Passive bufferbloat measurement exploiting transport layer information

C. Chirichella[1], D. Rossi[1], C. Testa[1], T. Friedman[2] and A. Pescape[3]
[1]Telecom ParisTech – first.last@enst.fr
[2] UPMC Sorbonne Universites – timur.friedman@upmc.fr
[3] Univ. Federico II – pescape@unina.it

***Abstract*—"Bufferbloat" is the growth in buffer size that has led Internet delays to occasionally exceed the light propagation delay from the Earth to the Moon. Manufacturers have built in large buffers to prevent losses on Wi-Fi, cable and ADSL links. But the combination of some links' limited bandwidth with TCP's tendency to saturate that bandwidth results in excessive queuing delays. In response, new congestion control protocols such as BitTorrent's uTP/LEDBAT aim at explicitly limiting the delay that they add at the bottleneck link. This work proposes a methodology to *monitor the upstream queuing delay experienced by remote hosts*, both those using LEDBAT, through LEDBAT's native one-way delay measurements, and those using TCP, through the Timestamp Option. We report preliminary findings on bufferbloat-related queuing delays on an Internet measurement campaign involving a few thousand hosts.**

I. Introduction

As a recent *CACM* article points out, "Internet delays now are as common as they are maddening" [12]. The combination of *bufferbloat*, or excessive buffer sizes, with TCP's congestion control mechanism, which forces a bottleneck buffer to fill and generate a loss before the sender reduces its rate, is the root cause for these delays. The problem is well known in the scientific community [13], but has worsened lately with the ever-larger buffers that manufacturers place in equipment that sits in front of low-capacity cable and ADSL uplinks. Bufferbloat-related queuing delays can potentially reach a few seconds [26].

The designers of BitTorrent, aware that their application's multiple TCP streams had a tendency to saturate upstream links to the detriment of other applications (such as VoIP or games) created uTP, a protocol with a congestion control mechanism that backs off in response to delays. The mechanism is now evolving under the auspices of the IETF in the form of the Low Extra Delay Background Transport (LEDBAT) protocol [29], which aims to deflate bufferbloat delays through efficient but low priority data streams. LEDBAT could be used for any sort of bulk data transfer, which is now commonplace as users upload video (YouTube, DailyMotion, etc.), pictures (Picasa, Flickr, etc.), music (GoogleMusic, etc.), and other forms of data (Dropbox, Facebook, etc.) to the Cloud. LEDBAT assumes that the bottleneck is on the upstream portion of an ADSL or cable access link, so that congestion is self-induced by a user's own traffic competing with itself. To maintain user QoE and avoid harming VoIP, gaming, Web, or other ongoing transfers, LEDBAT employs a delay-based congestion control algorithm. When LEDBAT has exclusive use of the bottleneck resources, however, it fully exploits the available capacity.

Like TCP, LEDBAT maintains a congestion window. But whereas mainstream TCP variants use loss-based congestion control (growing with ACKs and shrinking with losses), LEDBAT estimates the queuing delay on the bottleneck link and tunes the window size in an effort to achieve a target level of delay. The dynamic of TCP's AIMD window management systematically forces the buffer to fill until a loss occurs and the window size is halved. Recent studies [17] show that most home gateways have a fixed buffer size, irrespective of the uplink capacity. With cable and ADSL modem buffers ranging from, on average, 120 KB to a maximum of 365 KB [17], and common uplink rates of 1 Mbps, worst case TCP queuing delays range from 1 second on average to a maximum of 3 seconds. LEDBAT, by contrast, protects VoIP and other interactive traffic by targeting a delay cap of 100 ms [5]. By choosing a non-zero target, LEDBAT also ensures that capacity is fully exploited.

Although TCP's loss-based congestion control, coupled with large buffers, can clearly cause significant bufferbloat delays, it is unclear how often this happens in practice, and how badly it hurts user performance. Gettys describes [20] how the problem would disappear when he went about trying to investigate its root cause. Active approaches, such as Netalyzer [26], are likely to overestimate delay magnitude: by purposely filling the pipe, Netalyzer learns the maximum bufferbloat delay, but not its typical range.

Our main contribution is the design and validation of a methodology for inferring the queuing delays encountered by remote LEDBAT and TCP hosts. It is based on passive analysis of the timestamp information carried by either application-layer messages (LEDBAT) or transport-layer segments (TCP). This approach adhere to the literature's vision that suggests ISP characterization should be done at scale, continuously, and from end users [9]. Our methodology complements their measurements of achievable rates with estimations of the delays that end-users actually encounter.

We validate our methodology in a local testbed, against different variants of ground truth: kernel level, application logs, and traffic probes. Despite our main focus in this paper is to propose and validate the methodology, we also briefly report on experiments in the wild Internet (where we use BitTorrent as a case study, since it supports both the LEDBAT and TCP

protocols and is also amenable to large scale ISP performance characterization [9]), though we refer the reader to [14] for a more detailed quantitative analysis. In principle, the BitTorrent results should be representative of the bufferbloat delays that end-hosts will experience from other upload-intensive applications, like the one cited above.

II. Related work

Several branches of work relate to ours. Some work *points out the existence of the bufferbloat problem* [13], [20], [26]. This includes early work from the 90s [13] and more recent work [20]. Netalyzer [26] offers Web-based software able to analyze several features of users' home connections, including bufferbloat size. Our work differs from Netalyzer in that it does not require end-user cooperation. Our methodology is able to evaluate the user's buffer queue without interfering. Other work *offers a solution to the bufferbloat problem* [10], [18], [29], [30]. Solutions can be broadly divided into two categories, namely active queue management (AQM) techniques and end-to-end congestion control (E2E). AQM techniques [18] selectively schedule/mark/drop packets to both reduce queue size and induce reactions from E2E control algorithms. Despite initial interest, adoption of AQM has been slow so far, with limited deployment in user access gateways [12]. E2E solutions [10], [29], [30] generally employ a delay based controller; TCP Vegas [10] is the most famous example, to which NICE [30] and LEDBAT [29] are very much indebted. However, while TCP Vegas aims to be more efficient than TCP NewReno, both NICE and LEDBAT aim to have lower priority than TCP. The main difference between NICE and LEDBAT is that NICE reacts to round-trip time (RTT), and LEDBAT to one-way delay (OWD) difference. Further work *takes an experimental look at LEDBAT* [15], [28] (while yet other work adopts a simulation approach). This work addresses different questions from ours, such as testbed-based evaluation of early versions of the protocol [28], or algorithms to improve OWD estimates in light of clock skew and drift [15]. We are the first to use LEDBAT (and TCP) to remotely gauge bufferbloat delays, and thus also the first to experimentally validate such an approach. Finally, there is work that *examines BitTorrent through experimentation* [9], [21], [22], [27]. Early measurement work on BitTorrent dates back to [22], focused on the properties of an entire swarm over a long period of time. Closer work to ours, instead exploits it for end-host measurements [9], [21], [27]. In more detail, [21] focuses on capacity estimation, while [27] on peer availability. More recently, [9] proposed to use BitTorrent as a service for crowd-sourcing ISP characterization from the edge of the network via a BitTorrent plugin called Dasu. The plugin is successful in obtaining information from a large number of peers and vantage points (i.e., 500K peers in 3K networks), but it mainly focuses on achievable data rates for BitTorrent peers. We, in contrast, propose a methodology for enriching the information available about end-hosts. Namely, while previous work focused on bandwidth, we argue that in light of the bufferbloat problem, the delays actually experienced by users should receive greater attention – hence the methodology we propose. Finally, we point out that in very recent times, work started to appear that *explicitly focuses on bufferbloat measurement,* trough either passive [7], [8], [14], [19] or active measurement [11], [16], [25]. While for reason of space we cannot fully contrast these work, we refer the interested reader to [8] and [11] for such an analysis, having a passive and active measurement focus respectively.

III. Methodology

We estimate queuing delay by collecting one-way delay (OWD) samples, establishing the minimum as a baseline delay, and then measuring the degree to which a sample differs from the baseline. This is a classic approach used in congestion control to drive congestion window dynamics. Early work on this subject date back in late 80's, as the Jain's CARD approach [24], for pro-actively adapts the congestion window using the network understanding as it approaches congestion. Our innovation is to demonstrate how a passive observer of LEDBAT or TCP traffic can use this approach to estimate the uplink delays experienced by a remote host. We make our source code available at [1].

To establish differences from a baseline, one only requires notional OWD values, as opposed to actual values of the sort that one could measure with synchronized clocks. A packet that encounters empty queues will incur just the fixed components of OWD: propagation, transmission, and processing. In the absence of factors such as clock drift (which can be controlled for, e.g. [15]), clock resets, or routing changes, the only variable component is queuing delay. Figs. 1(a) and 1(b) illustrate the view of a passive observer of, respectively, LEDBAT and TCP packets flowing between a nearby host A and a remote host B. Three packets, i–1, i, and i+1 are shown. Based upon information directly available in LEDBAT packet headers and in the Timestamp Option of TCP packets, once it has seen packet i+1 (the moment is marked by a star), the observer is able to estimate the queuing delay experienced by packet i. These delays may result from all upstream traffic from B, including flows other than the one being observed (possibly other than A).

A. LEDBAT.

In the absence of a finalized LEDBAT standard, our protocol parser is based on BitTorrent's currently implemented BEP-29 definition [2]. This specifies a header field, named "timestamp" in Fig. 1(a), that the sender timestamps based on 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(a). In this way, each acknowledgement packet conveys the estimated OWD incurred by the most recently received data packet, and this information is used to grow/shrink the congestion window [29].

An observer close to A sniffs the packets and performs the same state updates as does the LEDBAT congestion control protocol running on A. Notice that there is no need for the

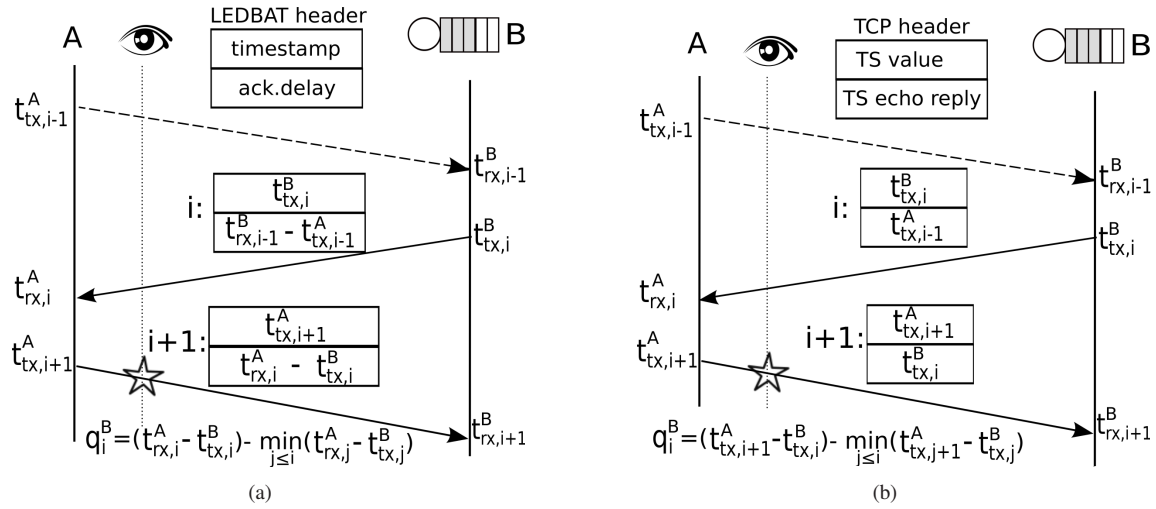

Fig. 1. A passive observer can infer packet i's queuing delay coming from B. LEDBAT case in the left plot, eqn. (2); TCP case in the right plot, eqn. (4).

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:

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

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

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 a notional value, dependent upon the unknown offset between the clocks at B and A (or the observer), the queuing delay is independent of the offset. Note that the observer could also use these techniques to estimate A's queuing delays, provided that it is upstream of A's bottleneck queue.

B. TCP.

Collecting OWD samples from observation of a TCP flow is more complicated than for LEDBAT for several reasons. To begin with, TCP congestion control is driven by inference of losses, not delays, so timestamps are absent from the default TCP header. To obtain timing information, the end hosts must have negotiated use of the Timestamps Option, introduced by RFC 1323 [23]. This means the observer must either be one of the hosts, work in cooperation with one of the hosts, or opportunistically measure only those flows that have this option enabled.

Then, timestamp units may differ from flow to flow, as RFC1323 only requires that the "values must be at least approximately proportional to real time." A factor ϕ is needed to convert timestamp values into real clock times. The factor is related to the kernel's ticking frequency and defaults to 4 for (recent) Linux releases and to 10 for Windows OS. As an observer does not in general know the OS of the remote host, we use already-developed OS fingerprinting techniques based on IP TTL values [6]. Another issue is that the TCP header, even with the Timestamp Option enabled, does not carry OWD samples directly, as the LEDBAT header does. Rather, it carries a timestamp from the TCP sender, and, when the packet is an ACK, an echo of the timestamp of the packet being ACKed. The purpose is to facilitate calculation of round trip time (RTT) rather than OWD. Though both of these timestamps mark packet sending times, with no explicit indication of packet receipt times, an observer can nonetheless estimate OWD. Referring to Fig. 1(b), if we assume that packet i+1 is issued as an ACK very shortly after A receives data packet i from B then we can use $t^A_{\mathrm{tx},i+1}$ instead of $t^A_{\mathrm{rx},i}$ for the purpose of calculating OWD samples received on $B \to A$:

$$\beta_{BA} = \min(\beta_{BA}, t^A_{\mathrm{tx},i+1} - t^B_{\mathrm{tx},i}), \quad (3)$$

$$q^B_i = \phi\big[(t^A_{\mathrm{tx},i+1} - t^B_{\mathrm{tx},i}) - \beta_{BA}\big] \quad (4)$$

This assumption is justified in most cases, as packet processing time is, as a general rule, much smaller than propagation time. Even were processing time to inflate OWD calculations by a fixed amount, this would not pose a problem for estimating differences from baseline OWD. However, there are two circumstances in which $t^A_{\mathrm{tx},i+1}$ potentially diverges from $t^A_{\mathrm{rx},i}$: (i) delayed ACKs and (ii) holes in the sequence number space.

When (i) delayed ACKs are in use, RFC 1323 specifies that the receiver echo the timestamp of the *earliest* unacknowledged data packet. The timestamp difference between the ACK and this earlier packet would overestimate OWD. We avoid this by correlating the ACK segment with the *last* data segment the ACK refers to, exploiting sequence and ACK number. When there are (ii) holes in sequence number space, the ACK for the out-of-order packet echoes the most recent packet that advanced the window – which is not necessarily the most recently received packet. Since the solution is in this case more tricky than in the previous one, and since out-of-order are infrequent, we merely discard OWD samples of out-of-order packets.

IV. Validation

We validate our methodology in a small testbed, of which we make the gathered packet level traces and logs available at [1]. Space constraints limit us to describe the most challenging

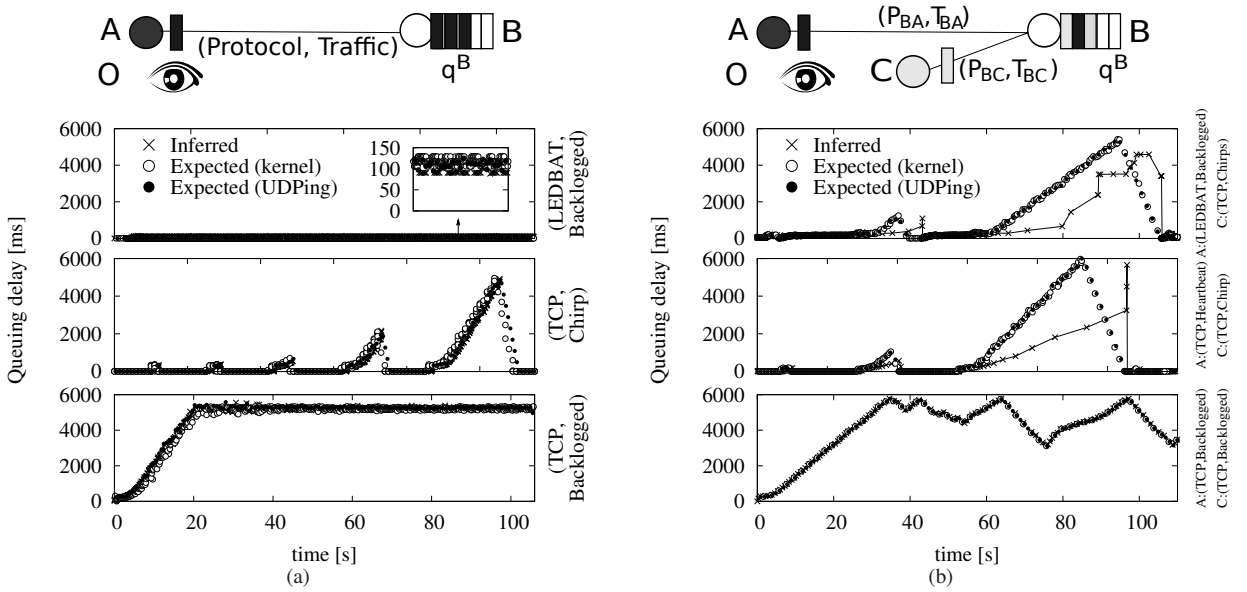

Fig. 2. Left plot: queueing delay at B, with an ongoing transfer $B \rightarrow A$: from top to bottom (LEDBAT,Backlogged), (TCP,Chirp) and (TCP,Backlogged) cases. Right plot: queueing delay with an additional "'hidden" transfer $B \rightarrow C$.

scenario, in which an observer close to (or co-located with) A estimates B's upstream queuing delay (complete results in [1]). We compare estimated delay against a measured delay ground truth, under several traffic models, with and without "hidden" traffic. The testbed comprises two Linux machines, each running a 2.6.38 kernel. The Hierarchical Token Bucket (HTB) of `tc` emulates access link bottlenecks, limiting both machines' uplinks to 1 Mbps. Traffic is captured by the `tcpdump` sniffer. We use the default CUBIC version of TCP, and the uTorrent 3.0native application-level Linux implementation of LEDBAT over a UDP framing that is provided by BitTorrent.[1] To generate LEDBAT traffic between the hosts, we set up a private tracker and let A and B join as the unique leecher and seed respectively. We compare the delay estimates generated by our passive methodology with two ground truths. First, we log the queuing delay in the Linux kernel, modifying the `sch_print` kernel module of the `netem` network emulator. Second, we send UDP[2] echo request/reply probes with UDPing [4] to get an estimation of the RTT between A and B. Since only acknowledgements are traveling back from A to B, the queuing occurs only at B, allowing us to gather yet another estimate of $q_i^B = RTT_i - min_{j \leqslant i} RTT_j$.

A. Without hidden traffic

In this scenario, the seed B sends traffic to A and no other traffic is sent from B to other hosts. Hence, an observer O close to A observes all the traffic sent by both A and B. Packets are sent $B \rightarrow A$ using transport protocol $P \in \{\text{TCP}, \text{LEDBAT}\}$. The application generates packets for the transport protocol according to traffic model $T \in \{\text{Backlogged,Chirp}\}$. In Backlogged traffic, applications always have data to send. In Chirp traffic, data generation follows an ON/OFF behavior, with a deterministic start time for ON periods (every 30 seconds) and an exponentially growing number of packets to send during the ON periods. While the Backlogged traffic model (as implemented by, e.g., Netalyzer [26]) forces bufferbloat queuing to reach its maximum, we want our methodology to be able to closely follow any scale of buffer inflation, leading us to use the Chirp traffic model.

Fig. 2(a) shows the temporal evolution of the queuing delay estimate and ground truths. The top plot reports the case $(P, T) =$ (LEDBAT,Backlogged), from which we see that, as expected, queuing delay at B reaches the 100 ms target specified in the LEDBAT draft. The middle plot reports the (TCP,Chirp) case, in which we see that, depending on the amount of data that the application gives to the transport layer, the queuing delay can possibly grow very large (up to about 5 seconds in our setup). Finally, the bottom plot reports the (TCP,Backlogged) case, in which the queuing delay grows up to the maximum value and then flattens as the TCP sender is continuously sending data. In all cases, we see that our methodology is very reliable w.r.t. both the kernel log and the UDPing ground truths: differences between the inferred vs. expected queuing delay are on the order of 1 packet of queuing delay for LEDBAT, a few packets for TCP.

B. With hidden traffic

In a typical scenario, however, the observer O will be able to observe *only part* of the traffic generated by the host of interest B (say, the traffic $B \rightarrow A$), but will miss another part (say, $B \rightarrow C$). Nevertheless, our methodology should allow the observer to get an unbiased view of B's queue occupancy,

[1]We do not use libUTP [3], which in our experience is an older implementation still affected by some bugs. Notably, libUTP sometimes resorts to the use of 350 byte packets, which leads to unstable behavior, and has been ruled out from later uTorrent versions [28].

[2]We prefer to use UDP, since it is well known that ICMP traffic is handled differently w.r.t. TCP and UDP by some devices/OSs.

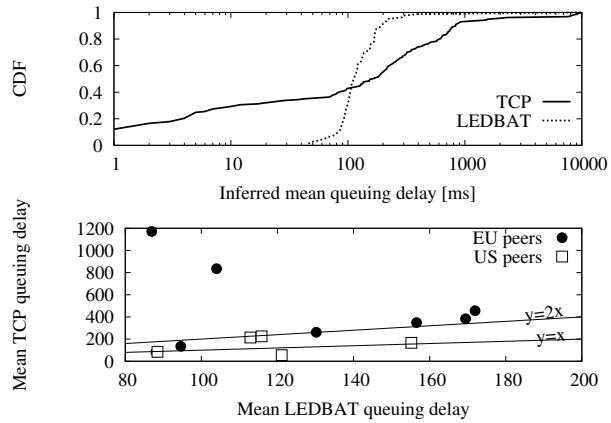

Fig. 3. Internet bufferbloat queuing delays: CDF of mean per-peer delay (top), and scatter plot of LEDBAT vs. TCP delays at the same peer (bottom).

at least provided that a sufficient number of samples reach A. In this case, we have two degrees of freedom for describing our setup, as we not only need to define the $B \rightarrow A$ exchange in terms of the protocol and traffic-model pair (P_{BA}, T_{BA}), but also need to characterize the $B \rightarrow C$ exchange in terms of (P_{BC}, T_{BC}). Here, we use an additional Heartbeat traffic model, behaving similarly to an ICMP echo request/reply or UDPing, and sending periodic data packets. The rationale behind the Heartbeat model is as follows. Since LEDBAT is a low priority protocol, when B is sending TCP traffic to C, any backlogged LEDBAT transmission toward A will sense a growing delay and reduce its sending rate, possibly down to a minimum of one packet per RTT. In this case, the number of samples available to the observer will be the minimum possible under LEDBAT. To perform a fair comparison under TCP, we therefore need to tune the TCP sending rate in a similar manner.

The top plot of Fig. 2(b) reports the case where B is sending backlogged LEDBAT traffic to A and is sending TCP Chirp traffic to C. Several observations can be made from the plot. First, notice that the time at which the observer can infer the queuing delay q^B is different from the ground truth reference time t_G: the kernel logs dequeue operations, while in the case of UDPing we correlate the RTT information with the time the first packet of the pair was sent. In the case of LEDBAT, the packet needs to reach the observer prior to q^B being inferred. While the observer knows that the queuing delay sample is received at about $t_G + q^B$, we prefer not to correct the time to better stress differences in the plot. Second, notice that the number of valid samples at A is lower than in the previous case. This is a consequence of two facts: (i) the LEDBAT sending rate reduces to about 1 sample per RTT under Chirp; (ii) some of the packets trigger a retransmission timeout in LEDBAT, and we may receive duplicate samples that, as explained earlier, we conservatively filter out. Third, and most important, even if only very few packets make it to the receiver A (which is already an implicit signal of congestion on the path from B to A), A is still able to extract valuable timing information from these samples, which at least lets it evaluate the magnitude of the bufferbloat delays.

Comparing the top plots of Fig. 2(a) to Fig. 2(b), from just a handful of packets A infers that queuing delay at B jumps from about 100 milliseconds to possibly 6 seconds. The middle plot of Fig. 2(b) reports a similar case, where this time queuing delay inference is performed over TCP Heartbeat traffic $B \rightarrow A$, while B is sending TCP Chirp to C. Again, A is able to correctly infer that a significantly large queue q^B is building up although he has no knowledge of the traffic $B \rightarrow C$. Finally, the bottom plot of Fig. 2(b) reports the case where both $B \rightarrow A$ and $B \rightarrow C$ are TCP backlogged transmissions. In this case, though the traffic is equally split between A and C, A is able to perfectly infer the queuing delay at B. We conclude that our methodology is able to reliably infer the queuing delay at B from the inspection of LEDBAT or TCP traffic with the Timestamp Option enabled. The error in the inferred measure is negligible in cases where a sizable amount of traffic makes it to the observer, but is still robust and reliable even when the observer is able to sniff only very few samples.

V. Internet Experiments

Finally, we report on preliminary experiments to gauge the degree of actual bufferbloat queuing in the uncontrolled Internet. We point out that, while these results do not aim at providing an exhaustive coverage of the bufferbloat in the Internet, they nevertheless already bring valuable insights of the current standpoint. For further experimental results, we refer the reader to [14], where we collect complementary data, such as reverse DNS queries (this would provide us with clues as to peers' access network types) and lightweight OS fingerprinting, to provide a root cause analysis of the bufferbloat. We make the packet level traces available at [1].

We use BitTorrent, as it allows us to probe a large number of hosts worldwide. Our BitTorrent clients exchanged with 20,799 peers, 8,688 of which transferred data using LEDBAT and 12,111 of which were using TCP. Experiments were performed over a 4-month period during 2012, principally using the uTorrent 3.0 and Transmission clients in their default configurations. We run these clients from vantage points in Italy, France, and Austria, from which we obtained about 20 GB of raw traces. The clients joined 12 legal torrents from Mininova, spanning different *categories* (2 eBooks, 2 games, 1 software, 2 documentaries, 1 podcast and 3 videos), *file sizes* (from 56 MB for the smallest eBook to 1.48 GB for the largest video, totalling 6.4 GB of data overall), and *number of seeds* (from 30 for an eBook to 9,635 for a video).

In these Internet experiments, we did not have access to the remote hosts. This means that we were deprived of the sources of ground truth that we used for validation purposes: we could not look at the kernel logs (requiring root access to the host), and we could not obtain UDPing logs (requiring host collaboration). We took two alternative approaches to obtaining reliable results: conservative filtering out exchanges for which we have little data, and carefully comparing queuing delay under LEDBAT and TCP for the same hosts.

Filtering approach. We filtered our results by focusing only on the subset of peers with which one of our clients exchanged at

least 100 packets, i.e., 50 queuing delay samples per peer. This left us with 2,052 LEDBAT peers and 987 TCP peers, which is still a significant, and we hope representative, portion of the entire population. The top plot of Fig. 3 shows the distribution of mean queuing delay for each type of peer.

Interestingly, we see a sharp increase of the LEDBAT CDF between 100 ms and 200 ms. This suggests that LEDBAT is at least partially effective in limiting bufferbloat queuing delays for long transfers. Some LEDBAT transfers however report queuing delays exceeding 1,000 ms. This is likely due to hidden TCP traffic directed to unobservable peers, as we had seen during validation (see Fig. 2(b)). The lesson that we take from this observation is that LEDBAT by itself would not be sufficient to eliminate bufferbloat delays, unless it were adopted globally.

An interesting feature of the TCP curve is that a significant portion of peers experience low queuing delay. Our best explanation is that these are hosts that have enabled bandwidth shaping precisely to avoid bufferbloat delays. We note that the benefit they gain from reduced delay comes at the expense of reduced bandwidth usage, and hence longer swarm completion times. The very lowest delays, we believe, reflect phenomena other than queuing in the modem, such as interrupt coalescing, uncertainty due to timestamp precision, token bucket burst size of the application layer shaper, etc.

Comparative approach. We run experiments back-to-back using the same client with LEDBAT alternately enabled or disabled – in the latter case forcing peers to use TCP. Since we cannot guarantee which peer the client will connect with, we looked for cases in which *the same peer* participated in both LEDBAT and TCP transfers.

We found only 11 peers (7 in Europe and 4 in the US) that did so. The bottom plot of Fig. 3 shows their mean queuing delay under LEDBAT and TCP as a scatter plot, along with two reference lines: for a TCP delay either equal to or double the LEDBAT delay. Interestingly, we found that for 8 of the 11 peers, queueing delay is double or worse under TCP, with significant congestion in some cases (e.g., mean delay above 800 ms).

VI. Discussion

We have presented a methodology to gauge upstream bufferbloat queuing delays at remote peers by observing timing information conveyed in LEDBAT or TCP headers. Validation on a controlled testbed confirms the methodology to be reliable even with very few observations. Preliminary experiments in the uncontrolled Internet confirm that (i) LEDBAT use is already fairly widespread among BitTorrent clients, (ii) partial LEDBAT adoption is insufficient to eliminate bufferbloat delays in the presence of TCP peers, and (iii) LEDBAT can at least reduce average delays.

Our ongoing work focus on development of alternative, more general, techniques for passive bufferbloat measurement that do not rely on TimeStamp information on packet headers [8], or active techniques for large scale Internet measurement [11].

Acknowledgement

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

References

[1] http://www.enst.fr/~drossi/dataset/bufferbloat-methodology.
[2] http://bittorrent.org/beps/bep_0029.html.
[3] http://github.com/bittorrent/libutp.
[4] http://perform.wpi.edu/tools/tools.
[5] ITU Recommendation G.114, One Way Transmission Time.
[6] Dynamic probabilistic packet marking for efficient IP traceback. *Computer Networks*, 51(3):866 – 882, 2007.
[7] M. Allman. Comments on bufferbloat. *SIGCOMM Comput. Commun. Rev.*, 43(1), Jan. 2012.
[8] A. Araldo and D. Rossi. Bufferbloat: passive inference and root cause analysis. Technical report, Telecom ParisTech, 2013.
[9] 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.
[10] L. Brakmo, S. O'Malley, and L. Peterson. TCP Vegas: new techniques for congestion detection and avoidance. *ACM SIGCOMM CCR*, 24(4):24–35, 1994.
[11] P. Casoria, D. Rossi, J. Auge, M.-O. Buob, T. Friedman, and A. Pescape. Distributed active measurement of internet queueing delays. Technical report, Telecom ParisTech, 2013.
[12] 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.
[13] S. Cheshire. It's the latency, stupid! http://rescomp.stanford.edu/~cheshire/rants/Latency.html, 1996.
[14] C. Chirichella and D. Rossi. To the moon and back: are internet bufferbloat delays really that large. In *IEEE INFOCOM Workshop on Traffic Measurement and Analysis (TMA'13)*, 2013.
[15] B. Cohen and A. Norberg. Correcting for clock drift in uTP and LEDBAT. In *9th USENIX International Workshop on Peer-to-Peer Systems (IPTPS'10)*, 2010.
[16] M. Dhawan, J. Samuel, R. Teixeira, C. Kreibich, M. Allman, N. Weaver, and V. Paxson. Fathom: a browser-based network measurement platform. 2012.
[17] L. DiCioccio, R. Teixeira, M. Mayl, and C. Kreibich. Probe and Pray: Using UPnP for Home Network Measurements. In *PAM*, 2012.
[18] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.
[19] S. Gangam, J. Chandrashekar, I. Cunha, and J. Kurose. Estimating TCP latency approximately with passive measurements. 2013.
[20] J. Gettys and K. Nichols. Bufferbloat: Dark buffers in the internet. *Communications of the ACM*, 55(1):57–65, 2012.
[21] T. Isdal, M. Piatek, A. Krishnamurthy, and T. Anderson. Leveraging BitTorrent for End Host Measurements. In *PAM*. 2007.
[22] M. Izal, G. Urvoy-Keller, E. Biersack, P. Felber, A. Al Hamra, and L. Garces-Erice. Dissecting BitTorrent: Five Months in a Torrent Lifetime. In *PAM*. 2004.
[23] V. Jacobson et al. TCP Extensions for High Performance. IETF RFC 1323, 1992.
[24] R. Jain. A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks. *ACM SIGCOMM CCR*, 19(5):56–71, 1989.
[25] H. Jiang, Y. Wang, K. Lee, and I. Rhee. Tackling bufferbloat in 3G/4G networks. 2012.
[26] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson. Netalyzr: Illuminating the edge network. In *ACM Internet Measurement Conference (ACM IMC'10)*, 2010.
[27] G. Neglia, G. Reina, H. Zhang, D. Towsley, A. Venkataramani, and J. Danaher. Availability in BitTorrent systems. In *IEEE INFOCOM*, 2007.
[28] D. Rossi, C. Testa, and S. Valenti. Yes, we LEDBAT: Playing with the new BitTorrent congestion control algorithm. In *PAM*, 2010.
[29] S. Shalunov et al. Low Extra Delay Background Transport (LEDBAT). IETF draft, 2010.
[30] A. Venkataramani, R. Kokku, and M. Dahlin. TCP Nice: A mechanism for background transfers. In *USENIX OSDI*, 2002.