Yes, we LEDBAT: Playing with the new BitTorrent congestion control algorithm

Dario Rossi, Claudio Testa, Silvio Valenti

Telecom ParisTech, Paris, France – first.last@enst.fr

Abstract. Since December 2008, the official BitTorrent client is using a new congestion-control protocol for data transfer, implemented at the application layer and built over UDP at the transport-layer: this new protocol undergoes the name of LEDBAT, for Low Extra Delay Background Transport.
In this paper, we study different flavors of the LEDBAT protocol, corresponding to different milestones in the BitTorrent software evolution, by means of an active testbed. Focusing on single flow scenario, we investigate emulated artificial network conditions, such as additional delay and capacity limitation. Then, in order to better grasp the potential impact of LEDBAT on the current Internet traffic, we consider a multiple flows scenario, and investigate the performance of a mixture of TCP and LEDBAT flows, so to better assess what "lower-than best effort" means in practice. Our results show that LEDBAT has already fulfilled some of its original design goals, though some issues still need to be addressed.

1 Introduction

Last December 2008, BitTorrent announced in the developer forum [1] that data transfer would move to UDP: shortly after this announcement, panic started spreading on popular websites [2], since the announcement directly led to the misbelief that BitTorrent plus UDP would equate with Internet meltdown. Yet, as already discussed at IETF [3] and later recognized on the Web [4], the BitTorrent development process embraces both ISP-friendliness (through AS-aware peer selection process) and TCP-friendliness (through a novel congestion control protocol for data transfer).

This work focuses precisely on this latter aspect of BitTorrent evolution: BitTorrent co-chairs an IETF Working Group on Low Extra Delay Background Transport (LEDBAT), which has very recently released its first draft document. To better understand the motivations behind LEDBAT, let us recall that the standard TCP congestion control mechanism needs losses to back off. Under a drop-tail FIFO queuing discipline, this means that TCP necessarily fills the buffer: as uplink devices of low-capacity home access networks have very large buffers, this may translate into poor performance of interactive applications (e.g., slow Web browsing and bad gaming/VoIP quality). LEDBAT attempts at avoiding this drawback, by implementing a distributed congestion control mechanism, tailored for the transport of non-interactive traffic with lower than Best Effort (i.e., TCP) priority. As stated in [5], among the main design goals of LEDBAT there are the ability (i) to minimize the extra delay it induces in the bottleneck, while (ii) saturating the available capacity at the same time. To fulfill these goals, LEDBAT has been designed as a windowed protocol (i) able to infer *earlier* than TCP the occurrence

of congestion, by estimating the queuing delay variation on the end-to-end path, (ii) to which it reacts by continuously modulating the congestion window growth/shrink by a proportional-integral-derivative (PID) controller.

The aim of this work is twofold. On the one hand, we target at understanding the performance of LEDBAT in a number of simple single flow scenarios, considering multiple versions of the official client so to better clutch its evolution. On the other hand, by means of multiple flows scenarios, we aim at gathering a preliminary understanding of the implication that a widespread adoption of LEDBAT could have on the current Internet landscape. We tackle the above issues with an active-measurement black-box study of the official BitTorrent client. Since LEDBAT is openly described in a IETF draft, the performance of the protocol could be assessed by means of simulations as we did indeed in [6]. Yet we still find active testbed experimentation extremely useful for several reasons. First, the BitTorrent implementation of the LEDBAT protocol may differ from any draft-compliant implementation by some design choices or parameter setting, that may have a deep impact on the protocol performance. Second, the most widespread LEDBAT implementation on the Internet will be the official BitTorrent version, rather than a legacy implementation, which motivates a direct evaluation of this client. Third, from our point of view, the analysis of proprietary applications by independent observers has the benefit of sheding light on the protocol inner workings. Finally real-world dynamics introduced by network devices are often much more complex than the synthetic ones that a simulation environment, although accurate, can reproduce.

With such an approach, we conduct a preliminary yet insightful evaluation of the protocol performance. First, we are able to report on the entire evolution of the protocol implementation, from the first (immature) version to the last (nearly stable) one. We point out that LEDBAT is able to, at least partly, fulfill its original design goals: under both controlled testbed and Internet experiments, LEDBAT avoids an uncontrolled queuing (unlike TCP), and is, under a range of conditions, able to saturate the available capacity (or, in case capacity is not saturated, this could be done by a simple tweaking of LEDBAT parameters). At the same time, we identify some open points regarding the protocol efficiency: for instance, TCP traffic on the "unrelated" backward path is able to slow down LEDBAT transmission on the forward path, whose capacity may be then significantly underutilized. Finally, we stress that the precise meaning of "lower-than best effort" should be carefully specified, as the mutual influence of TCP and LEDBAT traffic may significantly differ depending on the TCP flavor and settings as well.

2 Methodology and Preliminary Insights

For the investigation of the LEDBAT, we adopt an active-measurements black-box experimental approach, consisting in the analysis of the traffic generated by the BitTorrent client on different network scenarios. We run several versions of the new BitTorrent client on PCs equipped with dual-core processors featuring (i) unless otherwise stated, native installations of Windows XP or (ii) BitTorrent clients running on Linux using the `wine` Windows emulator. PCs are either (i) connected to the Internet through ISPs offering ADSL access, or (ii) in a local LAN testbed via Ethernet cards. In the first case we leave the default modem settings unchanged, while in the second one we disable

the interrupt coalescing feature and avoid the usage of jumbo frames. Moreover in the LAN testbed, the traffic is routed through a middlebox running a 2.6.28 Linux kernel, which acts also as network emulator by means of `netem`, in order to enforce artificial network conditions.

As formerly stated, in our experiments we consider both single flow and multiple flows scenarios. Single flow experiments are useful to understand the protocol performance under a range of different network conditions, while multiple flows experiments are needed to quantify the level of inter-protocol priority (e.g., with respect to TCP flows) and intra-protocol fairness (e.g., with respect to other LEDBAT flows) achieved by the distributed control algorithm. Under the classic BitTorrent terminology, every LEDBAT sender-receiver pair is a seeder-leecher pair, so that data transfer happens in a single direction. In case of multiple-flows experiments, every pair of actors belongs to a different torrent, so that no data exchange happens between different leechers.

We start by providing some insights on the BitTorrent evolution with the help of Fig. 1. Every picture refers to a different experiment, of which we report the first minute, corresponding to a different BitTorrent flavor. The seeder connects to the middlebox with a 100 Mbps Ethernet link, while between the middlebox and the leecher there is a 10 Mbps Ethernet bottleneck link. No other traffic is present on the bottleneck, and the one-way delay on the forward path is forced to 50 ms, to loosely emulate a scenario where two faraway peers with high speed Internet access (e.g., ADSL2+, FTTH or Ethernet) are connected together.

Pictures are arranged so that the macroscopic timescale of BitTorrent evolution also grows from left to right: Fig. 1-(a) shows, as a reference, the old open-source TCP-based client, while Fig. 1-(b) refers to the first closed-source version α_1, released December 2008. Then, Fig. 1-(c) depicts the α_2 version, released roughly at the same time of the first IETF draft [5] in March 2009. Finally, Fig. 1-(d) refers to the β_1 version, released after the draft was accepted as an official IETF WG item in August 2009.

The comparison of different versions of the protocol yields several interesting observations. First, notice that all versions analyzed correspond to important milestones in the development process of the protocol: thus, they provide a valuable perspective which highlights the flaws as well as the improvements of the subsequent steps of LEDBAT evolution. In particular, the α_1 version (which precedes the draft specification and motivates a black-box approach) was particularly instable and soon superseded. Moreover, from this study it emerges that the LEDBAT implementation is *constantly* evolving: as such, we believe that picking a single version, such as the most recent one, would limit the scope of our study.

For each flavor represented in Fig. 1, pictures depict the packet size on the y-axis, measured at the sender side, with time of the experiment running on the x-axis. As it can be seen, the application-layer segmentation policy is remarkably variable across different LEDBAT flavors. In contrast with TCP, which always transmits segments of maximum size, LEDBAT instead uses variable packet sizes. For instance, the α_1 implementation of Fig. 1-(b) mostly used small segments of about 350 bytes, transmitted at very high rate. Although this allows a finer tuning of the congestion window size, (e.g., likely to be more reactive to network condition), it definitively results in an unnecessary overhead. This segmentation policy is a bad choice for large transfers, and

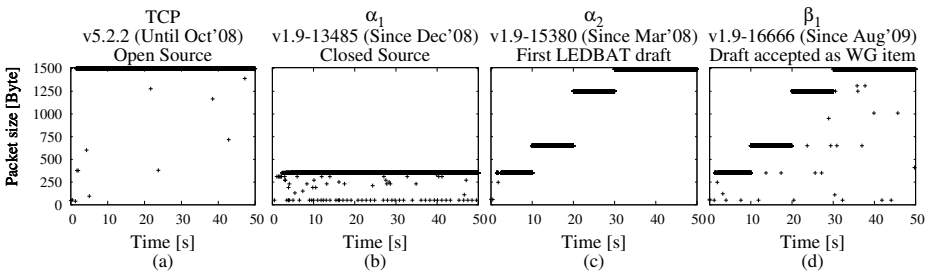

Fig. 1. The last few months of BitTorrent client evolution: Temporal plot of packet-level traces for different BitTorrent flavors, reporting packet size during the first minute of the transfer

was indeed soon dropped in favor of larger segment sizes. As can be gathered from Fig. 1-(c) and Fig. 1-(d), newer BitTorrent flavors start by segmenting data in small-size segments, and then gradually increase the segment size over time, rarely changing it once the full-payload segment size is reached. In case of α_2 flavor, we observe subsequent phases, about 10-seconds long, where only a single segment size is used: it takes about 40 seconds to the application-layer segmentation policy to settle to full-payload segment size. The β_1 flavor behaves similarly, although a wider range of segment sizes is employed during the whole experiment, probably to obtain a finer byte-wise control of the congestion window.

The corresponding time evolution of the achieved throughput, measured over 1 s time-windows is depicted in Fig. 2-(a), using a longer timeframe of about 4 minutes. We merely superpose the curves for the sake of comparison, but experiments have been independently performed. It can be seen that, shortly after achieving a sustained throughput of about 9 Mbps during about 50 seconds, the sending rate of the α_1 version suddenly drops, and about 2 minutes are necessary to recover from this starvation (this unstable behavior was observed under a wide range of conditions). In contrast, α_2 and β_1 achieve a lower but steady throughput, slightly above 4 and 7 Mbps respectively.

As a reference, we also report the throughput of a BitTorrent client using TCP running on the native Windows and Linux networking stacks under their default settings. The networking stack implementation and configuration dramatically impacts the protocol performance also in the TCP case. As reported in [7], in Windows XP, for transmission rates between 10-100 Mbps the default receive window is set to 17520 Bytes, whereas the default value of the Linux receive window (set in `net.ipv4.tcp_mem`) is about 6 times larger. Notice that in the Windows XP case, due to the 50 ms delay, the default value of the maximum window is not large enough to allow full saturation of the bottleneck pipe. This is an important, though not novel, observation on which we will come back later on Sec. 3.2.

3 Experimental Results

In this section, we start with simple single flow scenarios so to refine the performance pictures of the different flavors by testing the impact of varying network conditions.

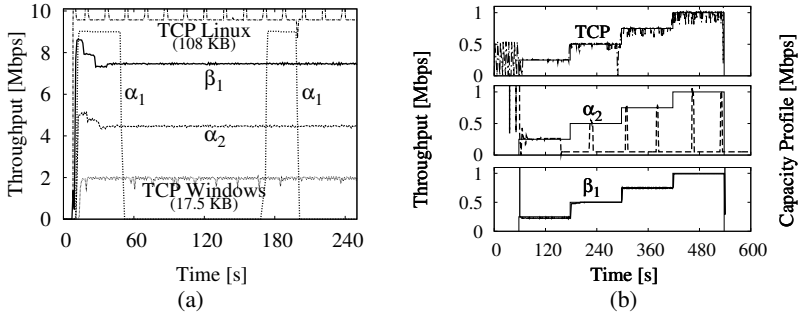

Fig. 2. Throughput for different flavors (a) without and (b) with bottleneck capacity limitations.

Among the several experiments conducted, we report the most relevant for our performance evaluation. In more detail, we consider (i) bottleneck capacity limitations, (ii) one-way delay impairment on either the forward or the backward path and (iii) different access technologies. We finally consider a scenario in which (iv) a single TCP flow interferes with LEDBAT on either the forward or the backward path, and (iv) multiple flows share the same bottleneck link, varying the ratio of LEDBAT and TCP flows so to better assess the protocols mutual influence.

3.1 Single flow: Bottleneck Capacity, Delay and Access Impact

Let us start by testing how BitTorrent copes with changing bottleneck capacity. We use a setup similar to the former experiment, but in this case the capacity of the link between the middlebox and the leecher is limited by means of the Hierarchical Token Bucket (HTB), available in `netem`. In more detail, we start at t=60 s to let LEDBAT throughput settle to a steady state, and then we turn on the HTB shaper. We initially tune it to 250 Kbps, increasing then the available capacity in steps of 250 Kbps every 2 minutes, as shown by the solid line capacity profile in Fig. 2-(b). A decreasing capacity profile yields to similar results and is thus not shown in the figure.

Time evolution of the throughput is reported for the new α_2, β_1 flavors as well as for the old TCP client. Flavor α_2 proves to be unable to quickly adapt to the changing link rate: it periodically enters a probing (or slow-start) phase, where it likely tries to infer network conditions by varying the segment size and sending rate. However, this phase is apparently unsuccessful and α_2 throughput starves (we did not observe such a starvation phenomenon for bottleneck larger than 1000 Kbps). This bug has been fixed by later releases: β_1 matches the available bandwidth, and moreover LEDBAT shows a much smoother curve than TCP. In this case, we may say that one of the LEDBAT design goals, namely, to efficiently exploit the available capacity, seems to be perfectly achieved.

Then, consider that the LEDBAT congestion control is based on a linear adaptation (i.e., growth/shrink) of the sender window to variations in the queuing delay on the forward data path (i.e., as inferred by the decrease/increase of the one-way delay,

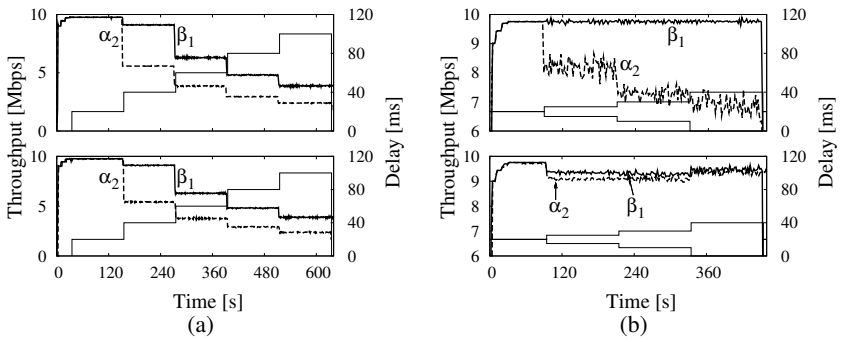

Fig. 3. Throughput evolution for different delay settings on the forward (top) and backward (bottom) path: (a) average delay increases over time, delay is equal for all packets (b) average delay is constant over time, delay variance increases over time.

with respect to the minimum measured one as reference): it is thus critical to assess its reaction to the measured one-way (OWD) delay. However, the sender response to queuing delay variations is nevertheless based on a closed-loop reaction with the receiver: therefore, we argue that the time instants at which the sender window growth/shrink decisions will be taken are also affected by the two-way delay, or Round Trip Time (RTT).

Thus, we setup and experiment in which we add an incremental OWD on either the forward (data) or backward (acknowledgement) paths. As before, after LEDBAT settles we increase the additional delay in steps of 20 ms every 2 minutes, for an RTT spanning on the 20–100 ms range as shown by the stepwise profile in Fig. 3-(a). The amount of OWD delay is added either to the forward path (top) or backward (bottom) path: in the former case, the delay incrementally adds to the OWD estimation performed by the sender so that it may directly affect the congestion control loop, while in the latter case it only delays the acknowledgement and may only indirectly affect the control loop.

As it can be seen from the comparison of the top and bottom plots of Fig. 3-(a), the overall effect on performance is the same: BitTorrent throughput decreases for increasing RTT, which is due to an upper bound of the receiver window (analogously to what seen before for TCP). With some back-of-the-envelope calculation based on the experimental results shown in Fig. 3-(a), one can gather that the receiver window limit has been increased from 20 full-payload segments of α_2 to 30 full-payload segment of β_1. While the picture shows that this limit may not be enough to fully utilize the link capacity (e.g., β_1 achieves about 4 Mbps throughput on a 10 Mbps link with RTT=100 ms), in practice it is not a severe constraint, as the capacity will likely be shared across several flows established with multiple peers of a BitTorrent swarm (or the receiver window limit could be increased).

In Fig. 3-(b) we instead investigate the effects of a variable OWD delay, that changes for each packet uniformly at random, with average OWD equal to 20 ms. In this case we keep the average constant but increase the delay *variance* every 2 minutes, so that

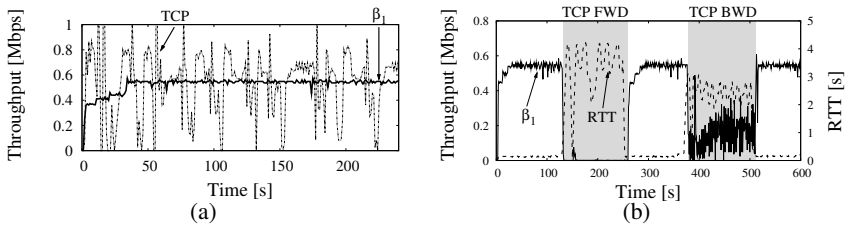

Fig. 4. Real Internet experiments: (a) different flavors and (b) interfering traffic.

the profile reports the minimum and maximum delays of the uniform distribution. The variable delay also implies that packet order is not guaranteed, because packets encountering a larger delay will be received later and thus out-of-order. Again, delay variance is enforced on either the forward (top) or backward (bottom) path. As it can be expected, LEDBAT is rather robust to a variable jitter on the backward path, where we observe only a minimal throughput reduction. Conversely, variance in the forward path has a much more pronounced performance impact: interestingly, α_2 throughput significantly drops, whereas β_1 performance is practically unchanged. This probably hints to the use of a more sophisticated noise filtering algorithm (e.g., that discards delay samples of out-of-order packets), although a more careful analysis is needed to support this assertion.

We finally perform an experiment using PCs connected through ADSL modems to the wild Internet. Thus, in this case we no longer have complete control over the network environment, but we still can assume that no congestion happens in the network and that the access link constitutes the capacity bottleneck. It can be seen from Fig. 4-(a) that in a realistic scenario, when the end-hosts only run LEDBAT, β_1 achieves a smooth throughput whose absolute value closely matches the nominal ADSL uplink capacity (640 Kbps). In contrast, TCP throughput is more fluctuating due to self-induced congestion, which causes fairly large queues before eventual losses occur. This confirms that the goal of avoiding self-induced congestion at the access is also met.

3.2 Multiple Flows

We now explore scenarios with several concurrent flows, starting with the simple one where a single LEDBAT flow interacts with a single TCP flow. Considering two PCs connected through ADSL modems to the wild Internet, Fig. 4-(b) reports an experiment where, during a single LEDBAT transfer, we alternate periods in which PCs generate no traffic other than LEDBAT, to periods (i.e., the gray ones) in which we superpose TCP traffic on either the forward or backward path.

The plot reports the time evolution of the LEDBAT throughput as well as the RTT delay measured by ICMP (as a rough estimation of the queue size seen by LEDBAT). During the silence periods (0–120 s and 240–360 s), as bottleneck is placed at the edge of the network, LEDBAT is able to efficiently exploit the link rate. As soon as a backlogged TCP transfer is started on the forward path (120–240 s), LEDBAT congestion control correctly puts the traffic in low priority. Notice that in this case, ICMP reports

Table 1. Efficiency and Fairness between multiple TCP and LEDBAT flows

		TCP$_W$, LEDBAT β_1							TCP$_L$, LEDBAT β_1						
TCP	LEDBAT	$\%_1$	$\%_2$	$\%_3$	$\%_4$	η	Fairness	RTX%	$\%_1$	$\%_2$	$\%_3$	$\%_4$	η	Fairness	RTX%
4	0	0.25	0.25	0.25	0.25	0.67	1.00	5e-4	0.25	0.25	0.25	0.25	0.98	1.00	0.06
3	1	0.14	0.14	0.14	0.57	0.94	0.64	-	0.35	0.32	0.32	0.00	0.98	0.75	0.14
2	2	0.10	0.10	0.40	0.40	0.93	0.74	-	0.43	0.51	0.03	0.03	0.98	0.56	4e-3
1	3	0.08	0.31	0.31	0.31	0.92	0.87	-	0.87	0.04	0.04	0.05	0.98	0.33	-
0	4	0.25	0.27	0.24	0.24	0.96	1.00	-	0.25	0.27	0.24	0.24	0.96	1.00	-

that a fairly large queue of TCP data packets builds up in the ADSL line (roughly 4 seconds, corresponding to about 300 KB of buffer space for the nominal ADSL rate). Conversely, whenever the backlogged TCP transfer is started on the backward path (360–480 s), LEDBAT transfer on the forward direction should only be minimally affected by the amount of acknowledgement TCP traffic flowing in the forward direction. However, as it can be seen from Fig. 4-(b), the LEDBAT throughput drastically drops, further exhibiting very wide fluctuations (notice also that the ADSL modem buffer space of the receiver appears to be smaller, as the RTT is shorter). Notice that in this case, LEDBAT forward data path shares the link capacity only with TCP acknowledgements, which account for a very low, but likely very bursty, throughput: this may led LEDBAT into a messy queuing delay estimate, and as a result, the uplink capacity of the device is heavily underutilized (about 74% of wasted resources).

We finally perform experiments to analyze the interaction of several flows. In this case, we setup several torrents, one for every different LEDBAT seeder-leecher pair, so that no data exchange happens between leechers of different pairs. Thus, flows are independent at the application layer, though their are dependent at the transport layer, as they share the same physical 10 Mbps RTT=50 ms bottleneck.

We consider a fixed number of F=4 flows, and vary the number of TCP and LEDBAT-β_1 connections to explore their mutual influence. All flows start at time $t = 0$, experiments last 10 minutes and results refer to the last 9 minutes of the experiment. We generate TCP traffic using Linux (so that we can reliably gather retransmission statistics using `netstat`), setting the congestion control flavor to NewReno. We perform two set of experiments, using either the Windows or Linux defaults values for the maximum receiver windows as early stressed in Fig. 2-(a): in our setup, the Windows-like TCP settings (TCP$_W$) are thus less aggressive than Linux ones (TCP$_L$).

For each experiment, we evaluate user-centric performance by means of the breakdown of the resources acquired by each flow, while we express network-centric performance in terms of the link utilization η. To further quantify the protocol mutual influence, we use the Jain's fairness index of the flows throughput and evaluate the percentage of TCP retransmissions (RTX). Results are reported in Tab. 1, with Windows and Linux settings on the left and right respectively. Comparing the two table portions, we argue that the exact meaning of "low-priority" may be fuzzy in the real-world. Indeed, while LEDBAT-β_1 is lower priority than an "aggressive" TCP, it may be competing more fairly against a more gentle set of parameters, thus being at least as high priority as TCP. In fact while LEDBAT is practically starved by TCP$_L$, LEDBAT is able to achieve a slightly higher priority than TCP$_W$. Although we recognize

that results may change using more realistic and heterogeneous network scenarios, or using the real Windows stack instead of simply emulating its settings, we believe that an important point remains open: i.e., the precise meaning of "lower than best effort", as the mutual influence of TCP and LEDBAT traffic may significantly differ depending on the TCP flavor as well.

4 Related work

Two bodies of work are related to this study. On the one hand, BitTorrent has been studied by means of theoretical analysis [8], simulation [6,9,10] or measurements [11]. On the other hand, there is a large literature on Internet congestion control that use either on fields measurement [12–14], or simulation and modeling [15–20]. Due to BitTorrent very recent evolution, with the exception [6], where we study LEDBAT by means of simulation, previous work on BitTorrent [8–11] focused on complementary aspects to those analyzed in this work. In [8] a fluid model is used to determine the average download time of a single file. Simulation has instead been used in [9] to propose incentive mechanism to avoid free-riding and in [10] to assess the performance of a locality-aware peer selection strategy. Finally, measurements study [11] analyzes the log of a BitTorrent tracker, examining flash-crowd effect, popularity and download speed of a single file. Congestion control work closer to our adopts a black-box experimental measurements approach to unveil proprietary algorithms of, e.g., Skype [12,13] or P2P-TV applications [14]. More precisely, [12, 14] analyzes system reaction to emulated network conditions, whereas [13] investigates the bottleneck share of multiple flows. Finally, relevant work has been devoted to the design of lower-than best effort protocols similar to LEDBAT, as for instance [17–20].

5 Conclusions

This paper presented an experimental evaluation of LEDBAT, the novel BitTorrent congestion control protocol. Single-flow experiments in a controlled environment show some of the fallacies of earlier LEDBAT flavors (e.g., instability, small packets overkill, starvation at low throughput, tuning of maximum receiver windows, wrong estimate of one-way delay in case of packet reordering, etc.), that have been addressed by the latest release. Experiments in a real Internet environment, instead, show that, although LEDBAT seems a promising protocol (e.g., achieving a much smoother throughput and keeping thus the delay on the link low), some issues still need to be worked out (e.g., performance in case of reverse path traffic). Finally, multiple-flows experiments show that "low-priority" meaning significantly varies depending on the TCP settings as well.

This work constitutes a first step toward the analysis LEDBAT performance. More effort is indeed needed to build a full relief picture of the LEDBAT impact on other interactive applications (e.g., VoIP, gaming), explicitly taking into account the QoE resulting from their interaction. Also, the methodology could be refined by, e.g., instrumenting the Linux kernel to measure the queue size, or by inferring the OWD measured by LEDBAT by sniffing traffic at both the sender and receiver, etc. Finally, the boundaries of the investigation could be widened by taking into account the effects of LEDBAT

adoption on the BitTorrent P2P system itself, as for instance LEDBAT interaction with throughput based peer-selection mechanism, or its impact on files download time.

Acknowledgement

This work has been funded by the Celtic project TRANS.

References

1. Morris, S.: μTorrent release 1.9 alpha 13485. http://forum.utorrent.com/viewtopic.php?pid=379206#p379206 (Dec 2008)
2. Bennett, R.: The next Internet meltdown. http://www.theregister.co.uk/2008/12/01/richard_bennett_utorrent_udp (Dec 2008)
3. Shalunov, S., Klinker, E.: Users want P2P, we make it work. In: IETF P2P Infrastructure Workshop. (May 2008)
4. : BitTorrent Calls UDP Report "Utter Nonsense". http://tech.slashdot.org/article.pl?sid=08/12/01/2331257 (Dec 2008)
5. Shalunov, S.: Low extra delay background transport (ledbat). IETF Draft (Mar 2009)
6. Rossi, D., Testa, C., Valenti, S., Veglia, P., Muscariello, L.: News from the internet congestion control world. Technical Report (Aug 2009)
7. Center, M.W.D.: Tcp receive window size and window scaling. http://msdn.microsoft.com/en-us/library/ms819736.aspx
8. Qiu, D., Srikant, R.: Modeling and performance analysis of BitTorrent-like peer-to-peer networks. In: ACM SIGCOMM'04, Portland, Oregon, USA (Aug 2004)
9. Bharambe, A.R., Herley, C., Padmanabhan, V.N.: Analyzing and Improving a BitTorrent Networks Performance Mechanisms. In: IEEE INFOCOM'06, Barcelona, Spain (Apr 2006)
10. Bindal, R., Cao, P., Chan, W., Medved, J., Suwala, G., Bates, T., Zhang, A.: Improving Traffic Locality in BitTorrent via Biased Neighbor Selection. In: IEEE ICDCS '06, Lisboa, Portugal (Jul 2006)
11. Izal, M., Urvoy-Keller, G., Biersack, E.W., Felber, P., Al Hamra, A., Garcés-Erice, L.: Dissecting BitTorrent: Five Months in a Torrent's Lifetime. In: Passive and Active Network Measurement (PAM), Antibes, France (Apr 2004)
12. Bonfiglio, D., Mellia, M., Meo, M., Rossi, D.: Detailed Analysis of Skype Traffic. IEEE Transaction on Multimedia **11**(1) (Jan 2009)
13. De Cicco, L., Mascolo, S., Palmisano, V.: Skype video responsiveness to bandwidth variations. In: ACM NOSSDAV '08, Braunschweig, Germany (May 2008)
14. Alessandria, E., Gallo, M., Leonardi, E., Mellia, M., Meo, M.: P2P-TV Systems under Adverse Network Conditions: A Measurement Study. In: IEEE INFOCOM'09. (Apr 2009)
15. Padhye, J., Firoiu, V., Towsley, D., Kurose, J.: Modeling TCP throughput: a simple model and its empirical validation. ACM SIGCOMM Comp. Comm. Rev. **28**(4) (Oct 1998)
16. Brakmo, L., O'Malley, S., Peterson, L.: TCP Vegas: New techniques for congestion detection and avoidance. In: ACM SIGCOMM'94, London, UK (Aug 1994)
17. Venkataramani, A., Kokku, R., Dahlin, M.: TCP Nice: a mechanism for background transfers. In: USENIX OSDI'02, Boston, MA, US (Dec 2002)
18. Kuzmanovic, A., Knightly, E.: TCP-LP: low-priority service via end-point congestion control. IEEE/ACM Transaction on Networking **14**(4) (Aug 2006)
19. Liu, S., Vojnovic, M., Gunawardena, D.: Competitive and Considerate Congestion Control for Bulk Data Transfers. In: IWQoS'07, Evaston, IL, US (Jun 2007)
20. Key, P., Massoulié, L., Wang, B.: Emulating low-priority transport at the application layer: a background transfer service. In: ACM SIGMETRICS'04, New York, NY, USA (Jan 2004)