LEDBAT: the new BitTorrent congestion control protocol

Dario Rossi, Claudio Testa, Silvio Valenti
TELECOM ParisTech
Paris, France
firstname.lastname@enst.fr

Luca Muscariello
Orange Labs
Paris, France
luca.muscariello@orange-ftgroup.com

***Abstract*—A few months ago, BitTorrent developers announced that the transfer of torrent data in the official client was about to switch to a new application-layer congestion-control protocol using UDP at the transport-layer. This announcement immediately raised an unmotivated buzz about a new, imminent congestion collapse of the whole Internet. As the new congestion control aims at offering a *lower* than best effort transport service, this reaction was not built on solid technical foundation. Nevertheless, a legitimate question remains: whether this new protocol is a necessary building block for future Internet applications, or whether it may result in an umpteenth addition to the already well populated world of Internet congestion control algorithms.**

To tackle this issue, we implement the novel congestion control algorithm and investigate its performance by means of packet-level simulations. Considering a simple bottleneck scenario, where the new BitTorrent competes against either TCP or other BitTorrent flows, we evaluate the fairness of resource share as well as the protocol efficiency. Our results show that the new protocol successfully meets some of its design goals, as for instance the efficiency one. At the same time, we also identify some potential fairness issues, that need to be dealt with. Finally, we point out that end-users will be the final judges of the new protocol: therefore, further research should evaluate the effects of its adoption on the performance of the applications ultimately relying on it.

***Keywords*-BitTorrent; Congestion Control; LEDBAT;**

I. Introduction

A few months ago, a post in the thread announcing the new μTorrent release 1.9-alpha-13485 in the BitTorrent developer forum [1] raised a lot of motivated interest as well as quite a few unmotivated buzz [2]. Not only the official BitTorrent client would no longer be open-source, but it was, above all, introducing a novel "micro transport protocol", a new application-layer protocol for data transfer implementing a innovative congestion-control algorithm and exploiting UDP at the transport layer.

Nevertheless, the main item retained was that BitTorrent would have switched its data transfer over UDP – which do not implement any kind of congestion control and is thus usually associated with *unresponsive* source. This fallacious interpretation raised serious concerns: as BitTorrent constitutes a significant portion of nowadays Internet traffic, its switchover to UDP was seen as the cause for the forthcoming collapse of the network. This "Internet meltdown" buzz rapidly flooded on the Internet, and only after an official reaction of BitTorrent followed by intense discussions, this climax started slowing down [2].

Yet, the buzz was not built on solid technical foundation: in fact, the original post [1] clearly stated that the design goal of the new protocol was to avoid *"killing your net connection – even if you do not set any rate limits,*" and to be able instead to "*detect problems very quickly and throttle back accordingly so that BitTorrent doesn't slow down the Internet connection and Gamers and VoIP users don't notice any problems.*" The inner working of this novel protocol is under discussion as BitTorrent Enhancement Proposals BEP29 [3], as well as IETF Low Extra Delay Background Transport (LEDBAT), whose first draft [4] has been accepted as a WG item in August 2009. To date, slight discrepancies exist between the two documents, mainly concerning parameter settings: as such, in the reminder of this work we will adhere to the LEDBAT flavor of the new protocol.

LEDBAT is described in [4] as a windowed protocol, governed by a linear controller designed to infer earlier than TCP the occurrence of congestion on a network path. Its congestion control algorithm is based on the one-way delay estimation: queuing delay is estimated as the difference between the instantaneous delay and a base delay, taken as the minimum delay over the previous observations. Whenever the sender detects a growing one-way delay, it infers that queue is building up and reacts by decreasing its sending rate. This way, LEDBAT reacts earlier than TCP, which instead has to wait for a packet loss event to detect congestion.

While LEDBAT design goals are sound, technical points have been raised by the scientific community participating to the LEDBAT working group, that ongoing discussion has not fully flattened yet. A legitimate question is whether the novel LEDBAT addition to the already well populated world of Internet congestion control algorithms is really necessary and motivated. LEDBAT-reluctants suggest indeed to consider already existing, and therefore more stable and better understood, algorithms for lower than best effort transport, such as TCP-NICE [5] or TCP-LP [6]. These comments, coupled with the move toward a closed source code, motivate the need for independent studies, so that claims concerning, e.g., the friendliness and efficiency of this new protocol, can be confirmed by the research community.

This work tackles precisely this issue, filling an important

gap in BitTorrent research: indeed, due to LEDBAT very recent evolution, with the exception of [7] where we analyze LEDBAT behavior through an active testbed methodology, previous work on BitTorrent [8], [9] focused on complementary aspects to those analyzed in this work. Rather that proposing any modification to LEDBAT, we aim at evaluating the draft specification [4] *as is*. Therefore, we implement and evaluate the simplest controller that strictly satisfies all the drafts requirements in `ns2` and evaluate the LEDBAT controller by means of packet-level simulations. The source code of our LEDBAT implementation is made available to the scientific community upon request.

Our results show that LEDBAT fulfills several of its design goals: it is able to efficiently exploit network capacity, to quickly yield to TCP or other higher priority traffic, and is by design robust to misconfiguration (e.g., fair competition with TCP in the worst case). However, we also point out that a late-comer advantage may arise between LEDBAT flows, with newly born connections absorbing all resources, bringing already started sessions to starvation. Although we show that this is mostly due to incorrect estimation of the base delay and can be easily fixed (e.g., by the use of slow-start), at the same time we believe that further effort is however needed to build a full relief picture of LEDBAT performance in actual scenarios.

Overall, LEDBAT may become a useful building block of the future Internet, and its deployment within a very popular P2P application such as BitTorrent already constitutes a very advantageous starting point. Yet, we underline that its success not only depends on its network friendliness, but also on the overall performance of applications relying on it. For instance issues like BitTorrent download time, interaction with peer selection strategies and tit-for-tat, will have a major impact on users experience and definitely deserve further investigation as well.

II. LEDBAT Overview

This section provides a basic overview of the LEDBAT draft [4]. To better understand the motivations behind LEDBAT, let us recall that the standard TCP congestion control needs losses to back off: this means that, under a drop-tail FIFO queuing discipline, TCP necessarily fills the buffer. As uplink devices of low-capacity home access networks can buffer up to hundreds of milliseconds, this may translate into poor performance of interactive applications (e.g., slow Web browsing and bad gaming/VoIP quality).

To avoid this drawback, LEDBAT implements a distributed congestion control mechanism, tailored for the transport of non-interactive traffic with lower than Best Effort (i.e., lower than TCP) priority, whose main design goals are:

- Saturate the bottleneck when no other traffic is present, but quickly yield to TCP and other UDP real-time traffic sharing the same bottleneck queue.
- Keep delay low when no other traffic is present, and add little to the queuing delays induced by TCP traffic.

```
on data_packet @ RX:
   remote_timestamp = data_packet.timestamp
   acknowledgement.delay =
       local_timestamp() - remote_timestamp

on acknowledgement @ TX:
   current_delay = acknowledgement.delay
   base_delay = min(base_delay, current_delay)
   queuing_delay = current_delay - base_delay
   off_target = TARGET - queuing_delay
   cwnd += GAIN * off_target / cwnd
```

Fig. 1. Pseudocode of the LEDBAT sender and receiver operations

- Operate well in drop-tail FIFO networks, but use explicit congestion notification (e.g., ECN) where available.

Intuitively, to saturate the bottleneck it is necessary that queue builds up: otherwise, when the queue is empty, at least sometimes no data is being transmitted and the link is under-exploited. At the same time, in order to operate friendly toward interactive applications, the queuing delay needs to be as low as possible: LEDBAT is therefore designed to introduce a non-zero *target* queuing delay.

In order to achieve this goal, LEDBAT follows a simple strategy. First of all, it exploits the ongoing data transfer to measure the one-way delay, from which it derives an estimate of the *queuing delay* on the forward path. Using one-way delay instead of round-trip time has the main advantage of preventing unrelated traffic on the backward path from interfering with data transmission. Second, it employs a *linear controller* to modulate the congestion window, and consequently the sending rate, according to the measured delay. LEDBAT operations can be summarized in the pseudocode in Fig. 1.

In the following, we first consider the two main components of the LEDBAT algorithm separately, and then we report some further considerations on the TCP-friendliness of the novel protocol.

A. Queuing Delay Estimate

Delay measurements are performed collaboratively by the sender and the receiver. The former puts a timestamp from its local clock in each packet. The latter, instead, calculates the one-way delay as the difference between its own local clock and the received timestamp, and communicates it back to the sender in the acknowledgements. The sender, besides, maintains a minimum of all observed delays, which represent the *base delay* used in queuing delay estimate.

To explain the rationale behind such technique, let us consider the different components of one-way delay: propagation, transmission, processing and queuing. Neglecting the processing delay, propagation and transmission delays are constant components, while the only variable component is the queuing delay. Intuitively, a packet which finds the queue empty (i.e., null queuing delay) will accurately estimate the constant portion of the one-way delay (i.e., the sum of propagation and transmission delays). This measure yields a minimum of

the delay, that will be stored as a reference: then, the queuing delay can be estimated as the difference between the current and the reference delays.

One-way delay measurements are notoriously difficult, especially for non-synchronized hosts. Yet the *variation* of delay with respect to the base delay, which is actually exploited by LEDBAT, is a much more robust metric. In particular, it does not suffer form timestamp errors such as fixed offsets and skews from the true time. For instance, the sender and receiver offsets could severely affect the absolute one-way delay estimate, but they happily cancel in the arithmetic difference `queuing_delay=current_delay-base_delay` (since both delays correspond in their turn to the difference of the receiver minus the sender delay). Further considerations about clock skew, noise filtering and route changes issues can be found in [4].

B. Controller Dynamics

A *proportional-integral-derivative (PID)* controller governs the dynamic of the congestion window in both the ramp-up and ramp-down phases. The controller continuously adapts the window to the estimated delay, in order to match the target delay. Clearly, when the queuing delay estimate is lower than the target (i.e., `off_target<0`) the sending rate has to increase, so that queuing delay reaches the target. Conversely, when the queuing delay estimate is higher than the target (i.e., `off_target>0`) the controller slows down the sending rate.

In Fig. 1 we observe that the controller itself is characterized by two parameters, the `TARGET` delay and the `GAIN` coefficient. The draft states that *"`TARGET` parameter MUST be set to 25 milliseconds and `GAIN` MUST be set so that max ramp up rate is the same as for TCP."*. The selection of a constant and moreover specific value for `TARGET` is quite controversial, as it is clear that non-compliant implementation with a larger target delay are advantaged and could introduce severe fairness issues (notice for instance that values in BEP29 [3] are larger than those specified in [4]). Concerning the second parameter, we set it to `GAIN=1/TARGET`, choice that we motivate in the next section.

We underline here a nice property of the PID controller: the window growth is directly proportional to the difference between the queuing delay estimate and the target `off_target`. In this way, when the queuing delay is close to the target, the controller response will be near zero, thus avoiding undesirable oscillations. Conversely, when the estimation is far from the target, the controller will increase the window faster and hopefully converge earlier.

C. TCP Friendliness Consideration

An important goal of LEDBAT concerns its ability to yield to TCP traffic when sharing the same bottleneck resources. LEDBAT should be able both to detect the traffic already present on links, and to yield quickly to newly incoming connections.

At the same time, LEDBAT must avoid starvation. In fact if LEDBAT always yielded to any kind of traffic, even to the one generated by non interactive application (e.g., a long-lived FTP transfer), the performance degradation perceived by users may convince them to simply revert to TCP-based transfers, regardless of LEDBAT potential advantages.

A first necessary condition for TCP friendliness, is that LEDBAT *should never ramp-up faster than TCP*. Since LEDBAT increases its congestion window of the largest amount when the delay estimate is zero (notice also that estimated delay can never be negative), by selecting `GAIN=1/TARGET` we guarantee that LEDBAT never ramps-up faster than TCP, as its maximum ramp-up speed is limited to one packet per RTT (i.e. like TCP in congestion avoidance).

A second requirement is that the delay-based LEDBAT congestion controller *should react earlier than loss-based TCP controller*: intuitively, if the former can ramp-down faster than loss-based connections ramp-up, it will yield to the latter. The draft states that LEDBAT should *"yield at precisely the same rate as TCP is ramping-up when the queuing delay is double the target"*. Again our choice of `GAIN=1/TARGET` fulfills this requirement: in fact, when the queuing delay is twice the target, LEDBAT will ramp-down at a rate equal to one packet per RTT, matching thus TCP congestion avoidance ramp-up speed.

A third final condition is that, *in case of loss, LEDBAT should behave like TCP does* (i.e., halve its congestion window). From all these considerations, one can derive that LEDBAT design follows a quite conservative approach, as in the worst case (when the queue estimation always equals zero) its most aggressive behavior simply degenerates into TCP.

III. LEDBAT Performance

In this section, we report results gathered with our implementation of the LEDBAT controller in the Network Simulator `ns2`: we start by illustrating some telling examples of the LEDBAT dynamics in simple cases, incrementally adding complexity to refine the picture later on. For details concerning the implementation, which is available as open source to the research community upon request, please refer to [10].

A. Reference scenario

As reference scenario, we consider a bottleneck link of capacity C Mbps and buffer size B packets. For the sake of simplicity, we assume that all transceivers adopt $P = 1500$ Bytes fixed-size packets. Traffic flows in a single direction, and acks are not delayed, dropped nor affected by cross-traffic on their return path. All flows have the same round trip time $RTT = 50\,ms$, half of which is due to the propagation and transmission delay components of the bottleneck link (i.e., a one-way base delay of 25 ms).

In this work we restrict our attention to a simple high-speed access scenarios, with a link of $C = 10\,Mbps$ capacity for downlink/uplink (an extended set of simulations, including an ADSL-like case is available in [10]), and different buffer sizes $B \in [10, 100] \subset \mathbb{N}$ packets. Notice that, once fixed the link capacity C and the packet size P, we can express the queuing delay `TARGET` in terms of either a time-lapse or

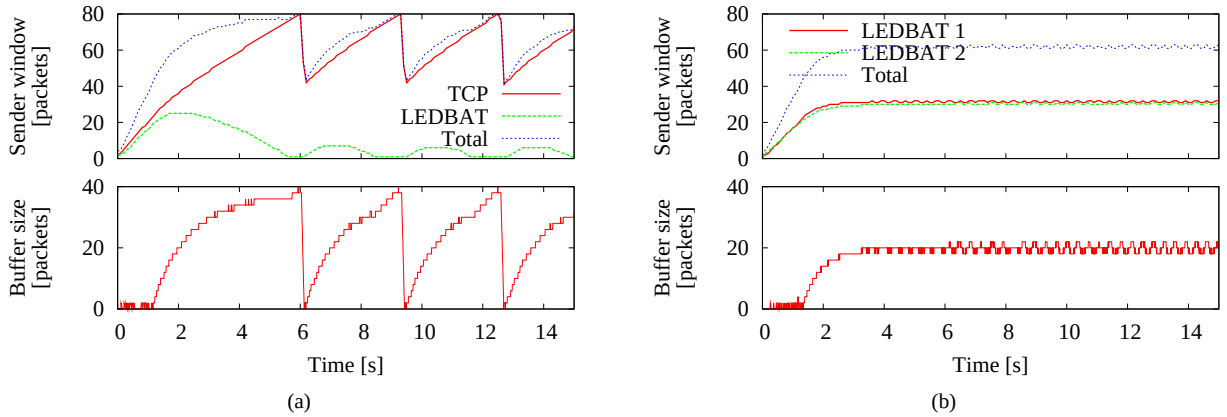

Fig. 2. Temporal evolution of the sender window (top) and of the queue size (bottom) for TCP-LEDBAT (a) and LEDBAT-LEDBAT interaction (b)

bytes (and packets). Denoting for short the TARGET as τ, in the following we will refer indifferently to the queuing delay in terms of time-lapse $\tau_T = 25\,\text{ms}$ or packets $\tau_P = \tau_T C/8P$ (with capacity expressed in kbps and packet size in bytes). For instance in our high-speed scenario, $\tau_T = 25\,\text{ms}$ corresponds to $\tau_P = 20.8$ packets. Thus, a buffer size of $B = 40$ packets, almost equal to the bandwidth-delay product, can accommodate twice as much queuing delay than the LEDBAT target τ.

As performance metrics, we consider the *fairness* and *efficiency* of the data transfer. For the former, we use Jain's fairness index F, which is defined as:

$$F = \frac{(\sum_{i=1}^{N} x_i)^2}{N \cdot \sum_{i=1}^{N} x_i^2} \tag{1}$$

where $\{x_i\}_{i=1}^{N}$ is the set of rates achieved by N flows sharing the same bottleneck resource. This index ranges between as maximum value of 1 (when the bandwidth is perfectly shared among the N flows) and a minimum of $1/N$ (in case one flow takes all the resource, leaving the others in starvation). Being LEDBAT a *lower* than best-effort protocol, we expect $F < 1$ when it competes with TCP, but $F \simeq 1$ when LEDBAT flows share the same bottleneck. Regarding efficiency, we consider the *link utilization* η metric, defined as the ratio of the overall link throughput (including headers) over the link capacity C.

B. Homogeneous Initial Conditions

Our investigation starts by considering a LEDBAT flow competing for the same bottleneck resources with either i) a TCP or ii) another LEDBAT flow. For the time being, we disable slow-start in both implementation as we are interested in the interaction of the LEDBAT PID the TCP AIMD controllers. We let both flows start at $t = 0$, when the queue is empty and no other traffic is present on the link, so that LEDBAT is able to accurately measure the base delay.

Fig. 2-(a) shows the temporal evolution of the LEDBAT and TCP windows (top) as well as of the queue length (bottom), with a buffer size of $B = 40$ packets. We recognize the usual TCP sawtooth behavior, which defines a number of cycles. During the initial ramp-up ($t < 2\,\text{s}$), LEDBAT and TCP windows grow *nearly* at the same speed of one packet per RTT. LEDBAT grows at its maximum speed because the available link capacity keeps the queue empty. As soon as queue builds up, the LEDBAT linear controller reacts accordingly by slowing down the increase of its sending rate, while TCP behavior remains instead unaltered. Soon after $t = 2\,\text{s}$, LEDBAT hits the $\tau_P = 20.8$ packet target, and halts the window growth, so presenting a flat sender window curve. TCP, instead, continues its additive increase, so that the queue keeps building up until the queuing delay exceeds the target: the LEDBAT controller, unlike TCP, reacts by decreasing its sending rate, finally reaching the minimum rate of one packet per RTT just before $t = 6\,\text{s}$.

Slightly afterwards, TCP causes a buffer overflow: consequently, TCP abruptly decreases its sending rate by halving is congestion window. The capacity drains the queue empty, giving thus start to a new cycle. In fact, LEDBAT detects the delay reduction and reacts by opening its window again. However, in this cycle TCP has an initial window size of about 40 packets, which means that it can create queuing sooner with respect to the previous cycle. Therefore, LEDBAT window growth is slower, the TARGET delay is hit earlier (at about $t = 7\,\text{s}$) and also the window shrink phase appears much shorter. When TCP is again the sole sender on the link, it increases its sending rate until a new loss happens, which in turn triggers the start of a new cycle.

Fig. 2-(a) confirms that, as LEDBAT reacts to congestion *earlier* than TCP by estimating the queuing delay, it is able to yield to TCP, which can *work undisturbed*. In fact, losses are due to the normal AIMD dynamic of TCP rather than to the LEDBAT-TCP interaction. Fairness in this case equals $F = 0.65$, with TCP transferring 6 times as much data with respect to LEDBAT during the same timeframe. Fig. 2-(a) also reports the sum of both TCP and LEDBAT sender

windows, which represents an estimate of the instantaneous link utilization. When TCP and LEDBAT coexist on the link, its *utilization increases* with respect to the case where TCP is alone – in the figure utilization increases by 16%, compared to the case where TCP is alone on the bottleneck.

Fig. 2-(b) shows a similar experiment, in which two LEDBAT sources start competing at $t = 0$ for the bottleneck resources. In this case, both senders employ a linear controller and are able to share resources fairly ($F > 0.99$) and efficiently (efficiency is only 0.7% less than in the Fig. 2-(a) case). As expected, once the delay target is reached, LEDBAT sources settle (since the offset from the target is zero, and so the controller response). Notice also that, since the two sources started together, they measured the same base delay at $t = 0$. Therefore, each sender independently settles when measuring a queuing delay equal to the target, thus it is actually responsible only for half of buffer occupancy.

C. Heterogeneous Initial Conditions

In this section we consider different start times for different sources. This implies that each sender will measure a different base delay at startup, gathering also a different estimate of the queuing delay. Indeed, assume that the first flow starts at time $t_1 = 0$, while the second starts at time $t_2 = t_1 + \Delta T$. In case the queuing delay at t_2 is not zero but equal to $t_Q(t_2)$, the second source will over-estimate the base delay $t_B(t_2)$ with respect to the one measured by the first source as $t_B(t_2) = t_B(t_1) + t_Q(t_2)$. So, the second source will set its target to a value higher than the first one, increasing the chances of a buffer overflow.

In case of interaction between LEDBAT and TCP, heterogeneity of initial conditions has a negligible impact. To convince of this, consider that, whenever LEDBAT starts first, it will be able to correctly estimate the base delay, and then to yield to TCP. Conversely if the LEDBAT flows starts later at t_2, it will over-estimate the base delay by the amount of TCP packets present in the buffer. This will in turn make LEDBAT under-estimate the queuing delay, resulting in an increased sending rate which will *anticipate* the first loss cycle. The system later evolves in a way similar to Fig. 2-(a), since after TCP halves its window, the capacity drains the queue empty and LEDBAT corrects its wrong base delay estimate. In subsequent cycles, LEDBAT will then dutifully yield to TCP.

By means of Fig. 3, we show, instead, that the interaction among LEDBAT flows is heavily influenced by the buffer size B and the start time gap ΔT. Each graph reports the sender window of two competing LEDBAT flows. In the top plot, obtained for $(\Delta T, B) = (2, 40)$, the second flows activates before the first one has started to create queuing. So, the two flows measure the same base delay and set the same target, which they together reach soon. But the first flow, having started before, attains a larger congestion windows, and actually owns the biggest share of the queue.

Instead, extremely different dynamics can be observed for $(\Delta T, B) = (10, 40)$ in the middle plot. In this case the second

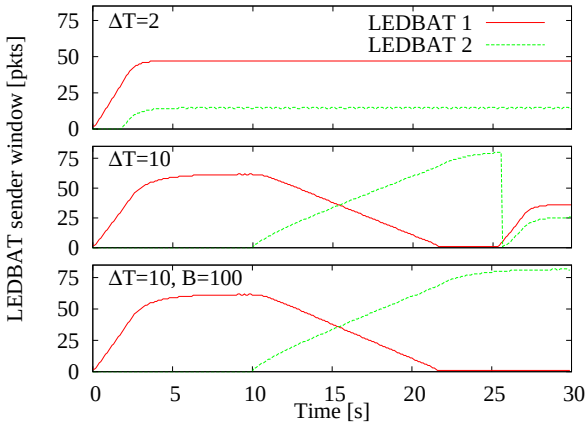

Fig. 3. LEDBAT vs LEDBAT: Time evolution of congestion window for different initial condition and late-comer advantage phenomenon

flow starts later enough to allow the first one to create some queuing delay, in particular a delay τ_T equal to its target. For this reason, the second flow wrongly senses a base delay equal to τ_T, and consequently sets its target to twice this value. Therefore, the newcomer starts increasing its rate right away, while the first one senses a growing queuing delay and begins to slowdown until, slightly after $t = 20$ s, it finally reaches the minimum rate.

Afterwards, dynamics depend on the specific buffer size. The middle plot shows a case where the buffer cannot accommodate the target queuing delay of the second flow (as B=40 $< 2\tau_P$=41.6). In fact, around $t = 25$ s, the second flow causes a loss on the bottleneck link and consequently drops its sending rate. Afterwards, similarly to the TCP case, the capacity drains the queue empty, providing the second flows the chance to correct its wrong base delay estimation. Subsequently, flows appear to share much more fairly the bottleneck capacity.

The bottom plot, depicts instead the effect of a larger $B = 100$ buffer, able to absorb the extra delay of the second flow. Basically, since no loss occurs, the second flows reaches its target and then settles, leaving the first flow in starvation. Unfortunately this unfair state persists for a possibly long time (namely, due to route changes considerations, the draft [4] imposes a reset of the base delay every 2-10 minutes).

D. Side Effects of Slow-Start

We have seen that competing LEDBAT flows may get stuck in an unfair state during a relatively long time. Yet, comparing the middle and bottom plots of Fig. 3, we notice that a loss event may partly re-establish the fairness. In fact, loss events resynchronizes the start of flows, possibly draining the queue empty and allowing each sender to gather correct measures of the base delay.

From this observation, the fairness problem could be solved by having each LEDBAT flow force a loss event at startup, so to gather a correct measure of the base delay. *Slow-start* represents a simple, even though intrusive, way of achieving this

effect. As the draft [4] deems slow-start as optional, we resort therefore to the standard TCP mechanism. In TCP, slow-start initially sets `ssthresh` to ∞, and performs an exponential window increase. Then, in case of loss, it sets `ssthresh = cwnd/2` and `cwnd=0`, and the process iterates until the window exceeds `ssthresh`.

Simulation results, only briefly reported here due to lack of space, but exhaustively presented in [10], confirm this intuition. We consider different capacity $C = \{2, 10\}$ and buffer $B = \{10, 50\}$ settings, and vary the start time of the second flow uniformly in $\Delta T = U(0, 10)$ s. For each setting we perform 100 simulations runs measuring fairness and efficiency among the two LEDBAT flows. Even in worst-case scenarios (i.e., when we have a behavior similar to the bottom plot of Fig. 3), the use of slow-start raises the LEDBAT vs LEDBAT fairness from $F = 0.53$ to $F = 0.99$.

Moreover, as slow-start generates losses events only at the beginning of each connection, we expect the loss rate to keep low. Simulation results show that the absolute amount of losses is limited to about 1 out of 100 packets in the worst case, with an average of 0.3% over all the scenarios considered. Nevertheless, a more comprehensive evaluation is definitively needed, considering the degradation this may have on VoIP/Gaming performance as well.

IV. Discussion and Conclusions

In this work, we report on the evaluation of LEDBAT, the novel BitTorrent congestion control algorithm for low-priority data transport. LEDBAT aims at being friendly and non-intrusive toward other protocols (such as TCP, VoIP and gaming), while being able to effectively exploit the available resources at the same time. By means of simulation, we illustrate some interesting aspects of the LEDBAT congestion window dynamics in simple scenarios. Summarizing:

- LEDBAT is able to achieve inter-protocol *friendliness* (i.e., yield to TCP) and at the same time to efficiently exploit the extra available resources.
- Inter-protocol *fairness* is maintained even in case of wrong parameter settings, in which case LEDBAT simply degenerates into TCP.
- The PID controller *alone* is not sufficient to guarantee intra-protocol fairness: in presence of large buffers, a late-comer advantage arises among LEDBAT flows.
- *Slow-start* offers an uncoordinated and distributed way of achieving intra-protocol fairness, allowing newcomer flows to correctly measure the base delay: interestingly, slow-start happens to be necessary for its beneficial side effect on fairness, more than for efficiency reasons.

Nevertheless these preliminary results convey just a limited view of the potential impact of a widespread adoption of LEDBAT in the Internet. First of all, simulation on a wider range of scenarios (e.g., heterogeneous RTT, multiple flows, impact on the QoE of VoIP traffic, comparison with other low-priority approaches, etc.) is needed to refine the picture. Also, performance of LEDBAT should be contrasted to those of related protocols sharing the same low-priority goal (e.g., TCP-LP [6] and TCP-NICE [5]).

Most important, LEDBAT underlying mechanism should be better analyzed, and a formal model of LEDBAT dynamics is required to back simulation evidence with more theoretical findings. Indeed, the *newcomer advantage* due to Additive-Increase Additive-Decrease (AIAD) control similar to the one adopted by LEDBAT has already been observed [11]. Under this light, the proposed slow-start solution can be seen as an uncoordinated mechanism that allows newcomers to introduce, by way of forced losses, a multiplicative decrease in the sender window of the already opened flow, which is known to re-establish fairness [11]. However, other techniques to ensure intra-protocol fairness shall be studied besides slow-start: specifically, as the ultimate goal of LEDBAT is low-priority, this technique should be as friendly as possible to other traffic – possibly trading efficiency for non-intrusiveness.

Besides, we stress that the success of LEDBAT will ultimately depend on its end-users, who will evaluate the novel protocol mainly in terms of the overall application performance. Under this light, the ability of LEDBAT to yield to interactive traffic is a clear incentive to protocol adoption, as it improves the QoE of interactive application that could otherwise suffer from self-congestion on the access link. Still, users may not welcome a degradation of the P2P performance caused by an excessive friendliness towards non-interactive traffic, such as P2P downloads of other users. As such, a wider-spectrum analysis of the impact of LEDBAT on BitTorrent remains, to date, a missing piece of the LEDBAT puzzle: important points that remain to date open concern e.g., the completion time of a torrent download under LEDBAT, or the impact of LEDBAT on BitTorrent peer selection mechanism.

References

[1] S. Morris. (2008, Dec.) μTorrent release 1.9 alpha 13485. [Online]. Available: http://forum.utorrent.com/viewtopic.php?pid=379206#p379206

[2] (2008, Dec.) BitTorrent Calls UDP Report "Utter Nonsense". [Online]. Available: http://tech.slashdot.org/article.pl?sid=08/12/01/2331257

[3] A. Norberg. uTorrent transport protocol. BitTorrent Enhancement Proposals. [Online]. Available: http://www.bittorrent.org/beps/bep_0029.html

[4] S. Shalunov, "Low extra delay background transport (ledbat)," IETF Draft, Mar. 2009.

[5] A. Venkataramani, R. Kokku, and M. Dahlin, "TCP Nice: a mechanism for background transfers," in *USENIX OSDI'02*, Boston, MA, US, Dec. 2002.

[6] A. Kuzmanovic and E. W. Knightly, "TCP-LP: low-priority service via end-point congestion control," *IEEE/ACM Trans. Netw.*, vol. 14, no. 4, Aug. 2006.

[7] *Yes, we LEDBAT: Playing with the new BitTorrent congestion control algorithm*, Zurich, Switzerland, April 2010.

[8] D. Qiu and R. Srikant, "Modeling and performance analysis of BitTorrent-like peer-to-peer networks," in *ACM SIGCOMM'04*, Portland, Oregon, USA, Aug. 2004.

[9] A. R. Bharambe, C. Herley, and V. N. Padmanabhan, "Analyzing and Improving a BitTorrent Networks Performance Mechanisms," in *IEEE INFOCOM'06*, Barcelona, Spain, Apr. 2006.

[10] D. Rossi, C. Testa, S. Valenti, P. Veglia, and L. Muscariello, "News from the Internet congestion control world," *ArXiv e-prints*, Aug. 2009.

[11] D.-M. Chiu and R. Jain, "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks," *Comput. Netw. ISDN Syst.*, vol. 17, no. 1, pp. 1–14, 1989.