the preliminary simulation campaign, we have that completion time is roughly the same irrespectively of the congestion control used as Fig. 10-(a) shows. Also, as expected per the flow-level analysis in Sec. 4, we can see from Fig. 10-(b) that the average buffer occupancy is higher for TCP NewReno and LP than for VEGAS, NICE or LEDBAT.

In the case we have a mixed population of peers within the same swarm, Fig. 11-(a) represents completion time for the TCP half-swarm with a white bar, and the remaining half with dark shaded bar (for reference, we still report the all-TCP swarm as well). Results confirm that TCP NewReno peers always experience higher completion time with respect to the other peers – though difference is only minimal in the LP case. This can again be explained looking at the average buffer occupation plotted in Fig. 11-(b), that is higher for TCP NewReno than for VEGAS, NICE or LEDBAT peers.

Overall, we see that effects tied to the packet-level dynamics induced by the congestion control protocol in use by peers can have notable effects on the BitTorrent performance. This is in accordance with [17], that however limitedly focus on homogeneous TCP NewReno case.

Our results further show an unexpected advantage of

delay-based protocols, that reducing the delay experienced by control messages, can give competitive advantage with respect to bufferbloated TCP NewReno peers. This observation is coherent with our experimental findings [39], that further show the completion time to be linearly correlated with the queue size. Additionally, simulations reported in this section show these phenomena to hold for a set of delay-based protocols (namely, VEGAS, NICE and LEDBAT) whose behavior significantly deviates from TCP NewReno. Conversely, delay-based protocols like LP that still have additive-increase component, suffer similar queuing delays to TCP NewReno, so that they cannot gain any opportunistic advantage.

5.3. LEDBAT target heterogeneity

As we have previously seen (Sec. 4.2.2), LEDBAT possibly leads to QoS fairness problems in case of backlogged connections having heterogeneous targets. As LEDBAT has been originally been designed by BitTorrent, it is equally important to assess whether target heterogeneity leads to QoE unfairness[4] also in terms of the BitTorrent completion time. More precisely, we separately consider the case of mixed TCP and LEDBAT swarms (Sec. 5.3.1), and of swarms with heterogeneous LEDBAT settings (Sec. 5.3.2).

5.3.1. Inter-protocol case: TCP vs LEDBAT

As for the LEDBAT sensitivity analysis in the swarm-case, we argue that is not necessary to perform any simulation in the inter-protocol case. Implicitly, such a sensitivity was already shown in Fig. 9 where, instead of letting the target vary, we varied the buffer size.

Still, additional considerations are worth reporting. The results just shown testify that LEDBAT flows have a competitive advantage provided their queuing delay is smaller with respect to TCP NewReno peers – which is equivalently implied by either smaller target for a fixed buffer size, or larger buffer for a fixed target (as in Fig. 9).

Clearly, the queuing delay of LEDBAT peers grows proportionally to the target, so that the difference between LEDBAT and TCP NewReno performance reduce as τ grows. At the limit, τ exceeds the buffer size and LEDBAT becomes loss-based as TCP NewReno, so that completion time is the same under both protocols.

[4]Quoting the uTorrent post [26] that originally introduced the LEDBAT (then called uTP): "*Same performance is what users have come to expect from their BitTorrent application unless we can offer the same performance, then people will switch to a different BitTorrent client.*"

5.3.2. Intra-protocol case: LEDBAT vs LEDBAT

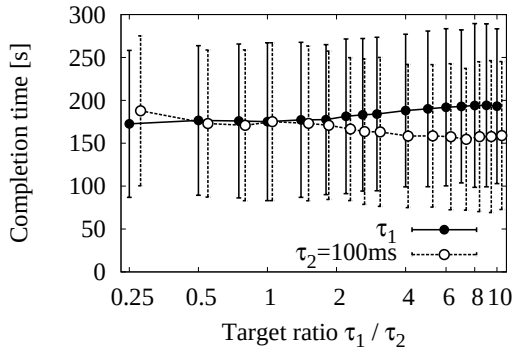
We now focus on the intra-protocol case, letting τ vary. Recall that from Sec. 4, as far as the backlogged flow viewpoint is concerned, even a slightly higher target can cause starvation of a compliant flow, which can be easily exploited by malicious users. We therefore want to study whether such competitive advantage persists under the swarm perspective.

The scenario we hypothesize in this case is one where LEDBAT has taken over TCP NewReno for P2P file-sharing in BitTorrent, so that all legacy clients now use the open-source LEDBAT API. Changing τ settings is however very easy for both application developers or even end-users. Hence, in case violating the mandatory target values specified by the RFC could provide, as in flow-level, to a competitive advantage, this would provide incentives to selfish users (to reduce their download time) or application developers (to gain a competitive advantage over other applications). We recall that, since the LEDBAT RFC only requires $\tau \leq 100$, target heterogeneity arise also in the case of RFC compliant implementations, and is thus of special interest.

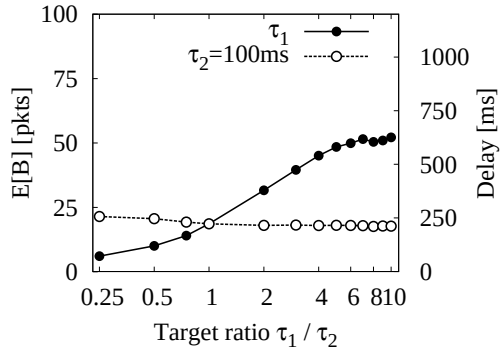
We consider an heterogeneous target among peers within the same swarm: in this scenario, half of the population employs the RFC-compliant target $\tau_2 = 100$ ms, while the second half uses a different target τ_1 with $\tau_1/\tau_2 \in [0.25, 10]$. Notice that with respect to our previous discussion, $\tau_1/\tau_2 \in [0.25, 1]$ correspond to users with RFC-compliant τ_1 settings, while $\tau_1/\tau_2 \in [1, 10]$ to malicious τ_1 settings. Fig. 12-(a) reports the average completion time of the two set of peers, in the *random stay* scenario. As we can see, for similar targets $\tau_1/\tau_2 \in [0.75, 1.5]$ peers achieve almost the same completion time on average.

Results further confirm that a lower target translates into a lower completion time: indeed, there is an advantage of selecting a lower target $\tau_1 < \tau_2$. When half of peers in the swarm use a smaller target $\tau_1 = 25$ ms than the maximum recommended value $\tau_2 = 100$ ms, their completion time decreases. As previously noted, this happen because peers with a lower target setting are more responsive in the control plane, as they strive to keep less data into the buffer. Thus, they face a lower self-induced congestion, as suggested also from the average buffer occupancy $E[B]$ reported in Fig. 12-(b), which positively impacts the user QoE of both BitTorrent as well as other interactive applications.

On the contrary, there appears to be no incentive in exceeding $\tau_2 > \tau_1$ the target value recommended by the LEDBAT RFC: for large target ratios the completion time increases by as much as 30% (additionally, the av-



(a)



(b)

Figure 12: Target sensitivity, swarm-level, LEDBAT intra-protocol case. (a) Average completion time and (b) average Buffer occupancy of peers with heterogeneous target within the same swarm.

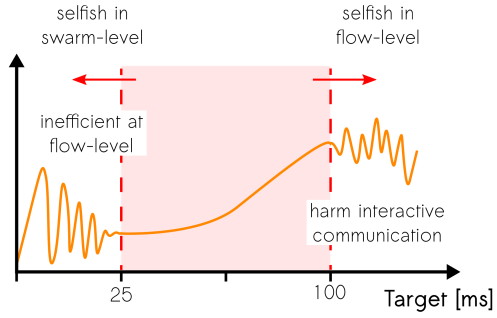


Figure 13: Synaptic of LEDBAT performances for varying target settings and workload models.

erage delay of peers using large target delays $\tau_1/\tau_2 > 5$ settles to 630 ms, which is more than the double of peers employing $\tau_2 = 100$ ms). This is an especially interesting findings, as settings that lead to opportunistic advantage in the swarm-case are completely opposite (and non-harmful) with respect to the previous flow-based case.

6. Conclusions

This work contrasts performance of a representative set of delay based congestion control protocols (namely, LEDBAT, NICE, VEGAS, and LP) from both a flow-level and a swarm-level perspectives. Our investigation focuses not only in a broad relative assessment of these protocols, but also in a detailed analysis of the most recent one (namely LEDBAT), that was not studied in such depth beforehand.

In the flow-level perspective, important since LEDBAT has been standardized at the IETF, we study classic QoS metric (e.g., buffer size, loss, efficiency and fairness) under a backlogged traffic model. In this scenario, we learn that LEDBAT has the lowest priority and is followed, in order of increasing priority, by NICE, VEGAS, and LP. It also follows that LEDBAT is the least intrusive protocol, leading to queuing delay that are shorter than NICE, VEGAS, and LP. Finally, LEDBAT efficiently exploits the bottleneck, so that its design goals are met.

However, in the intra-protocol case we also find that selfish users gain competitive advantages using *higher* targets τ, which can happen when settings are either compliant with, or in violation of, the RFC recommendations. Since even slightly higher targets may lead compliant flows to starvation, it follows that malicious users may gain an unfair advantage with by only minimally hurting its performance with respect to compliant users (at the same time, the same considerations apply to other congestion protocols, such as, e.g., TCP users with a faster rampup with respect to the RFC recommendations).

Yet, the above findings limitedly hold in the case of backlogged connections, for which we also analyze a more realistic swarm-level perspective, important since LEDBAT has been invented by BitTorrent to relieve bufferbloat of its users. In this scenario, we faithfully simulate BitTorrent dynamics at packet level and study how congestion control dynamics affect the most relevant QoE metric (i.e., the torrent completion time).

In this second case, we learn that LEDBAT congestion control can be beneficial to the torrent completion time. While surprising at first, this can be explained with the competitive advantage gained in the timely delivery of control plane messages. Interestingly, results confirm this phenomenon to hold across delay-based congestion control (e.g., including NICE and VEGAS)

as long as they keep the buffer size limited.

Most interestingly, competitive advantages in terms of completion time can be gathered in the swarm case by employing *lower* targets τ – a dual scenario with respect to flow-level. Hence, in the case of BitTorrent users, performance considerations inherently remove incentives to violate the LEDBAT RFC recommendation.

This duality is summarized with the help of Fig. 13. On the one hand, at flow-level small targets may not be feasible due to inefficient operation, while selfish users may resort to higher targets to gain competitive advantage. On the other hand, at swarm-level, selfish BitTorrent users gain competitive advantage by setting a lower target. From the above tradeoff it follows that, at least for BitTorrent, feasible operational points, that lay in the dark shared area of Fig. 13, can be found in practice.

Acknowledgement

This work has been carried out at LINCS `http://www.lincs.fr`, and funded by the FP7 mPlane project (grant agreement no. 318627).

References

[1] BitTorrent in ns2. `https://sites.google.com/site/koljaeger/bittorrent-simulation-in-ns-2`.

[2] LEDBAT Mailing List Archives. `http://www.ietf.org/mail-archive/web/ledbat`.

[3] LEDBAT `ns2` code. `http://perso.telecom-paristech.fr/~drossi/index.php?n=Software.LEDBAT`.

[4] J.S. Ahn, P.B. Danzig, Z. Liu, and L. Yan. Evaluation of TCP Vegas: emulation and experiment. In *ACM SIGCOMM Comp. Comm. Rev.*, volume 25, pages 185–195. ACM, 1995.

[5] Arnaud Legout and Nikitas Liogkas and Eddie Kohler and Lixia Zhang. Clustering and sharing incentives in bittorrent systems. In *Proc. of ACM SIGMETRICS'07*, San Diego, CA, Jun 2007.

[6] A. R. Bharambe, C. Herley, and V. N. Padmanabhan. Analyzing and Improving a BitTorrent Performance Mechanisms. In *25th IEEE Conference on Computer Communications (INFOCOM 2006)*, Barcelona, Spain, Apr 2006.

[7] R. Bindal, P. Cao, W. Chan, J. Medved, G. Suwala, T. Bates, and A. Zhang. Improving Traffic Locality in BitTorrent via Biased Neighbor Selection. Jul 2006.

[8] L.S. Brakmo, S.W. O'Malley, and L.L. Peterson. TCP Vegas: new techniques for congestion detection and avoidance. *ACM SIGCOMM Comp. Comm. Rev.*, 24(4):24–35, 1994.

[9] G. Carofiglio, L. Muscariello, D. Rossi, and C. Testa. A hands-on Assessment of Transport Protocols with Lower than Best Effort Priority. In *35th IEEE Local Computer Network (LCN 2010)*, Denver, CO, Oct 2010.

[10] G. Carofiglio, L. Muscariello, D. Rossi, C. Testa, and S. Valenti. Rethinking the Low Extra Delay Background Transport (LEDBAT) protocol. In *Elsevier Computer Networks (to appear)*, 2013.

[11] G. Carofiglio, L. Muscariello, D. Rossi, and S. Valenti. The quest for LEDBAT fairness. In *IEEE Global Communication (GLOBECOM 2010)*, Miami, FL, Dec 2010.

[12] Vint Cerf, Van Jacobson, Nick Weaver, and Jim Gettys. Bufferbloat: what's wrong with the internet? *Communications of the ACM*, 55(2):40–47, 2012.

[13] C. Chirichella and D. Rossi. To the moon and back: are internet bufferbloat delays really that large. In *IEEE INFOCOM Workshop on Traffic Measurement and Analysis (TMA 2013)*, Turin, Italy, Apr 2013.

[14] B. Cohen. How has BitTorrent as a protocol evolved over time. `http://www.quora.com/BitTorrent-protocol-company`.

[15] B. Cohen and A. Norberg. Correcting for clock drift in uTP and LEDBAT. In *Invited talk at 9th USENIX International Workshop on Peer-to-Peer Systems (IPTPS 2010)*, San Jose, CA, Apr 2010.

[16] L. DiCioccio, R. Teixeira, M. Mayl, and C. Kreibich. Probe and Pray: Using UPnP for Home Network Measurements. In *Passive and Active Measurement (PAM 2012)*, 2012.

[17] 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.

[18] S. Floyd and T. Henderson. RFC 2582: The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 2582, Apr 1999.

[19] Y. Gong, D. Rossi, and E. Leonardi. Modeling the interdependency of Low-priority Congestion Control and Active Queue Management. *ArXiv e-prints*, Mar 2013.

[20] Y. Gong, D. Rossi, C. Testa, S. Valenti, and D. Taht. Fighting the Bufferbloat: on the Coexistence of AQM and Low Priority Congestion Control. In *IEEE INFOCOM Workshop on Traffic Measurement and Analysis (TMA 2013)*, Turin, Italy, Apr 2013.

[21] M. Izal, G. Urvoy-Keller, E.W. Biersack, P.A. Felber, A. Al Hamra, and L. Garces-Erice. Dissecting bittorrent: Five months in a torrents lifetime. In *5th Passive and Active Measurement (PAM 2004)*, Antibes, France, Apr 2004.

[22] R. Jain. A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks. *ACM SIGCOMM Comp. Comm. Rev.*, 19(5):56–71, 1989.

[23] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson. Netalyzr: Illuminating the edge network. In *ACM Internet Measurement Conference (IMC 2010)*, Melbourne, Australia, Nov 2010.

[24] M. Kühlewind and S. Fisches. Evaluation of different decrease schemes for LEDBAT congestion control. *Energy-Aware Communications*, pages 112–123, 2011.

[25] A. Kuzmanovic and E.W. Knightly. TCP-LP: low-priority service via end-point congestion control. *IEEE/ACM Transactions on Networking (TON)*, 14(4):752, 2006.

[26] S. Morris. μTorrent release 1.9 alpha 13485. `http://forum.utorrent.com/viewtopic.php?pid=379206#p379206`, Dec 2008.

[27] F. Picconi and L. Massoulié. ISP friend or foe? making P2P live streaming ISP-aware. In *In Proc. of IEEE International Conference on Distributed Computing Systems (ICDCS'09)*, Montreal, Quebec, Canada, 2009. IEEE.

[28] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips. The BitTorrent p2p file-sharing system: Measurements and analysis. Ithaca, NY, Feb 2005.

[29] D. Qiu and R. Srikant. Modeling and performance analysis of BitTorrent-like peer-to-peer networks. *ACM SIGCOMM Comp. Comm. Rev.*, 34(4):367–378, 2004.

[30] Rao, A. and Legout, A. and Dabbous, W. Can realistic bittorrent experiments be performed on clusters? In *10th IEEE International Conference on Peer-to-Peer Computing (P2P 2010)*, Delf, The Netherlands, Aug 2010.

[31] D. Rossi, C. Testa, and S. Valenti. Yes, we LEDBAT: Playing with the new BitTorrent congestion control algorithm. In *11th*

Passive and Active Measurement (PAM 2010), Zurich, Switzerland, Apr 2010.

[32] D. Rossi, C. Testa, S. Valenti, and L. Muscariello. LEDBAT: the new BitTorrent congestion control protocol. In *19th IEEE International Conference on Computer Communications and Networks (ICCCN 2010)*, Zurich, Switzerland, Aug 2010.

[33] I. S. Ha, Rhee and L. Xu. CUBIC: A new TCP-friendly high-speed TCP variant. In *ACM SIGOPS Operating System Review*, New York, NY, Jul 2008.

[34] J. Schneider, J. Wagner, R. Winter, and H. Kolbe. Out of my Way – Evaluating Low Extra Delay Background Transport in an ADSL Acciess Network. In *22nd International Teletraffic Congress (ITC 2010)*, Amsterdam, The Netherlands, Sep 2010.

[35] S. Shalunov. Low Extra Delay Background Transport (LEDBAT). IETF Draft, Mar 2010.

[36] S. Shalunov, G. Hazel, J. Iyengar, and M. Kuehlewind. RFC 6817: Low Extra Delay Background Transport (LEDBAT). RFC 6817, Dec 2012.

[37] K. Tan, J. Song, Q. Zhang, and M. Sridharan. A compound TCP approach for high-speed and long distance networks. In *25th IEEE Conference on Computer Communications (INFOCOM 2006)*, Barcelona, Spain, Apr 2006.

[38] C. Testa and D. Rossi. The impact of uTP on BitTorrent completion time. In *11th IEEE Peer to Peer (P2P 2011)*, Kyoto, Japan, Sep 2011.

[39] C. Testa, D. Rossi, A. Rao, and A. Legout. Experimental Assessment of BitTorrent Completion Time in Heterogeneous TCP/uTP swarms. In *4th Traffic Measurement and Analysis (TMA) Workshop at 13th Passive and Active Measurement (PAM 2012)*, Wien, Austria, Mar 2012.

[40] C. Testa, D. Rossi, A. Rao, and A. Legout. Data Plane Throughput vs Control Plane Delay: Experimental Study of BitTorrent Performance . In *13th IEEE Peer to Peer (P2P 2013)*, Trento, Italy, Sep 2013.

[41] G. Urvoy-Keller and P. Michiardi. Impact of inner parameters and overlay structure on the performance of BitTorrent. In *25th IEEE Conference on Computer Communications (INFOCOM 2006)*, Barcelona, Spain, Apr 2006.

[42] A. Venkataramani, R. Kokku, and M. Dahlin. TCP Nice: A mechanism for background transfers. In *8th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2002)*, Boston, MA, Dec 2002.

[43] C. Zhang, P. Dhungel, D. Wu, and K. Ross. Unraveling the bittorrent ecosystem. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, (99), 2011.