Fighting the bufferbloat: on the coexistence of AQM and low priority congestion control

Y. Gong, D. Rossi, C. Testa, S. Valenti
Telecom ParisTech, Paris, France
first.last@enst.fr

M. D. Täht
bufferbloat.net
dave.taht@gmail.com

***Abstract*—Nowadays, due to excessive queuing, delays on the Internet can grow longer than several round trips between the Moon and the Earth – for which the "bufferbloat" term was recently coined. Some point to active queue management (AQM) as the solution. Others propose end-to-end low-priority congestion control techniques (LPCC). Under both approaches, promising advances have been made in recent times: notable examples are CoDel for AQM, and LEDBAT for LPCC. In this paper, we warn of a potentially fateful interaction when AQM and LPCC techniques are combined: namely (i) AQM resets the relative level of priority between best effort and low-priority congestion control protocols; (ii) while reprioritization generally equalizes the priority of LPCC and TCP, we also find that some AQM settings may actually lead best effort TCP to starvation. By an extended set of experiments conducted on both controlled testbeds and on the Internet, we show the problem to hold in the real world for all tested combination of AQM policies and LPCC protocols. To further validate the generality of our findings, we complement our experiments with packet-level simulation, to cover cases of other popular AQM and LPCC that are not available in the Linux kernel. To promote cross-comparison, we make our scripts and dataset available to the research community.**

***Index Terms*—Bufferbloat, AQM, Scavenger protocol, Experiments, Simulation**

I. Introduction

Internet delays are now as common as they are maddening [12]. The root cause for these delays can be identified with excess buffering a.k.a. bufferbloat" inside a network. Although this is nothing new [9], the situation has deteriorated in recent years due to mainly two facts: (i) TCP loss-based design, that forces the bottleneck buffer to fill before the sender reduces its rate and (ii) relatively large memories in front of low-capacity ADSL and cable links that can translate into many seconds of queuing delay [14].

Some point towards *local active queue management (AQM)* techniques (i.e., affecting the scheduling and discard of packets in the buffer differently from a traditional FIFO discipline) as the ultimate solution to reduce queuing delay. Others prefer another direction: the engineering of *end-to-end flow and congestion control (CC)* alternatives to best effort TCP, and specifically aiming at lower than best effort priority. In this work, we focus on the coexistence of best effort TCP CC and Low Priority CC (LPCC) transiting a bottleneck link governed by AQM. In the rest of this section we explain the reasons supporting the relevance and timeliness of this goal.

AQM is not a new research field, with numerous techniques proposed over the years such as RED [11], SFQ [17], DRR [25], Choke [20] and, very recently, CoDel [18]. Yet, despite numerous AQM proposals, they have so far encountered limited adoption. The difficulties in tuning RED [10] are well known, and the computational cost of Fair Queuing was, back in the 90s, considered to be prohibitive (see [12] for an historical perspective). The situation has however started to change, with operators worldwide implementing AQM policies in the upstream of the ADSL modem (e.g., in France, Free implements SFQ since 2005 [4]) to improve the quality of user experience.

Similarly, research in the CC domain has produced many proposals for low-priority (or background) transfers, such as NICE [26], TPC-LP [15] or, more recently, LEDBAT [24]. While TCP-LP has been around in the Linux kernel for about a decade, it has seldom been used[1]. However, ignited by the ease of application-layer deployment, scavenging CC services are now becoming popular: examples of this trend are represented by Picasa's background upload option and the adoption of a LPCC by BitTorrent. Indeed, BitTorrent recently abandoned TCP in favor of LEDBAT, a "low extra delay background transport" protocol implemented at the application-layer over a UDP framing[2]. Quoting B. Cohen, LEDBAT is now the bulk of all BitTorrent traffic, [...] most consumer ISPs have seen the majority of their upload traffic switch to a UDP-based protocol" [5].

It is out of the scope of this paper to provide an overview of AQM and CC research, for which we point the reader to the above sources. Yet, it is worth to highlight an interesting similarity between the most recent approaches of either class, namely the CoDel AQM and the LEDBAT CC, as both aim at explicitly controlling queuing delay. They both employ a *target delay* parameter upon which dropping decisions or congestion window modifications are based respectively.

A. A motivating example

Interestingly, while the underlying ideas and knobs that AQM and LPCC can exploit are similar (e.g., LEDBAT and

[1]In recent kernels, it is not even on `net.ipv4.tcp_allowed_congestion_control`, i.e., among the TCP flavors allowed by default.

[2]The protocol is usually referred to either as LEDBAT (in the IETF community [24]) or as uTP (in the BitTorrent BEP [19] community). To avoid ambiguity, in this paper we employ its IETF name.

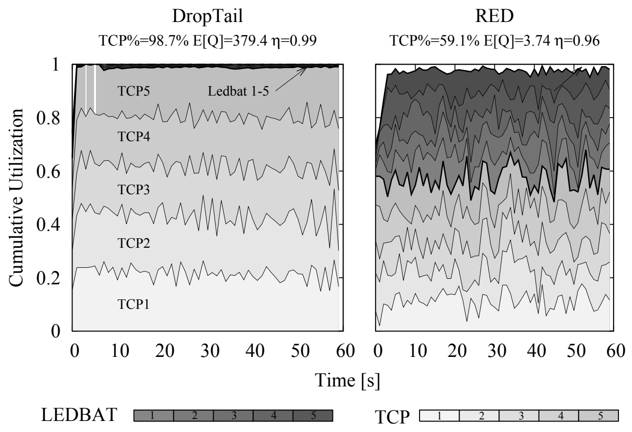

Fig. 1. Illustrational example of problems arising from the interaction of AQM and CC techniques (ns2 simulation for AQM=RED, CC=TCP+LEDBAT).

CoDel), their interplay can have negative consequences: we find *that AQM can induce a reprioritization of CC*. In other words, current scavenging protocols can successfully realize a lower-than-best-effort priority only if the bottleneck buffer operates according a DropTail discipline.

Fig. 1 illustrates the phenomenon: we stress that, while the picture depicts simulation results gathered in a very specific case, the remainder of this paper will show the phenomenon to hold under a large range of AQM+LPCC scenarios (using both real-world experiments and `ns2` simulations). The picture shows a breakdown of the link utilization when 5 TCP NewReno (each of which is represented with a different shade of light-gray) and 5 LEDBAT (dark-gray) backlogged flows share the same bottleneck. Capacity is set to 10 Mbps and the buffer has room for 500 packets (600 ms worth of delay); for the sake of simplicity, delays are homogeneous across flows. The left plot reports the case of a DropTail queuing discipline, while the right one reports the case of RED. Plots are annotated with further statistics concerning the average queue size in packets $E[Q]$, the capacity share exploited by the aggregate TCP%, and the average link utilization η.

In the DropTail case, the LEDBAT protocol operates in a lower-than-best-effort mode: we see that this delay-based scavenging protocol successfully exploits the spare capacity left unused by NewReno (as shown in [22]). The TCP aggregate uses the bulk of the capacity (TCP%=99%), with a fair share among TCP flows (due to homogeneous delay). However, the queuing delay approaches half a second because nearly 400 full-size TCP packets are queued on average. Clearly, bufferbloat would be even worse for lower (e.g., ADSL-like) capacities, or larger buffers (common defaults for home gateways are well in excess of 1000 packets).

In the RED case, while it is successful in limiting the queue size (less than 4 packets on average), this comes at the cost of (i) a slight 3% reduction of the link utilization, (ii) a complete reset of the relative level of priority between flows. In the case depicted in the figure, the share is now fair among all LEDBAT and NewReno flows, so that LEDBAT operates in a best effort mode, and is thus as aggressive as TCP. *While an AQM fixed the bufferbloat, it destroyed the relative priority among CC protocols.*

While the interaction between LEDBAT and AQM is pointed out in [23] and mentioned by the draft [24], we believe both the depth and the extent of the problem to be underestimated. As for the depth, Sec. II not only confirms the phenomenon to hold in the real world via Internet experiments on multiple AQM and LPCC techniques, but also shows that the *relative level of priority seldom reverse under AQM.* As for the extent, in reason of the limited availability of actual implementation of AQM and LPCC in the Linux kernel, we are forced to complement our experimental methodology with a simulation based one: Sec. III summarizes results of over 3,000 simulations, showing that the phenomenon just illustrated is a fairly general problem, arising from the interaction of AQM and any LPCC protocol. In spirit with the open TMA workshop series, we make our scripts and datasets available to the community at [1].

II. Experimental results

The goal of our experimental campaign is to confirm the occurrence of the phenomenon illustrated in Fig. 1 under a wide range of scenarios. Since our aim is not to propose any new AQM or LPCC, we select the only ones that are already available in recent Linux 3.2 kernel: namely, RED [11] and SFQ [17]) for AQM and TCP-LP [15] for the LPCC protocol family. As previously pointed out, LEDBAT cannot be excluded from the mix since it is, by far, the most successful LPCC: as such, we employ the `libUTP` [13] application-level implementation of BitTorrent that we already analyzed in [21].

A couple of points are worth stressing. Although we are aware of the fallacies of RED (as are the authors of CoDel [18]), we believe it needs to be considered as a reference benchmark (to which indeed CoDel is compared to in [18]). Additionally, note that RED is one of the few AQM policies available on common recent Linux kernels shipped with standard distributions, hence we believe it would not make sense to exclude it from this study due to its known performance issues [16], [10]. Similarly, we are aware of the latecomer unfairness issues of LEDBAT [22], [8], but its widespread use makes it necessary to consider it in the mix.

Incidentally, while our choice is forced by the availability of AQM and LPCC implementation in Linux, we can expect the performance of other AQM+LPCC combinations to fall into the boundaries defined by $\{$RED,SFQ$\}\times\{$TCP-LP,LEDBAT$\}$. On the one hand, this is due to the fact that RED vs. SFQ yield to small vs. larges queuing delays respectively; on the other hand, it is known that TCP-LP and LEDBAT have the most and the least aggressive behavior in the LPCC family [7] – simulation in Sec. III will confirm this expectation.

For any AQM and LPCC combination, we explore several experimental settings, including varying bottleneck capacity C, configuration of AQM parameters, number of flows in the bottleneck N, in both an emulated testbed and in a

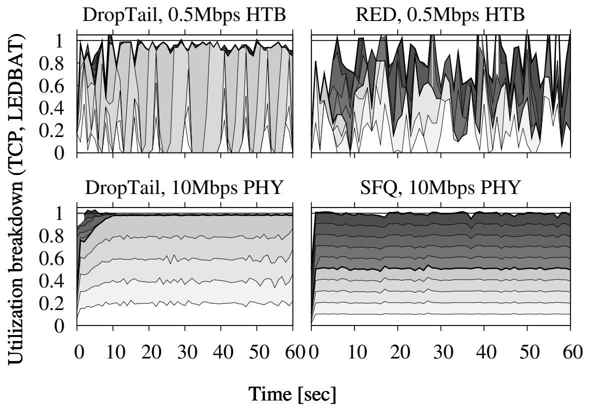

Fig. 2. Testbed experiments. Utilization breakdown among TCP and LEDBAT, for different AQM techniques (DropTail, RED, SFQ), capacities (500 Kbps, 10 Mbps) and emulation technique (HTB, PHY).

real Internet deployment. Throughout this paper, we express system performance mainly in terms of the link utilization η, the share of the link exploited by the TCP aggregate $TCP\%$, and the average queue length $E[Q]$. For convenience, we can either express $E[Q]$ in number of packets (possibly normalized over the buffer size $E[Q]/Q_{max}$ to gauge the bufferbloat intensity), or as the actual packet sojourn time in the queue (a more direct measure of the user quality of experience).

A. *Testbed experiments*

In the testbed experiments, we directly connect two PCs through a crossover Ethernet cable. Capacity limitations are either emulated through a Hierarchical Token Bucket (HTB) of the standard Linux traffic control `tc` suite (as in Fig. 1), or natively by forcing $C = 10$ Mbps PHY Ethernet through `ethtool` (which is a more reliable option than HTB). In both cases, we turn off most features that could possibly interfere with the experiments (e.g., jumbo frames, TCP segmentation offload, interrupt coalescing). To emulate a WAN setup, we inflate RTT delay by a constant amount equal to 30 ms using `netem`. We configure the `tc` queuing discipline to either DropTail (i.e., the default `pfifo_fast`), RED (with state of the art configuration) or SFQ.

We generate backlogged traffic during 60 seconds experiments, using `iperf` for all TCP flavors (both Best-Effort and low-priority) and simple client/server applications provided by `libUTP` for the application-layer LEDBAT implementation. For best effort TCP, we report results using NewReno, the protocol of choice at the IETF, and leave the study of Compound (default TCP flavor in Windows) and Cubic (default in Linux) for future work[3].

As for queuing delay measurement, we point out that dark buffers may lay at multiple points in the kernel stack (e.g., TCP buffers, device driver), and that buffering may occur outside the host (e.g., in the ADSL modem in Internet experiments). Hence, we opt for a simple methodology that mimics the way in which NICE measures the queuing delay: specifically, we monitor the RTT through a low frequency ICMP ping command and estimate queuing delay samples as $Q_i = \text{RTT}_i - \min_{j\leq i} \text{RTT}_j$. Notice that, as the reverse direction is carrying only ACK data and is thus not congested, we are safe in assuming that the RTT variation is due to queuing at the sender side.

[3]While Windows, and thus Compound, constitutes the largest portion of hosts, it would be difficult to replicate the same methodology. Preliminary tests with Cubic suggest the phenomenon to hold; at the same time, we incur in problems with Cubic vs. TCP-LP since the latter appears to be *more aggressive* than Cubic under DropTail– so we prefer to examine the issue at a later time.

TABLE I
TESTBED: LEDBAT VS. TCP-LP

		5+5 flows		1+1 flows	
		TCP%	E[Q] ms	TCP%	E[Q] ms
LEDBAT	FIFO	94.9	6437.3	99.5	5304.3
	RED	49.3	11.3	1.7	9.5
	SFQ	76.1	106.4	57.7	15.1
TCP-LP	FIFO	50.8	7870.6	65.8	7471.9
	RED	57.0	21.0	97.5	2.7
	SFQ	49.8	144.2	50.0	25.2

A further example of the temporal evolution of the utilization breakdown of TCP vs. LEDBAT flows is depicted in Fig. 2 for different AQM techniques (DropTail, RED, SFQ), capacities (500 Kbps, 10 Mbps) and emulation technique (HTB, PHY). The phenomenon early shown in Fig. 1 is thus confirmed for different AQMs and testbed settings: shortly, AQMs induce reprioritization of heterogeneous CC flows. We also see that, while when the capacity is abundant the breakdown is smooth, the opposite happens when capacity is scarce (which additionally leads to unfairness at short timescales).

We conduct a more systematic experimental testbed campaign that is summarized in Tab. I, reporting the TCP breakdown and average queuing delay for an emulated HTB capacity of $C = 500$ Kbps. The total number of flows varies in $N \in \{2, 10\}$, with flows equally split in the best effort TCP and LPCC families. Experiments report the average over 3 independent runs. As we may now expect, DropTail leads to bufferbloat of multiple seconds, which is instead solved by both RED and SFQ. The picture may change completely under RED, depending on the LPCC flavor and the number of flows: indeed, while for $N = 10$ the reprioritization happens for both LEDBAT (TCP%=49.3) and TCP-LP (57%), in the case that $N = 2$ flows compete for the same access, the LEDBAT+NewReno combination, through RED, *results in TCP NewReno starvation* (1.7%) while the opposite happens under TCP-LP+NewReno (97.5%).

These differences reflect the LPCC dynamics of LEDBAT and TCP-LP. The latter is still loss-based and AIMD-controlled, and its low priority stems from a slower recovery *after* losses that the AIMD dynamics force. The former is delay-based and PID-controlled: by limiting the queue size, it will seldom be penalized under RED. Whenever the queue grows due to best-effort TCP AIMD, LEDBAT reduces its own window, and so the chances that a packet will get dropped are

TABLE II
LEDBAT: TESTBED VS. INTERNET EXPERIMENTS

		5+5 flows		1+1 flows	
		TCP%	E[Q] ms	TCP%	E[Q] ms
Testbed	FIFO	94.9	6437.8	99.5	5304.3
	RED	49.3	11.3	1.7	9.5
	RED⋆	71.9	763.9	98.2	333.8
Internet	FIFO	97.0	6551.7	98.9	916.0
	RED	2.6	45.0	5.1	29.1
	RED⋆	63.8	745.2	86.7	305.4

very low. Whenever TCP experienced a timeout and abruptly shrank its window, LEDBAT instead grows its window again, but in reason of its low target delay, it will limit the amount of queuing and again prevent its packets from being dropped.

Under SFQ, TCP starvation phenomena are avoided. All flows get hashed in different buckets at enqueue time, and since hash buckets are queried in a round robin fashion, this guarantees that each flow is able to send data in turn – including the best-effort TCP that was heavily penalized under RED. Yet, reprioritization still occurs. However, the queuing delay under SFQ (which has a default buffer size of 127 packets) grows up to about 100-150 ms, thus approaching the limit of what is considered to be harmful for interactive communication.

B. *Internet experiments*

We then perform additional experiments on the wild Internet. Since we have already shown the reprioritization to hold for different combinations of LPCC and AQM, our aim here is to disproof that these are artifacts that only arise in testbeds due to, e.g., the extremely controlled settings or, conversely, due to unexpected interactions due to the emulation layer.

In order to replicate a setup as close as possible to the typical user scenario, the sender is connected in 802.11g WiFi to an ADSL box. The receiver is connected to another ADSL box of another ISP through the Ethernet interface. The minimum RTT delay between the two hosts is approximately 50 ms, and the capacity between the hosts only slightly exceeds 500 Kbps (so that it can be compared with the testbed).

We carry on experiments only for the LEDBAT LPCC, using two configurations[4] of RED with results summarized in Tab. II. It can be seen that TCP starvation persists under RED when the number of flows is small: we stress that we observed TCP starvation only under the RED+LEDBAT combination, since as previously explained RED tries to penalize flows proportionally to the queue they create, while LEDBAT is designed to precisely avoid queuing. Hence, it seems that we have found yet another reason not to deploy RED other than those listed in [16]: since RED is one of the few AQMs available in the Linux kernel, and due to the growing amount of LEDBAT traffic, we believe this potential problem is worth highlighting once more.

[4]RED⋆ is a configuration inspired by [10] and crafted ad hoc by a standard trial and error procedure. Configuration details and scripts used for these experiments are available at [1].

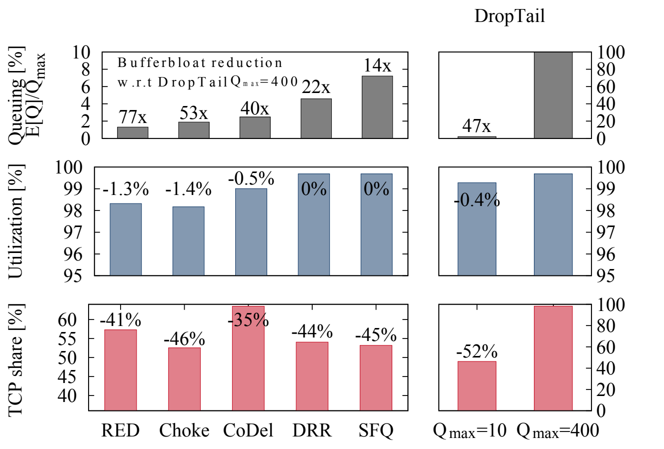

Fig. 3. Impact of AQM policies on the bufferbloat intensity $E[Q]/Q_{max}$ (top) link utilization η (middle) and TCP% breakdown (bottom).

Finally, the ad hoc RED⋆ configuration reinstates instead the relative CC priorities, but at the cost of an already sizeable bufferbloat. Intuitively, in this case our configuration starts dropping packets after the LEDBAT queuing delay target, so that TCP has the chance to grow its congestion window at the expense of LEDBAT (that by its LPCC nature will backoff in presence of best effort TCP) before one of its packets get dropped by AQM; however, TCP recovery is in this case fast enough, and the additional buffer space beyond the LEDBAT target is large enough, to let TCP prevail over LEDBAT.

III. SIMULATION RESULTS

Finally, the aim of our `ns2` simulation campaign is to test the validity of the reprioritization phenomenon under the largest possible set of scenarios. We therefore include additional AQM techniques that, though popular in the research community, are however not available in the Linux kernel – namely, Choke [20], DRR [25], CoDel [18]. Similarly, we extend our investigation of LPCC techniques to include NICE [26]. Notice that some of these modules are not directly available in `ns2` (version 2.33), that we patched to support Choke [20], CoDel [18], LEDBAT and NICE (the last two LPCCs using our own open source implementations [2]).

In what follows, we aim at conveying the most relevant message we gather from simulation in a straightforward way: thus we first consider a subset of the results to better stress (i) the impact of AQM policies in Sec. III-A, (ii) the joint impact of AQM and LPCC in Sec. III-B, and (iii) the validity of the observation over a large parameter range in Sec. III-C.

A. *Impact of AQM policy*

Unless otherwise stated, we consider an equal number $N = 5$ of best effort TCP NewReno and low-priority flows sharing a link having capacity $C = 4$ Mbps and a buffer size $Q_{max} = 400$packets (corresponding to a maximum bufferbloat of 1.2 seconds). All flows are backlogged, start at time $t = 0$, and have a homogeneous-delay $RTT = 50$ ms. Simulations last for 60s, and 10 runs are repeated for each parameter settings.

For the time being, we fix the LPCC to LEDBAT, and examine the impact of different AQM techniques. Fig. 3 reports the bufferbloat intensity $E[Q]/Q_{max}$ (top) link utilization η (middle) and TCP% breakdown (bottom) for varying AQM policies, and for DropTail as a comparison (right). As expected, at the price of a slight decrease in the link utilization, AQM reduces the intensity of the bufferbloat: compared to a persistently full DropTail buffer, queue size and delay is from 14 to 77 times smaller, using SFQ and RED respectively. This implies that, under AQM the queuing delay is always less than 85 ms (SFQ, worst case), and generally much lower. However, we see that the capacity share of the TCP aggregate can drastically reduce 35%-46% (under CoDel and Choke respectively). Notice that this result alone extends the validity of the phenomenon observed under unknown[5] AQM policies in [23]. Finally, we point out that simply reducing the DropTail buffer size is not a solution to the problem either: notice that (i) as shown in Fig. 3, shorter DropTail buffer behaves like AQM, in that bufferbloat is reduced but at the expense of reprioritization and (ii) in case of varying link bandwidth as in case of WiFi, there is no fixed size that would not translate into bufferbloat.

B. *Impact of AQM policy and LPCC protocol*

We now explore the full product of LPCC flavors and AQM techniques. Under DropTail, it is known [7] that LEDBAT achieves the lowest priority against best effort TCP, followed by NICE and TCP-LP. Both LEDBAT and NICE are delay-based (the difference being that LEDBAT relies on one-way delay measurement, NICE on RTT ones), while TCP-LP is AIMD-based. We illustrate results as a *parallel coordinate* plot in Fig. 4. Having noticed that link utilization is subject to small variations, we consider the two main metrics of interest: namely the bufferbloat intensity $E[Q]/Q_{max}$ (left y-axis) and TCP% share (right y-axis). Each (LPCC, AQM) pair is represented as a line in Fig. 4, and a light-gray strip represents the ideal case where the queue is short but priorities are unaltered. For instance, we see that the three (LPCC,DropTail) combinations appear as horizontal lines on the top of the figure, since under DropTail bufferbloat has maximal intensity and TCP% monopolizes the bottleneck (the order of the curves reflect the order of priority in [7]). Conversely, all (LPCC,AQM) combinations excluding DropTail are very close, as AQM implies low bufferbloat but jeopardizes CC priorities (specifically, all TCP and LPCC flows have roughly the same priority).

C. *Sensitivity analysis*

We conducted over 3,000 simulations to investigate the effect of other parameters such as (i) buffer size $Q_{max} \in [100, 400]$, (ii) link capacity $C \in [0.25, 10]$ Mbps, (iii) homogeneous RTT=50 ms vs. heterogeneous RTT delay with $E[RTT] = 50$ ms, (iv) number of LPCC and TCP flows $N \in [1, 5]$, (v) flow duty cycle in [10%,100%] – flows have

[5]Authors just mention that "different prioritized traffic classes" are implemented in the "modern home gateways" used in their experiments.

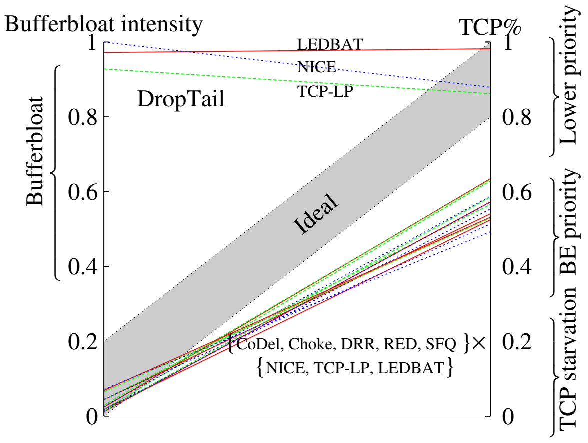

Fig. 4. Joint impact of AQM and LPCC on the queuing delay and TCP breakdown.

an on/off behavior, with exponentially distributed on/off times, and have infinite data to send during on periods. An extended set of results (that we avoid to report here for lack of space but that we make available along with the simulation scripts at [1]), confirms that the reprioritization phenomenon remains valid under all explored circumstances.

Here, we only briefly comment on the impact of (iii) RTT heterogeneity, in which case it is interesting to observe the *intra-protocol* fairness (i.e., among flows in the same TCP or LPCC aggregate), that may change significantly depending on the AQM policy. We measure the average Jain fairness index as $F = (F_{TCP} + F_{LPCC})/2$, where $F_X = (\sum_{i=1}^{N} T_i)^2/(N \sum_{i=1}^{N} T_i^2)$ is the fairness within aggregate X, with T_i throughput of the i-th connection. We have that fairness under CoDel ($F = 0.81$) is only slightly better than RED ($F = 0.78$) and significantly smaller than SFQ ($F = 0.99$). Notice that CoDel fairness range is coherent with [18], which however does not address a comparison with SFQ and reports significantly smaller fairness values for RED (under a more heterogeneous scenario).

IV. Related work

It would be extremely cumbersome to retrace over 20 years of Internet research in these few pages (we refer the reader to [12], where without providing a complete picture, it does however present a historical viewpoint). We extend this viewpoint by reporting in Fig. 5 a timeline of the AQM and LPCC algorithms used in this paper. The timeline clearly shows a temporal separation of the two research topics, which in our opinion helps understand why the AQM vs. LPCC interaction assessed in this paper was not previously exposed.

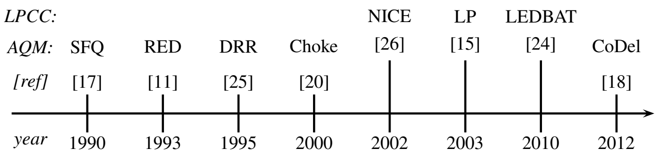

Fig. 5. Timeline of AQM and LPCC algorithms.

To the best of our knowledge, only [23] mentions AQM and a LPCC (namely, LEDBAT) in the same paper. In one of the tests authors experiment with a home gateway that implement some (non-specified) AQM policy other than DropTail. When LEDBAT and TCP are both marked in the same "background class" the "TCP upstream traffic achieves a higher throughput than the LEDBAT flows but significantly lower than" under DropTail [23]. Moreover, under AQM it is possible that "LEDBAT reverts to standard TCP behavior, rather than yield to other TCP flows" [24]. In this paper, we not only show that this behavior is general and can arise from the interaction of any AQM and LPCC, but we also show that the AQM/LPCC interplay can lead to best effort TCP starvation.

V. Conclusion

This work points out possible negative issues arising from the interaction of AQM and CC techniques. Specifically, under AQM it is likely that LPCC techniques will become as aggressive as best effort TCP, and can mislead at least one flavor of TCP to starvation (under RED). We now discuss the implication of these findings.

As it seems that AQM and LPCC will have to coexist, there is a need to find possible ways out of the negative interplay we have shown in this paper. A general solution is hard to find, as testified by the current standpoint after over 20 years of research. Yet, a patch to the problem may be within reach, but may be hidden by radical positions in favor of either AQM or LPCC. While some see low priority protocols as useful in the transient period until AQM will be deployed [23], others are not convinced by AQM and propose "the end to end approach as the solution to bufferbloat and just forget about changing router behavior." [6]. Arguing that a practical solution requires a compromise between both extremes, we agree with more moderate viewpoints that "AQM is just one piece to the solution of bufferbloat" [18].

Consider indeed that an ideal solution (recall Fig. 4) should guarantee low access delay irrespectively of the mix of CC protocols and maintain the relative level of priority among flows in the mix as well. Since even a single AIMD flow may bufferbloat the others, the solution *needs AQM*. At the same time, to avoid the CC reprioritization phenomenon, we argue that classification capabilities will be needed in AQM to account for flows' *explicitly advertised* level of priority. Notice that while in the more general case classification has failed to be adopted (IP TOS field, DiffServ, etc.), and the ability to claim *higher* priority could be easily gamed, in a hybrid AQM vs. LPCC world it makes sense for flows to claim a *lower* priority.

Other avenues in AQM and LPCC remain to be explored. First, AQMs evaluated in this paper either implement drop (RED/CoDel/Choke) or scheduling (SFQ) strategies. At the same time, hybrid AQM techniques that *jointly* exploit fair queuing with early drop are appearing – as the `fq_codel` technique that will make its way in future Linux kernels, starting from 3.5 [3]. Though further testing will be needed on these new AQM, we argue that the reprioritization problem will remain.

Another sensible question is whether it would be possible to differentiate priorities at a finer grain within the LPCC class. For instance, the LEDBAT draft claims that different levels of priorities can be induced by using heterogeneous targets: while this solution may lead to starvation in case of backlogged flows under a DropTail discipline [7], is not clear what the behavior would be under AQM.

Acknowledgement

This work was carried out at LINCS http://www.lincs.fr. The research leading to these results has received funding from the European Union under the FP7 Grant Agreement n. 318627 (Integrated Project "mPlane").

References

[1] http://www.infres.enst.fr/~drossi/dataset/ledbat+aqm.
[2] http://www.infres.enst.fr/~drossi/ledbat.
[3] https://lists.bufferbloat.net/pipermail/codel/.
[4] Aduf - historique firmware, 01/07/2005. http://88.191.250.12/viewtopic.php?t=164746&view=previous.
[5] B.Cohen. How has bittorrent as a protocol evolved over time. http://www.quora.com/BitTorrent-protocol-company.
[6] B.Cohen. Tcp sucks. http://bramcohen.com/2012/05/07/tcp-sucks.
[7] G. Carofiglio, L. Muscariello, D. Rossi, and C. Testa. A hands-on Assessment of Transport Protocols with Lower than Best Effort Priority. In *IEEE LCN*, 2010.
[8] G. Carofiglio, L. Muscariello, D. Rossi, and S. Valenti. The quest for LEDBAT fairness. In *Globecom*, 2010.
[9] S. Cheshire. It's the latency, stupid! http://rescomp.stanford.edu/~cheshire/rants/Latency.html, 1996.
[10] M. Christiansen, K. Jeffay, D. Ott, and F.D. Smith. Tuning red for web traffic. In *ACM SIGCOMM CCR*, volume 30, pages 139–150, 2000.
[11] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.
[12] J. Gettys and K. Nichols. Bufferbloat: dark buffers in the internet. *Commun. ACM*, 55(1):57–65, 2012.
[13] G. Hazel. uTorrent Transport Protocol library. http://github.com/bittorrent/libutp, May 2010.
[14] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson. Netalyzr: Illuminating the edge network. In *ACM IMC*, pages 246–259, 2010.
[15] A. Kuzmanovic and E.W. Knightly. TCP-LP: A distributed algorithm for low priority data transfer. In *IEEE INFOCOM*, 2003.
[16] M. May, J. Bolot, C. Diot, and B. Lyles. Reasons not to deploy RED. In *IEEE IWQoS,*, pages 260–262. IEEE, 1999.
[17] P.E. McKenney. Stochastic fairness queueing. In *IEEE INFOCOM*, 1990.
[18] Kathleen Nichols and Van Jacobson. Controlling queue delay. *Commun. ACM*, 55(7):42–50, July 2012.
[19] A. Norberg. BitTorrent Enhancement Proposals on uTorrent transport protocol (BEP29), 2009.
[20] R. Pan, B. Prabhakar, and K. Psounis. Choke - a stateless active queue management scheme for approximating fair bandwidth allocation. In *IEEE INFOCOM*, 2000.
[21] D. Rossi, C. Testa, and S. Valenti. Yes, we LEDBAT: Playing with the new BitTorrent congestion control algorithm. In *PAM*, 2010.
[22] D. Rossi, C. Testa, S. Valenti, and L. Muscariello. LEDBAT: the new BitTorrent congestion control protocol. In *IEEE ICCCN*, 2010.
[23] J. Schneider, J. Wagner, R. Winter, and H.J.Kolbe. Out of my way – evaluating low extra delay background transport in an ADSL access network. In *ITC22*, 2010.
[24] S Shalunov. Low Extra Delay Background Transport (LEDBAT). IETF Draft, March 2010.
[25] M. Shreedhar and George Varghese. Efficient fair queueing using deficit round robin. In *ACM SIGCOMM*, 1995.
[26] A. Venkataramani, R. Kokku, and M. Dahlin. TCP Nice: A mechanism for background transfers. In *USENIX OSDI*, 2002.