To the Moon and back: are Internet bufferbloat delays really that large?

Chiara Chirichella and Dario Rossi
Telecom ParisTech, Paris, France
first.last@enst.fr

Abstract—**Recently, the "bufferbloat" term has been coined to describe very large queuing delays (up to several seconds) experienced by Internet users. This problem has pushed protocol designer to deploy alternative (delay-based) models to the standard (loss-based) TCP best effort congestion control. In this work, we exploit timestamp information carried in the LEDBAT header, a protocol proposed by BitTorrent as replacement for TCP data transfer, to infer the queuing delay suffered by remote hosts. We conduct a thorough measurement campaign, that let us conclude that (i) LEDBAT delay-based congestion control is effective in keeping the queuing delay low for the bulk of the peers, (ii) yet about 1% of peers often experience queuing delay in excess of 1 s, and (iii) not only the network access type, but also the BitTorrent client and the operating system concurr in determining the bufferbloat magnitudo.**

I. Introduction

Sending packets over the Internet sometimes takes more than Earth-to-Moon communication: one-way propagation delay in the latter case takes slightly more than one second, while TCP/IP packets sometimes gets queued for several seconds [7], [12].

Reasons under this phenomenon, for which the "bufferbloat" term was reacently coined [7], are due to excessive buffer sizes coupled to TCP congestion control mechansim: as TCP halves its sending window as a reaction to *losses*, the buffer forcibly has to fill. Notice that buffer sizes involved are often small in absolute terms (few KBs), but are large relatively to capacity (few Kbps) of the narrow cable, ADSL or WiFi links in front of these buffers, translating into possibly very large queuing delays. Notice also that the problem is hardly solved by reducing the buffer size: this is clearly understood considering WiFi links, where the range of capacities for the same AP-client pair depends on the signal to noise ratio and may vary by orders of magnitude at any moment.

While the problem has been known for quite some years now [8], Moore's law has driven down manufactures cost, making large buffer sizes cheaper. To make the point, [10] assesses the relative popularity of different set-top-boxes. The top-10 models account have an average buffer size of 136KB. The most popular model, that represents nearly 40% of all samples, has the smallest buffer among all models (48KB), that would however translate into 1 sec worth of queuing delay for the common uplink capacity of 384Kbps.

Clearly, bufferbloat could be worst for other models, as indeed authors of Netalyzr [12] show. At the same time, we point out that Netalyzr methodology exposes by design the *maximum achievable bufferbloat*, as it saturates a link with a backlogged transfer lasting for several seconds. However, currently little is known about the *typical bufferbloat* that users experience during their daily Internet activities: as such, while the networking community is aware of the problem existence and of the magnitudo of its worst case, it mostly ignores its typical instance.

In this work, we measure the typical bufferbloat by exploiting LEDBAT, a novel delay-based protocol used by BitTorrent as a replacement of TCP for data swarming, that has lately been standardized at the IETF. We design our methodology such that we are able to passively monitor the upstream queuing delay of any remote host that is sending BitTorrent data chunks to one of our controlled (but otherwise unmodified) BitTorrent peers. We note that is perfectly in line with the Dasu approach [6], that advocates to move the ISP characterization toward the network edge. We stress that Dasu as well leverages BitTorrent for Internet measurement: as BitTorrent is a well established and popular application, it has the ability to reach a large user-base.

Our main contributions are as follows. First, we implement the bufferbloat methodology in Tstat [1], a popular flow-level logger – as we are currently in the process of merging our patches in the main Tstat trunk, we make the software available upon request in the meanwhile. Second, we carry on a fairly large measurement campaign, involving probes connecting to 12 torrents, from 3 vantage points for a total of 88 experiments in which we contacted over 25K external peers – in spirit of the TMA workshop series, we make these traces available at [2]. Third, we analyze the dataset in order to make a valuable assessment of the typical bufferbloat experienced by BitTorrent users. Our methodology is able not only (i) to *accurately describe the bufferbloat* delay typically encountered by BitTorrent users, e.g., in terms of per-peer average and percentiles, but also (ii) to give partial *explanation of its root causes*, e.g., relatively weighting the impact of the BitTorrent client, operating system, or access type of different peers. We find that (i) while the bulk of the users have a moderate bufferbloat, a small percentage of users is significantly impacted and that (ii) not only the type of access network, but also the BitTorrent client and the operating system have a sizeable impact in determining user experience.

II. Methodology

A. Bufferbloat inference

To gauge remote bufferbloat delay, we collect one-way delay (OWD) samples d_i, establishing the minimum as a baseline delay $min_{j\leq i}d_j$, and then equating the queuing delay q_i as the difference of the current sample with respect to the baseline $q_i = d_i - min_{j\leq i}d_j$. Intuitively, a packet finding an empty queue at the bottleneck will incurr the propagation delay component only ($min_{j\leq i}d_j$).

We point that this is a classic approach used in congestion control to drive congestion window dynamics: early work on this subject date back in late 1980s and more recently LEDBAT [14] followed suit. Our innovation is to demonstrate how a passive observer of LEDBAT traffic can exploit this approach to estimate the uplink delays of remote hosts.

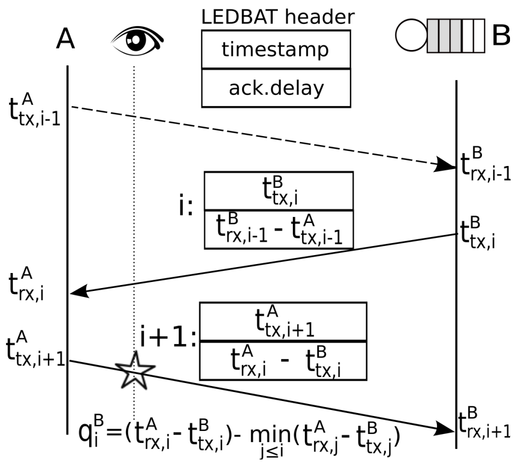

Fig. 1. Remote buffer measurement methodology

Fig. 1 illustrates our methodology. Based upon information directly available in LEDBAT packet headers, once it has seen 3 packets (the moment is marked by a star), a passive observer O is able to estimate the queuing delay experienced by packet i of host B. These delays may result from all upstream traffic from B, including flows other than the one being observed and most importantly, possibly directed to other hosts than A, and of which thus the observer has no knowledge.

In the absence of framing specification in the LEDBAT standard, our protocol parser is based on BitTorrent's currently implemented BEP-29 definition [3]. This specifies a header field, named "timestamp" in Fig. 1, that the sender fills with its local clock. On reception of a new packet, the receiver calculates the OWD as the difference between its own local clock and the sender timestamp, and sends this value back to the sender in another LEDBAT header field, named "ack.delay" in Fig. 1. In this way, each acknowledgement packet conveys the estimated OWD incurred by the most recently received data packet: upon reception of an acknowledgement, the sender infers the queuing delay from simple operations on the packet headers, and uses this information to grow/shrink the congestion window [14]. At high level, LEDBAT strives to add a small fixed amount of additional "target" queuing delay: once the inferred queuing delay reaches the target, the congestion window settles and no longer increase (unlike in the TCP case).

As the observer close to A sniffs the packets, it performs the same state updates as does the LEDBAT congestion control protocol running on A. Notice that there is no need for the passive observer to estimate the clock of the probe host A: all the needed information are carried in the LEDBAT header. At each packet reception, the observer updates the base delay β_{BA} as the minimum over all OWD $B \to A$ samples:

$$\beta_{BA} = \min(\beta_{BA}, t^A_{\mathrm{rx},i} - t^B_{\mathrm{tx},i}), \qquad (1)$$

$$q^B_i = (t^A_{\mathrm{rx},i} - t^B_{\mathrm{tx},i}) - \beta_{BA} \qquad (2)$$

Then, the queuing delay q^B_i incurred by packet i can be inferred by subtracting β_{BA} from the timestamp difference carried in packet $i+1$. Whereas the base delay is dependent upon the unknown offset between the clocks at B and A, the queuing delay is independent of the offset, as the offset cancels in the difference operation.

B. Validation and coupling bias

In a controlled testbed, we have carefully validated the methodology, that yields to extremely reliable results against the ground-truth provided by kernel-level queue length measurement. While validating the above methodology is imperative, in this work we focus on applying the methodology at large, so to assess the current state of the Internet bufferbloat – a more interesting and challenging subject. We refer the reader to [9] or to a companion technical report available at [2] for more details of the validation.

Yet, before proceding in our analysis, an important point is worth stressing. As our bufferbloat measurement methodology opportunistically exploits data transmission over LEDBAT, this implies that our bufferbloat sampling process is governed by the very same LEDBAT window dynamics.

According to the LEDBAT standard, once the sender measures the queuing delay q_i with (2), it then reacts via the following congestion window (cwnd) dynamics:

$$\mathrm{cwnd}_{i+1} = \max(1, \mathrm{cwnd}_i + \gamma(\tau - q_i)/(\tau\mathrm{cwnd}_i) \qquad (3)$$

where γ and τ (respectively "GAIN" and "TARGET" in the standard terminology) are set to 1 and 100 ms by default [14].

From (3), it is evident that the number of q_i samples gathered in a RTT will be affected by the value of the queuing delay q_i itself. More precisely, when the queuing delay exceeds the target ($q_i > \tau$) due, e.g., to other traffic crossing the same bottleneck, LEDBAT reduces the window (when $\tau - q_i < 0$ then $\mathrm{cwnd}_{i+1} < \mathrm{cwnd}_i$): if the cross traffic persists and the delay grows, LEDBAT reduces its sending rate down to a minimum[1] of 1 packet per RTT. This means that, whenever the queuing delay is large, we expect to receive few packets: hence, in case every sample is equally weighted, large queueing delays are *undersampled*.

Conversely, when no other traffic than LEDBAT is active on the bottleneck link, by definition LEDBAT strives to yield a target τ queuing delay and settles the congestion window growth when the target is reached. We thus expect to receive many samples[2] carrying values close to τ: hence, in case every sample is equally weighted, τ-long queueing delays are *oversampled*.

Illustrative examples of this phenomenon are shown in Fig. 2 and Fig. 3. In more details, Fig. 2 examplifies the temporal evolution of (i) the number of samples in a $\omega = 1$ s window (left y-axis, with empty point ○) vs (ii) the value of queuing delay q_i (logarithmic scale, right y-axis, with filled point ●). Top-right plot shows the example of a controlled testbed experiment used in our validation, where we artificially inject cross TCP traffic at periodic times, by transferring files of exponentially increasing size at each subsequent period. In this particular case, it is possible to observe that congestion window stays often at its maximum value (in absence of TCP traffic) or drops to the minimum value (in presence of TCP traffic). The other plots report the case of peers in our Internet experiments. Top-left peer achieves very steady queuing delay, unlike peers in the bottom plots (right: ADSL; left: WiFi), whose queuing delay however fluctuates around $\tau =$100 ms as expected.

[1]Minimum sending rate ensures that the LEDBAT sender continues to periodically measure the queuing delay, and opportunistically grow the window again when the other traffic slowsdown or stops.

[2]The exact number of samples depends on the bandwidth-delay product: e.g., for instance, to fill a 1Mbps link about cwnd=4 packets are needed when RTT=50ms and cwnd=21 when RTT=250ms.

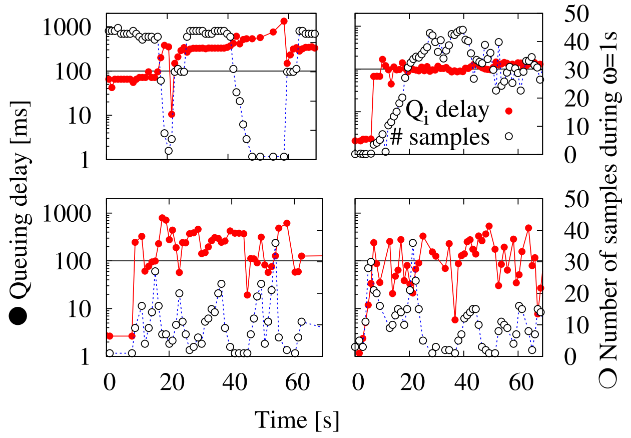

Fig. 2. Examples of temporal evolution of (i) the number of samples in $\omega = 1$ s windows vs (ii) the value of queuing delay q_i samples

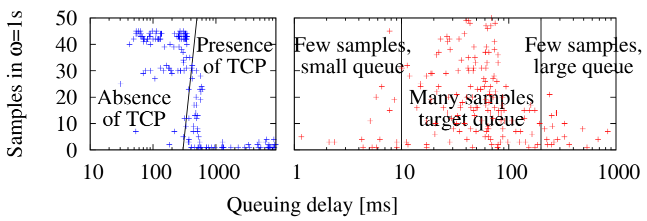

Fig. 3. Scatter plot of the per-packet queuing delay versus the number of samples in $\omega = 1$ s windows. Controlled testbed peer (left, corresponding to Fig. 2 top-left) vs Internet peers (right, corresponding to Fig. 2 top-right).

Fig. 3 shows instead a scatter plot representation of peers in the top plots of Fig. 2. For each $\omega = 1$ s window, where a number s of samples are carried by individual LEDBAT packets, all q_i samples with $i \in [0, s] \subset \mathbb{N}$ gathered in that window are shown in the plot as (q_i, s) pairs. Left plot of Fig. 3 refers to the controlled testbed peer of top-left plot of Fig. 2: in this case, LEDBAT is continuously transmitting, with periodic TCP cross-traffic interference. Two regions can be identified: either (i) a full window worth of packets carries queuing delay close to the LEDBAT target (absence of TCP), or (ii) due to excessive queuing the number of samples falls down to about one per RTT (presence of TCP). Right plot of Fig. 3 instead refers to the Internet peer of top-right plot of Fig. 2. Three regions can be discerned in this case: (i) a low-delay sample-sparse zone, when peers send few traffic in non-backlogged mode (e.g., at the beginning of a flow, see Fig. 2 toward $t = 0$), then (ii) a moderate-delay samples-dense zone (around queuing delay target τ, where oversampling possibly occurr) and (iii) a high-delay sample-sparse zone (where undersampling possibly occurr).

This potential measurement bias is intrinsic to the *coupling* of the measurement process with the LEDBAT cwnd dynamics: a very simple but effective countermeasure is to make the measurement process *asynchronous* with respect to the window dynamics. In other word, rather than counting every packet as a sample, we batch packets in staggered windows of short temporal duration ω. Periodically, we compute statistics over the packets arrived during the window: we use the average of the aggregate as bufferbloat sample of the new process.

In more detail, denote with t^O_{i+1} the time at which the (i+1)-th acknowledgement packet carring information to compute the queuing delay q^B_i incurred by the i-th data packet (i.e., t^O_{i+1} is the time marked with a star in Fig. 1, which is larger than $t^A_{tx,i+1}$, the transmission time of the (i+1)-th packet from A). Then, denote as Q^B_k the measurement sample aggregating information gathered from samples received during the k-th window $[k\omega, (k+1)\omega)$:

$$Q^B_k = E[\{q^B_i : t^O_{i+1} \in [k\omega, (k+1)\omega)\}_i] \qquad (4)$$

It is intuitive that in case the LEDBAT sender B is transmitting at full bottleneck rate, multiple $q^B_i = \tau$ samples[3] will be counted as a single $Q^B_k = \tau$ sample. Similarly, when the queue is full and the LEDBAT sender rate is minimal, the possibly unique[4] q^B_i sample will be counted once by Q^B_k, eliminating thus the bias.

We now compare the biased q_i and unbiased Q_k statistics, for two values of $\omega = \{1, 5\}s$. Notice that the choice of ω is guided by the timescale of user activities and is correlated with quality of experience estimation. For every window of duration ω, we want to assess what are the typical delay users can expect to wait. A $\omega = 1$s duration is thus very relevant for both interactive VoIP/gaming communication and Web traffic. Concerning VoIP/gaming, an average $Q_k \geq 150$ms is symptomatic of bad QoE during the k-th window [4]. Concerning Web traffic, notice that the serial actions performed by the browser (e.g., DNS lookup, TCP three-way handshake at layer-4, and subsequent HTTP GET at layer-7) all incurr the queuing delay, so that $Q_k \geq 1$s is symptomatic of bad QoE.

We show in Fig. 4 the distribution of the biased q_i and unbiased Q_k statistics for $\omega = \{1, 5\}s$. Notice that there may be samples Q_k, obtained during the k-th window, reporting values larger than the window ω itself. We point out these to be valid samples: indeed, since LEDBAT minimum sending rate is 1 packet per RTT, this happens whenever the RTT exceeds ω due to the queuing delay. For this reason, we also consider a larger window $\omega = 5s$, though in this case we loose a precise time granularity (an average Q_i over $\omega = 5$ s large windows is less telling as far as user QoE is concerned). As Fig. 4 shows, the Cumulative Distribution Function (CDF) of all packet-level queuing delay samples q_i is, as we early argued, biased to a systematic underestimation of high queuing delays. This is solved in the windowed approach, that furthermore exhibits very similar statistics for both $\omega = \{1, 5\}$ s. This is even more evident in the Complementary CDF (CCDF) depicted in the inset of Fig. 4, showing that while only about 1/1000 q_i samples exceeds 10 seconds of queuing delay, a windowed Q_i estimate corrects this amount to 1/100 when for

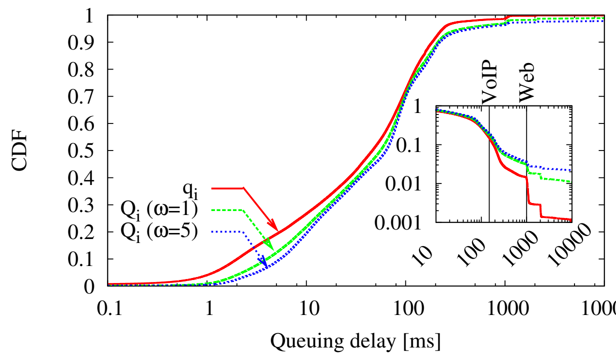

Fig. 4. Cumulative distribution function of per-packet biased (q) and windowed unbiased (Q) queuing delay statistics

[3]Whose number is cwnd $\cdot\, \omega/RTT$

[4]In this case cwnd=1 and due to queuing delay, RTT can grow larger than ω

TABLE I
TORRENTS DESCRIPTION

Size (GB)	0.06	0.08	0.2	0.1	1.4	1.2
Type	B	B	S	G	G	G
Seeds	30	86	40	50	57	65

Size (GB)	0.5	0.2	0.04	1.5	0.7	0.7
Type	D	D	P	V	V	V
Seeds	93	92	79	236	8132	9635

(B)ooks, (G)ames, (S)oftware, (D)ocumentary, (P)odcast, (V)ideo

$\omega = 1$ (and to 2/100 for $\omega = 5$). That is to say, for 1% of 1 s long windows, new browsing sessions may be terribly slow; placing new VoIP calls may be significantly delayed, and user may equate this with call being blocked; finally, ongoing VoIP calls and game sessions may suffer from seldom QoE disruption.

Since our evaluation of LEDBAT testify its low priority [13], queuing delay is likely due to interfering TCP traffic, that is not necessarily due to BitTorrent swarming. Still, we notice that presence of BitTorrent TCP traffic is possible due to (i) firewalls blocking UDP, as BitTorrent's LEDBAT incarnates as an application-layer protocol encapsulated in UDP at transport layer, or (ii) legacy clients that do not implement LEDBAT congestion control, or (iii) uTorrent clients having disabled LEDBAT/uTP by tweaking the `bt.transp_disposition` parameter.

Overall, this preliminary analysis let us conclude that (i) a non marginal fraction (1%) of 1 s windows samples incurr queuing delay in excess of Earth-to-Moon delay, and that (ii) batching packets in windows decouples delay statistics from the undergoing LEDBAT dynamic cwnd adjustment process. Since only minimal difference arise for $\omega = \{1, 5\}$ s in the following we limitedly consider 1s long windows as they are more relevant for user QoE.

III. DATASET

We apply the previously described methodology to an active measurement campaign, involving heterogenity of torrents, clients and vantage points. The experimental campaign was performed over a 8-month period during 2012, principally using uTorrent and Transmission in their default configurations, for a total of 88 experiments. Clients were run from 3 vantage points in Italy, France and Austria, from which we obtained about 23 GB of raw packet traces, that we make available at [2].

To avoid being biased on a specific content type, size and popularity (the latter correlated with the swarm size), we let our probes join several (legal) torrents gathered from Mininova. As reported in Tab. I, torrents span different categories (2 eBooks, 2 games, 1 software, 2 documentaries, 1 podcast and 3 videos), file sizes (ranging from 56 MB for an eBook to 1.5 GB for a video, and number of seeds (from 30 for the least popular eBook to 9,635 for the most popular video).

Specifically, a probe under our control joined several torrents as a leecher, end exchanged data with remote peers over either LEDBAT (when available) or TCP (legacy clients). Though our methodology also applies to TCP traffic [2], in the following we limitedly focus on queuing delay samples gathered by LEDBAT traffic. As LEDBAT has become BitTorrent default congestion control protocol, replacing thus TCP, this choice is not detrimental to the extent of our analysis: indeed, according to Brahm Cohen and to our own measurements more than half of the BitTorrent traffic is now carried over LEDBAT.

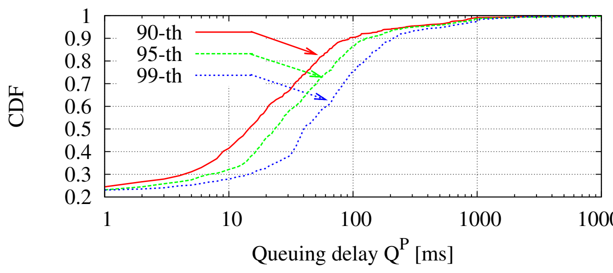

Fig. 5. Distribution of per-peer queuing delay percentiles

Overall, our BitTorrent probes exchanged with 24,678 external peers, about half of which transferred data using LEDBAT (over which we gathered, e.g., Fig. 4). Clearly, an host may participate into multiple swarms. Furthermore, as our experiments are repeated with different clients, we can thus encounter the same external peer multiple time over our experimental campaign. This is indeed the case, as we find 8914 unique IP addresses. Among these peers, in order to gather statistically meanignful results, we require a minimum amount of queuing delay samples q_i^B per peer: we only consider peers yielding at least 25 queuing delay samples (or 50 packets). Notice that, in case we encounter a peer multiple times in our experiments, we count it (at most) once per each experiments in which it appears (for each experiment, the peer is considered provided that it sends 50 packets). In the reminder of the analysis, we focus on this peer subset, consisting of 6,896 peers (1,931 unique IPs).

IV. ANALYSIS

A. Bufferbloat observation

First, we start by considering the distribution of the queuing delay percentiles over all peers. Intuitively, the 90, 95 and 99-th percentiles statistics are related to the queuing delay experienced by a peer during the most highly loaded 10%, 5% and 1% observation windows respectively.

While Fig. 4 contrasted per-packet q_i vs windowed Q_i queuing delay *considering all samples over all peers*, Fig. 5 instead considers each peer *once per each experiments*. Let us consider the 90-th percentile statistics for the sake of the example. During a single experiment, for each remote peer P we gather multiple observations of windowed queuing delay statistics Q_i^P, based on which we compute the 90-th queuing delay percentile for peer P: Fig. 5 reports the CDF distribution of 90-th queuing delay percentile over all peers, over all experiments.

First, notice that part of the samples report a very low queuing delay (i.e., below 1ms): these corresponds to cases where external leechers receive data from our probes, and we are receiving in turn only an acknowledgement stream. In this case, since queuing is in our upstream, the low-bandwidth acknowledgement stream is not incurring any access bottleneck and thus traverse an empty queue. Notice that, additionally, the remote peer is also not sending data to other peers either, as otherwise we would observe a rise in queuing delay.

Otherwise, we gather that 99% of the windows of the median peer have a queuing delay below 60 ms. Moreover, only 10% of the peers experience a 90-th (99-th) percentile above 100 ms (200 ms). In other words, for the 10% of peers that are most affected by bufferbloat, only 10% (1%) of their 1-second windows incurr more than 100 ms (200 ms) queuing delay. Finally, only 1%

TABLE II
FEATURE DESCRIPTION: ACCESS TYPE (AT), OPERATING SYSTEM (OS), BITTORRENT CLIENT (BC)

Feature	Class	Details	Cardinality
AT	Low-speed	DSL	2660
	High-speed	Cable	245
		Fiber	152
		Total	*3057*
		Unclassified	*1144*
OS	Windows	WinXP/200x/Vista/7	5668
		Win95	897
	Other	MacOS/Linux<2.x/BSD	254
		Linux>2.x	77
		Total	*6896*
BC	uTorrent	uTorrent	3552
	Other	Vuze	345
		Transmission	14
		Azureus	11
		Total	*3937*

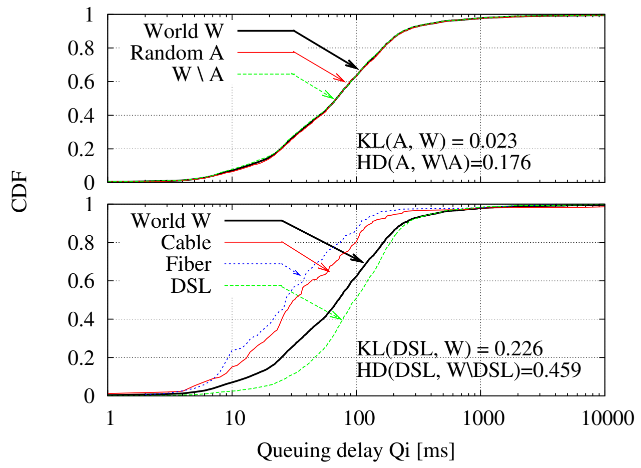

Fig. 6. Breakdown of queuing delay CDF according to random splitting feature (R, top) or access type (AT, bottom). Plots are annotated with HD and KL statistics.

of peers have a 90-th (99-th) queuing delay percentile above 1 s (2 s).

Summarizing, from Fig. 5 we gather that (i) the bulk of peers experience queuing delays that are generally non-harming for interactive traffic: i.e., for 90% of peers, 90% of windows have a queuing delay smaller than 100 ms. At the same time (ii) some users may perceive the rest of their activities annoyed by the ongoing BiTorrent transfers, since about 1% of the peers have at least 10% of windows have a queuing delay exceeding 1 s.

Coupling findings in Fig. 5 and Fig. 4 (showing that 1% of all windows over all peers can reach up to 10 s worth of delay), we understand that these high-delay windows mostly concern a small population of very active BitTorrent users, whose Internet activities can suffer from significant bufferbloat. Since a single TCP connection is able to fill the buffer, LEDBAT transfers to our probe are however not the cause of their bufferbloat, which is rather tied to TCP uploads to some other peers in the swarm.

B. *Root cause analysis*

We now adopt the user viewpoint, and assess the importance of different choices she can make to relieve her bufferbloat problem. At high level, users can select an ISPs and access technology (e.g., subscribing DSL or Fiber contracts depending on her budget), are free to adopt one of the several operating systems available on the market (some of which free of charges), and can install one of the many BitTorrent clients (most of which are free of charges).

We thus now formally analyze the impact on the bufferbloat of external factors such as the BitTorrent client (BC), the host Operating System (OS) and access type (AT). To do so, we need to correlate queuing delay statistics, gathered by applying our methodology to the raw packet-level traces, with additional information. To this purpose, during the BitTorrent transfer the probe also collected complementary data for each contacted peers. In more details, we infer (i) the access type from reverse DNS queries, (ii) the operating system type by OS fingerprinting based on the IP TTL field [5] and (iii) the BitTorrent client by parsing the L7 BitTorrent handshake message.

For instance, reverse DNS queries sometimes provide us clues about the peers' access network type (e.g., when domain label explicitly contains keywords such as `ftth`, `fttc`, `cable`, `adsl`, `vdsl`, etc.), letting us compare the typical bufferbloat incurred by DSL vs Cable vs Fiber users. For the sake of the example, we report a breakdown of the queuing delay statistics according to the access type in the bottom plot of Fig. 6, and label as World the statistics over all samples irrespectively of their access type. As expected, DSL peers experience worse queuing delay with respect to Cable and especially Fiber users.

We point out that this additional information is not always available. We report the exact cardinalities of the annotated dataset per each feature in Tab. III. Notice for instance that DNS reverse information were available for only a subset of IPs (3057 hosts), and that furthermore some of the domain names provided by DNS response did not provide labels that were explicitly tied to the access type of an host (1144 unclassified).

To make the analysis simpler, for each of the AT, BC and OS features, we consider two classes: namely, we divide AT in low vs high speed access, OS in windows vs other operating systems, and BC in uTorrent vs other clients. For the sake of comparison, we additionally consider a World set (W), that is representive of all samples, irrespectively of any feature or class, and that is based on the full peer population.

We then compute queuing delay statistics for each class, as in Fig. 6, and resort to standard metrics to compare differences in the distribution. More precisely, we make either (i) class-to-class comparison using the Hellinger Distance or (ii) class-to-world comparison using Kullback-Leibler divergence.

$$HD(A, B) = \sqrt{1 - \sum_{x \in X} \sqrt{A(x)B(x)}} \tag{5}$$

$$KL(A, W) = \sum_{x \in X} A(x) \ln \frac{A(x)}{W(x)} \tag{6}$$

Let us denote with A, B and W the probability distribution functions of the windowed queuing delay metric for classes A, B and world respectively. Hellinger distance (HD), defined in (5), is a score of similarity between distributions. Values of distance between two classes ranges in $HD(A, B) \in [0, 1]$, with lower values corresponding to higher similarity. Additionally, HD is a symmetric function, so that a single value $HD(A, B) = HD(B, A)$ quantifies the distance between classes. The Kullback Leibler (KL) divergence, defined in (6), is a measure of the difference between two probability distribution, and is related to the amount of information encoded in A, W. Since KL is non-

TABLE III
FEATURE ANALYSIS: HELLINGER DISTANCE(HD), KULLBACK LEIBLER (KL), MEDIAN (P50) AND 90-TH (P90) QUEUING DELAY PERCENTILES

Feature	Class	HD	KL	p50 [ms]	p90 [ms]
R	Rand A	0.176	0.023	70	230
	$W\backslash A$		0.023	70	240
AT	Low-speed	0.459	0.227	94	240
	High-speed		0.333	30	120
OS	Windows	0.341	0.141	55	220
	Other		0.308	31	190
BC	uTorrent	0.349	0.135	77	220
	Other		0.291	39	190

symmetric, we compute both $KL(A, W)$ and $KL(B, W)$.

For the sake of the example Fig. 6 reports the HD and KL scores in two cases. For reference purposes, top plot consider two randomly generated classes (namely, a random subset A and its complement $W\backslash A$): the HD and KL scores obtained in this case are representive of the lowest possible level of correlation between the feature and the queuing delay distribution. As it can be seen, queuing delay CDFs for random classes A and $W\backslash A$ overlap with the world set W, that yield to the minimum distance scores of $HD = 0.176$ and $KL = 0.023$. Bottom plot of Fig. 6 reports instead HD and KL scores for the access type feature: in this case, differences in the CDFs are evident and translate into higher values of $HD = 0.459$ (about 3 times bigger than in the random case) and $KL = 0.226$ (about 10 times bigger).

Tab. III reports HD and KL scores and queuing delay percentile for all features, including random partitioning as a reference. To avoid a class imbalance bias (recall Tab. II), we compute average HD and KL over 10 samples of the largest class having the same cardinality of the smallest class. For the sake of clarity, consider the access type feature, where the low-speed class have 2660 samples, whereas the high-speed class has only 397 samples: in this case, we compute HD and KL over 10 subsets of 397 samples from the low-speed class. From Tab. III it can be seen that, as expected, the access type feature plays the biggest role in determining the bufferbloat extent: the class-to-class difference scores $HD = 0.459$, median delays are 3 times higher in the low-speed class, and 90-th delay percentiles are 2 times higher. Interestingly, the operating system and BitTorrent client have a sizeable role as well, with furthemore very similar magnitudo (i.e., similar HD, KL and delay percentiles).

As for the BC feature, difference in the connection management policies among clients may have an impact on the bufferbloat. For instance, uTorrent handles the dual TCP/LEDBAT stack by simultaneously opening a TCP and a LEDBAT connection: only in case the latter open is succesful, then the TCP connection is dropped. Diffent dual stack management policies (e.g., serial rather than parallel open, preference of either TCP or LEDBAT, etc.) can clearly affect the queuing delay statistics. As for the OS feature, we further point out that flavors of TCP differ among operating systems (with TCP NewReno for old Windows version, Compound for Windows since Vista and Cubic for Linux), whose different congestion window dynamics clearly affects the queuing delay statistics. Finally, other hidden factors may play an important role. For instance, the number of torrents on which external peers are participating at the same time (incresing the number of simultaneous connections, and the size and diversity of the neighborhood), whether a peer is leecher or seed on those torrents (seed always have content and may experience higher delay), etc. Such factors are clearly unavoidable, but even a larger-scale study involving more features could have hard times in precisely isolating the impact of each factor.

V. RELATED WORK

Recently, work started characterizing user bufferbloat [11], [12]. Netalyzr [12] measures the bufferbloat with an active methodology that saturates the uplink/downlink during short periods of time, so that maximum queuing delays are measured. Differently from [12], our methodology does not require end-user cooperation and can continuously monitor the actual user bufferbloat. Bufferbloat problems in 3G/4G networks are exposed in [11] by instrumenting devices to perform RTT measurement – RTT could possibly couples the sum of two queueing delays, unlike in our work where we precisely measure a single queue.

Recent experimental work on LEDBAT and BitTorrent [6], [13] focuses on the congestion control properties of LEDBAT [13], or exploits BitTorrent for crowdsourcing ISP characterization [6] mainly focusing on datarates – our work is in line with this approach, that it complements with queuing delays information.

VI. AFTERMATH

This paper analyze the bufferbloat experienced by BitTorrent users: it seems that while LEDBAT alleviates the problem for most users, it does not solve it entirely, as some users may still experience significant queuing delays. Additionally, we find that several factors such as access type, operating system and BitTorrent client may concurr in determining user QoE degradation.

As future work, we aim at applying the methodology at large, either e.g., integrating the methodology in Dasu [6], that has been installed by over 500K peers in 3K networks, or by conducting a large scale passive measurement campaign with Tstat.

ACKNOWLEDGEMENT

This work has been 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://tstat.tlc.polito.it.
[2] http://www.telecom-paristech.fr/~drossi/dataset/bufferbloat-internet.
[3] http://bittorrent.org/beps/bep_0029.html.
[4] ITU Recommendation G.114, One Way Transmission Time.
[5] Dynamic probabilistic packet marking for efficient IP traceback. *Computer Networks*, 51(3):866 – 882, 2007.
[6] Z. Bischof, J. Otto, M. Sánchez, J. Rula, D. Choffnes, and F. Bustamante. Crowdsourcing ISP characterization to the network edge. In *ACM SIGCOMM W-MUST'11*, 2011.
[7] V. Cerf, V. Jacobson, N. Weaver, and J. Gettys. Bufferbloat: what's wrong with the internet? *Communications of the ACM*, 55(2):40–47, 2012.
[8] S. Cheshire. It's the latency, stupid! http://rescomp.stanford.edu/~cheshire/rants/Latency.html, 1996.
[9] C. Chirichella, D. Rossi, C. Testa, T. Friedman, and A. Pescape. Remotely gauging upstream bufferbloat delays. In *PAM*, 2013.
[10] L. DiCioccio, R. Teixeira, M. Mayl, and C. Kreibich. Probe and Pray: Using UPnP for Home Network Measurements. In *PAM*, 2012.
[11] H. Jiang, Y. Wang, K. Lee, and I. Rhee. Tackling bufferbloat in 3G/4G networks. In *ACM IMC*, 2012.
[12] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson. Netalyzr: Illuminating the edge network. In *ACM IMC'10*, 2010.
[13] D. Rossi, C. Testa, and S. Valenti. Yes, we LEDBAT: Playing with the new BitTorrent congestion control algorithm. In *PAM*, 2010.
[14] S. Shalunov et al. Low Extra Delay Background Transport (LEDBAT). IETF draft, 2010.