Passive Inference of Bufferbloat Root Cause

Andrea Araldo[1], Dario Rossi[2]

[1]LRI-Université Paris-Sud, Orsay, France – araldo@lri.fr
[2]Telecom ParisTech, Paris, France – dario.rossi@enst.fr

***Abstract*—In this work, we propose a methodology to gauge the extent of queuing delay (aka bufferbloat) in the Internet, based on purely passive measurement of bidirectional TCP traffic. Leveraging on Deep Packet Inspection (DPI) and behavioral classification, we next show a per-application breakdown of the queuing delay in ISP networks, that assists in binding the queueing delay to the performance perceived by the users of that application. Finally, we report preliminary results to further correlate the amount of queuing delay seen by each host with the set of active applications for that host during small time windows, to find the root cause of bufferbloat.**

I. Introduction

Despite the steady growth of link capacity, Internet performance may still be laggy in the early 2010. Already in 1996, a famous post [9] pointed out that delay, more than bandwidth, was an important metric for user perception. As confirmed by the recent resonance of the "bufferbloat" buzzword [16], this may still hold today. Shortly, excessive buffer delays (measured in seconds) are *possible* in today's Internet due to the combination of loss-based TCP congestion control coupled to excessive buffer sizes (e.g., in user AP and modem routers, end-host software stack and network interfaces) in front of slow access links.

While, through controlled testbed and experiments, it is clear that high latency can hamper user QoE of Web [5], [23], multi-media [17], [8] or even peer-2-peer [24] users, it is unclear how high is queuing latency in practice. Indeed, while it is known that queuing delays can potentially reach a few seconds [20] under load stress, and while these delays have been anedoctically observed [16], it however is unclear how *common* they are for end-users daily experience – which is precisely the goal of this paper.

Summarizing, our contributions are as follows: first, we propose a passive TCP queuing delay estimation methodology and make our open source implementation, based on Tstat[14], available to the community. Second, we quantify the typical bufferbloat seen by different applications, by leveraging on Tstat's Deep Packet Inspection (DPI) and behavioral classification capabilities. Third, we propose a methodology to find the root cause of bufferbloat, and report preliminary results that further correlate the amount of queuing delay seen by each host with the set of applications active on that host during small time windows.

II. Background

Delay measurement over the Internet are definitively not a new subject – indeed, over 20 years passed since seminal work such as [6]. Yet, despite the Internet steady evolution, performance problems resurface that are actually *worsened* by technology advances: indeed, Moore law not only increased the memory size, but also the amount of packets standing in modem buffers. As such, recent effort has focused on explicitly measuring, among other performance indicators, the latency and queuing delay experienced by end-users. With few exceptions [3], [15], [10], [11], most related effort [20], [12], [19], [23], [4], [5], [1], [21] employs active measurement techniques.

A. Active vs Passive measurement.

Our methodology relies on passive measurement that, due to its unobtrusiveness and realism of the user traffic, is the ideal candidate to answer our questions – i.e., what delay users observe in their daily activities? under what applications? Hence, due to lack of space, we invite the reader to a companion technical report [7] for a thorough overview and comparison of related work on active measurement. Here we limitedly observe that, though active methodologies are potentially simpler and more accurate, at the same time they exhibit some weak points of worth pointing out. First, many work measures latency under controlled load [20], [12], [19], [23], [4], [5], which tends to give *maximum* (rather than *typical*) bufferbloat. Second, periodic measures of latency have generally very coarse granularity (with the exception of [23], where samples are still spaced out by 6 seconds in the best case), so that bufferbloat can go unnoticed. Third, and most important, active techniques miss one crucial ingredient: the knowledge of the traffic that caused the bufferbloat. In other words, while [23] points out queuing delay to vary between 800ms and 10s depending on modem make and model, active methods are unable to gauge how often users actually see bufferbloats in excess of 1s, or to pinpoint the application that caused it – which are precisely our goals.

Another interesting tradeoff between active vs passive measurements concerns their representativeness, in terms of number of users and networks observed. Scale of active measurements can be as small as O(10)–O(10^3) users in O(1) networks (resp. [19] and [23]), growing up to O(10^5) users in O(10^3) networks [20], [4] (though it is worth stressing that the user base is possibly gathered over several months [20]). Passive methodologies allow to observe a larger number of users at any given time. At the same time, as observation of both forward and backward paths is necessary, they can hardly be applied in the network core (due to routing asymmetry, only about 2%[15] flows are bidirectional) so that, generally vantage points are sited at the network edge (e.g., near to a DSLAM as in this work) and represents about O(10^4) users in O(1) networks.

B. Passive measurement.

Some recent work tackled the problem of passive measurement of queuing delay [15], [3], [11], [10]. In our previous work [11], [10], we propose methodologies to infer *remote* host queues exploiting transport layer information available in packet headers, for both uTP [10] (the new protocol proposed by BitTorrent as TCP replacement for data swarming) and TCP [10] (using RFC1323 TimeStamp option[18]). Contrarily to [11], [10], in this work we focus on the *local* host queue (since we have full knowledge of *all* traffic on that host) and adopt a more general methodology, of which we outline some important differences. First, notice that while uTP timestamps allow to precisely gauge the remote queue (even in presence of cross-traffic toward unseen hosts) observations are limited in both space (to hosts that are running BitTorrent) and time (precisely when they run it). This constrains measurement campaign [10] on the one hand (a disadvantage shared with [4]), and possibly induces a biased view of the Internet bufferbloat on the other hand (since BitTorrent is a data-intensive application) – problems that this work instead avoids. Second, contrarily to [11], we avoid relying on timestamps carried in packet headers for TCP, increasing the reach of the methodology (despite growth of TCP TimeStamp option usage, this still account for modest 5%-30% at our vantage points).

Closer to our work is [3] that, using `bro`, employs a similar methodology to ours, relying on TCP data/ acknowledgement pairs, using trace timestamps as oppostie to TCP Timestamp option and takes care of rejecting RTT samples from retransmitted segments (though the methodology is not validated in a testbed, see Sec. III-A for potential issues). Since our dataset significanly differ from [3] (in terms of US vs EU location, FTTH vs ADSL access, duration, etc.), we cannot attempt a *direct* comparison of bufferbloat results – but still point out that, as in [3], [10] we find delays above 1 sec to be rare in practice. Additionally, while [3] equally counts all RTT samples (i.e., equally weighting *packets* of the same TCP burst, so that users transferring large volumes are over-represented) we give a more unbiased view (equally weighting each *second* of all hosts). Finally, a more important difference is that while [3] measures TCP queuing delay blindly across all applications, we instead give a fine-grained per-application view – binding queuing delay to user QoE and explaining its root cause.

Finally, [15] focuses on a memory-efficient bufferbloat measurement methodology, by keeping approximate TCP state in a probabilistic data structure (that can fit the cache of current MIPS and ARM processors used in home DSL gateways), at the price of a minimal accuracy loss (error is less than 10 ms in 99% of the cases, compared to `tcptrace` as a baseline). However, as the focus of [15] is on the relative accuracy of the methodology, it reports differences with respect to the baseline rather than absolute bufferbloat measurement. Our approach is instead complementary and, assuming a high performance dedicated measurement box (i.e., no memory constraint), implements a methodology to accurately gauge current Internet bufferbloat (incidentally, building over `tcptrace`, of which Tstat is an evolution).

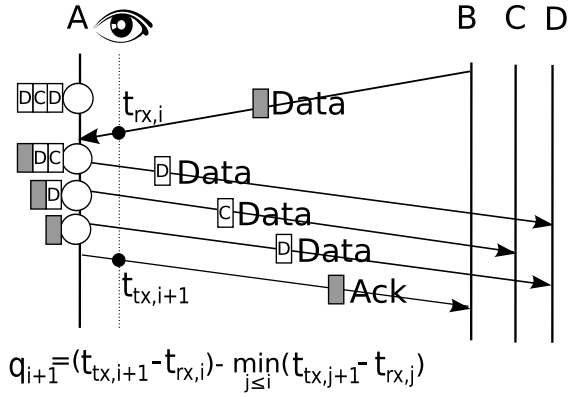


Fig. 1: Synopsis of our passive methodology

III. Queuing delay inference

A. Methodology

We infer queuing delay of local hosts simply as depicted in Fig. 1. For each acknowledgement packet reporting a valid RTT sample (i.e., no reordered, nor retransmitted as in [3]), we compute the difference between the time $t_{tx,i+1}$ the acknowledgement packet is observed and the time $t_{rx,i}$ the data packet corresponding to the sequence number was observed. Packets are timestamped at the measurement point via Endace DAG cards, so that timestamp is reliable. With respect to Fig. 1, assume the local host queue A contains at time $t_{rx,i}$ packets directed toward hosts B, C and D. Neglecting for the sake of simplicity delayed-acknowledgement timers (that are by the way small compared to the bufferbloat magnitudo reported in [20]), as soon as data is received at about $t_{rx,i}$ at A, the TCP receiver issues an acknowledgement that will be serviced after the already queued data segments. The monitor can then estimate the queuing delay q_{i+1} incurred by the $(i+1)$-th acknowledgement segment as the difference between the current samples $t_{tx,i+1} - t_{rx,i}$ and the minimum among the previously observed samples of that flow (that represents the propagation delay component and, as the monitor is close to the end-user, is by the way expected to be small).

B. Validation

At low level, `tcptrace` keeps sequence numbers in a circular data structure named `quad` (i.e., after the "quadrants" the structure is divided into, to speed-up lookup). In case the `quad` has a fixed size, it may happen that, if the number of outstanding segments grow larger than the `quad` size, then sequence number are overwritten – so that ack cannot be paired with data and RTT samples are lost. We show an occurrence of this problem in Fig. 2, where we configured two values of the quad size – one is fixed (to a purposely small value) and the other is variable and can grow arbitrarily large to avoid overwriting outstanding sequence numbers (notice that variable size is handled with linked lists, so that in this validation phase we take precisely the opposite direction to [15], as we do not want to compromise accuracy).

In a local testbed, we send bidirectional TCP traffic between LAN hosts, mimicking Fig. 1. Host B sends rate-limited

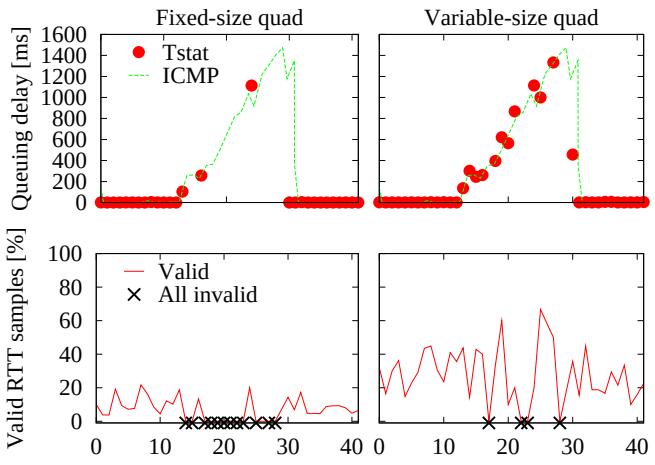
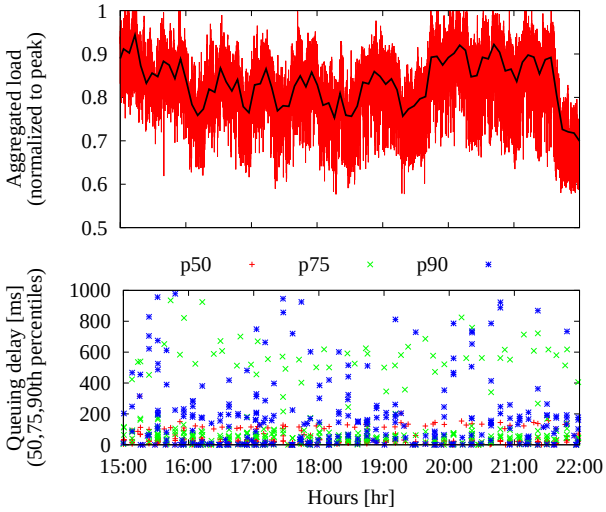


Fig. 2: Testbed validation of the methodology



Correlation coefficient	*Delay percentile*				
	p50	*p75*	*p90*	*p95*	*p99*
Load	0.007	-0.017	-0.081	-0.091	-0.092

Fig. 3: Time evolution of normalized load (top) and queuing delay percentiles (bottom). Correlation coefficient among load and queueing delay samples (1 minute window).

data (at application-layer) to A, whose acks are used to passively infer queuing delay samples at A (denoted Tstat in the picture). The uplink of host A is limited to 1Mbps with a token bucket shaper and, after about 10sec, we start a backlogged transfer between A and C, causing congestion to build up. To validate passive inference, we send ICMP echo requests to B at 1Hz, and compare average Tstat queuing delay during 1 sec long windows. To gather the impact of low-level Tstat settings, we record the packet trace and repeat the analysis with small and fixed (left) or variable (right) size `quad`. It can be seen that, in case the structure is under-dimensioned, sequence numbers are possibly overwritten when large amounts of packets are queued, so that possibly all samples during a 1 sec long window are lost. As large buffer sizes are common [13], [23], it follows that careful settings of monitoring tools are needed to avoid underestimating bufferbloat[1].

IV. Experimental results

We now report experimental results of the inference. As our focus on this paper is to build a solid methodology, rather than providing full-blown measurement campaign, for the time being we analyze offline traces gathered in a single ISP network during FP7 NapaWine project, when P2P was still fairly popular (we consider a 9 hr-long period starting at 14:00, gathered during 2009).

We argue that in order to give statistics that are useful from the user perspective, we need to batch[2] consecutive samples (e.g., belonging to the same TCP burst) into windows whose duration relates with the timescale typical of user dynamics. For the network under observation, we consider each internal IP as a single[3] host. For each host, we collect delay samples for each active flow, corresponding to the average queuing delay over short time windows of 1 second duration, as estimated by valid data-ack pairs of each flow. Overall, our processing gathers about 10^7 individual per-flow samples.

A. Time evolution

We first show the temporal evolution of the queuing delays statistics (namely, the 50th, 75th and 90th percentiles of the distribution over a short time window), along with the load on the link under study in Fig. 3 (where, due to NDA agreements, we normalize the istantaneous load over the peak load during the period), for the central portion of the trace. More precisely, for each time period, we construct an histogram of the 1-second queuing delay samples for all flows, and reset the histogram at each new period.

As it can be seen from the picture, aggregated load and queueing delay seem highly uncorrelated: to convince of this, we also report the correlation coefficient using 1 minute windows (different timescales, omitted for the sake of readability, yield very similar results).

Though it may seem counter-intuitive, this results is expected. Notice indeed that we are not saying that "delay and load are not correlated": rather, our results point out that "delay in the user home and load at the first hop IP router are not

[1]Notice that, at 1Mbps, 1500ms worth of queuing delay correspond to more than 100 queued packets, which exceeds the default `quad`=100 size in Tstat – hence, at higher capacities, even lower delay could be underestimated for lower `quad` sizes.

[2]In the specific case of uTP, we have already shown [10] that queuing delay statistics can be biased (precisely, queuing delay is underestimated) in case each packet is counted as a sample (as opposite to windows of equal duration).

[3]This is known to be simplistic as, due to the penetration of NAT devices, the same IP is shared by multiple hosts (50% of the cases [22]), that are possibly active at the same time (10% of the cases). Yet we point out that this simplification has no impact for our methodology, since these potentially multiple hosts share the same access bottleneck link

necessarily[4] correlated". This is due to the fact that congestion in the user home happens at the home gateway: thus, each user can see local congestion, that does not translate into congestion at the first aggregation router, that is properly dimensioned to operate below congestion.

B. Per-application breakdown

From the previous section, it follows that monitoring aggregated user load at a PoP is not informative so as to whether users are experiencing bufferbloat. Yet, even though we are actually observing bufferbloat for some flows (recall the 50th, 75th and 90th percentiles), this does not necessarily translate into bad user experience either, as we do not know how does the queueing delay statistics relate to user applications.

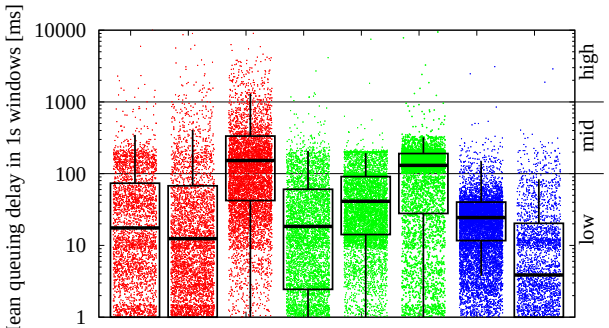
To have a more fine-grained view, we exploit Tstat Deep Packet Inspection (DPI) and behavioral classification capabilities (that we are unable to describe in full details here [14]). Each delay sample carries an application label and, though Tstat is capable of fine-grained classification of different applications, we cluster similar applications into few classes depending on the service they offer (namely, Mail, Web, Multimedia, P2P, SSH, VoIP, Chat and other uncategorized applications). To the best of our knowledge, this work is the first to report a detailed per-application view of the Internet queuing delay – that depends on the traffic mix and user behavior of each household.

We present our results in Fig. 4 where we depict a jittered density map of queuing delay samples (y-axis) for different application classes (x-axis), along with boxplots reporting the quartiles (and 5th, 95th percentiles). Applications are ordered, left to right, in increasing order of delay sensitivity. It can be seen that, for most applications the 75% of windows experience less than 100ms worth of queuing. The only exceptions are consituted by, rather unsurprisingly, P2P applications and, somehow more surprisingly, Chat applications, with *median* delays exceeding 100ms.

Before dwelving the root cause of the above observations, let us dig further its implication. Due to studies assessing the impact of delay on the QoE of several applications such as Web [5], [23], multimedia [17], [8] or P2P [24], [10] applications, we can easily map a QoS metric such the queuing delay, into an coarse indication of QoE for the end-user. Based on the above work, we set two thresholds at 100 ms and 1 second, such that:

- performance of interactive multimedia (e.g., VoIP, video-conference and live-streaming) or data-oriented applications (e.g., remote terminal or cloud editing of text documents) significantly degrades when the first threshold is crossed [17], [8];
- performance of mildly-interactive application (e.g., Web, chat, etc.) significantly degrades when the second threshold is crossed [5], [23];
- additionally, while bulk transfers (e.g., P2P, long TCP connections) are elastic in nature, it has been shown that also TCP performance degrades [16] in presence of excessive buffering (i.e., control becomes unstable due to absence/delay of feedback information) and furthermore queuing delay affect control plane of P2P applications [24], [10] – so that even these applications performance start to degrade when the second threshold is crossed.

[4] A corner case would be the one where congestion happened at the IP router, where in this case we would expect correlation to arise



	Oth	Mail	P2P	Web	Media	Chat	SSH	VoIP
low	89.7	93.2	58.4	91.9	86.8	45.7	98.6	97.8
mid	10.2	6.2	38.7	8.0	**13.2**	**54.1**	**1.4**	**2.2**
high	0.1	0.6	2.9	**0.1**	0.0	**0.1**	0.0	0.0

Fig. 4: Breakdown of queuing delay per application. Plots report jittered density maps and (5,25,50,75,95)-th percentiles. Table report the percentage of per-application samples falling into either of the three delay regions (boldface values highlight possible QoE degradation).

Additionally, Fig. 4 tabulates the percentage of 1 sec windows for each application that fall into either of the three delay regions, where we use boldface values to highlight possible QoE degradation. It follows that, in practice, bufferbloat impact on QoE appears to be modest. A limited 0.1% of Web and Chat sessions may be impacted by significant delay, and 2.2% (1.4%) of VoIP (remote terminal) sessions may be impacted by moderate delay. P2P clearly stand out, raising the odds to induce high delays (2.9%) followed by SMTP (0.6%), though with likely minor impact for the end-users.

V. Root cause analysis

A. Methodology

As we have full knowledge of the traffic generated and received by hosts, we can correlate the queuing delay samples with the traffic active on the hosts. As before, we limitedly consider the upstream traffic direction, and divide time in 1 second long slots. For all active flow of each hosts, we log the application label and its average queueing delay during that window. Notice that in Fig. 4 we independently consider all delay samples: in other words, the delay seen by packets of an application flow can be induced by another flows active on the same time in the same host (or household). In this section, we leverage standard data mining techniques to, if possible, pinpoint the root cause of the observed delay.

In our analysis, we face known, yet non trivial problems. First, techniques such as frequent itemset mining or rule-mining are known not to be scalable, and would be hard to apply to our full dataset. Second, these techniques are affected by class imbalance, that we need to cope with. At this stage, we make the problem tractable by performing

TABLE I: Rule inference with Apriori

High delay

Ref	Other	Mail	P2P	Web	Chat	Supp.	Conf.
(a)		✓				11%	99%
(b)			++			10%	96%
(c)				✓		10%	64%
(d)			++	✓		20%	98%
(e)	✓		++			28%	87%
(f)	✓			✓		23%	80%
(g)	++		++			14%	61%
(h)			++	++		19%	54%
(i)	✓		++	✓		13%	99%

Medium delay

Ref	Other	Mail	P2P	Web	Chat	Supp.	Conf.
(j)			✓			35%	76%
(k)				++		44%	53%
(l)			✓		✓	10%	85%
(m)	++		✓			11%	88%
(n)			✓	✓		19%	85%
(o)			✓	++		10%	77%
(p)	✓		✓			15%	76%
(q)	++			++		16%	72%
(r)				✓	✓	10%	51%

stratified sampling[5], initially including 8,000 random samples per-application (64,000 samples overall). Let us denote by (t,h,a,q) a queuing delay sample q gathered on host h during the time window t and corresponding to an application a. Evidently, we need to ensure that our population includes *all* other samples corresponding to applications that were active on h during t, and that concurred in generating bufferbloat. We therefore complement the initial population, to achieve an overall population of 107,825 samples. To guarantee the statistical relevance of our analysis, we further verify that statistics of the sampled population correspond to those reported in Fig. 4.

For each host h and time window t pair, we next compute the queuing delay average as seen by all applications during t. As rule-inference techniques are known to be largely ineffective on continuous variables, and in reason of the previous QoE considerations, we quantize queuing delay in a *low*, *medium* and *high* score.

We also aggregate all application labels, and denote with ✓ (or ++) applications that have generated respectively one (or more) flows during t. For the sake of illustration, consider 2 applications are active at host h during $t = 0$. One is a Chat application, consisting of a single flow, the other is a P2P application with 3 active flows during $t = 0$: we thus aggregate the application labels as (Chat✓, P2P++). Notice that this criterion again implements an aggressive quantization into classes: intuitively, we argue that there is more information in knowing coarsely whether there was a single or more flows, rather than distinguishing precisely the exact number of flows. Following this spirit, we additionally encode the previous example with purely binary indicators as in (Chat✓, P2P✓).

We yet have to perform a final step in order to facilitate the analysis of the root cause analysis. Namely, we group together windows, irrespectively of the host, having the same overall amount of active flows. This is necessary to simplify interpretation of the support of the inferred rules. Intuitively, as windows with x active flows for an host will not have the same frequency as windows with y concurrently active flows in our dataset, grouping window by the number of active flows allows to let overally infrequent (i.e., with small support) but still interesting (i.e., with high confidence) rules to emerge.

Once our dataset is build and properly quantized, we resort to standard algorithm for frequent item set mining and association rule learning, namely Apriori [2]. While reporting the full details of the algorithm would be out of scope, is worth pointing out that the main idea in Apriori is to find general and relevant rules by first identifying the most frequent individual items, and enlarge them in wider item sets as long as they occurr sufficiently often in the dataset.

B. Experimental results

Results of Apriori [2] are reported in Tab. I for both medium and high delays, where for each rule (essentially, a group of contemporary application labels inducing a given delay) we report confidence and support (percentage). Rules are reported, top to bottom, for incresing size (i.e., number of application label in the rule). For each rule size, rules are sorted, top to bottom, for decreasing confidence. Already from this small dataset, several interesting observation can be made.

First, notice that SMTP can cause bufferbloat: this is summarized by rule (a), which can be expected due to the use of persistent TCP connections, used to send multiple messages, each of which encapsulates possibly large e-mail bodies due to the use of MIME to encode attachments of various kind (e.g., pictures, music, archives, etc.). Notice that mail application does not otherwise frequently generate delay, confirming the intuition that small textual messages have no impact on the QoE of other applications.

Second, high delays are also due to (b) multiple concurrent P2P flows or (c) single persistent HTTP connections. Interestingly, the specular case in which (j) a P2P application has a single active flow, or (k) multiple HTTP connections are active in parallel, the delay generally remains bound to the 100 ms-1 sec range. The intuition is thus that users often browse the Web in parallel to P2P transfers. However, packets of short lived TCP Web connections often pile up behind bursts of TCP packets due to multiple P2P connections. An implicit confirmation of this intuition comes from the fact that BitTorrent recently[6] redesigned the data transfer, by introducing the delay-sensitive uTP protocol, aimed at bounding the queuing delay to no more

[5]Note that this spatial per-host sampling step is orthogonal to the per-flow temporal aggregation step where we coalesce delay samples coming from multiples packets into a single queuing delay statistics.

[6]Our dataset is before BitTorrent uTP became popular in late 2010.

than a configurable target parameter (set to 100 ms by default). Reason why a single Web connection should generate larger uplink queuing delay is instead less obvious, and need further investigation.

Third, various combinations of P2P, Web and uncategorized traffic (Other) jointly concurr in creating bufferbloat (we notice that uncategorized traffic possibly include P2P applications for which Tstat has no valid classification signature). We again notice that rule (d) combining a single Web and multiple P2P flows, has a higher confidence that rule (h). Moreover, as the support of (h) is included in the support of (d), it follows that including multiple Web connections weakens the rule significance – as already observed comparing (c) against (k), the delay seem to be inversely correlated with the number of Web connections.

Finally, we see that Chat sessions often happens in parallel with (l) P2P or (r) Web traffic – with the former especially cause of delay suffered by Chat application, in reason of the relative confidence of rules (a) and (j) with respect to (l). Hence, user behavior of Chatting in parallel to other applications is the cause of relative high frequency of medium delays (54% fall in the 100 ms-1 sec range) seen early in Fig. 4. From this observation, we can also conjecture that Chat users likely do not consider these levels of delay harmful for QoE. Indeed, since all other applications have lower delay statistics, this may follow from the fact that user naturally disactivate data-intensive background applications (e.g., P2P) and do not perform other activities (e.g., Web browsing) while delay-sensitive applications are ongoing (e.g., VoIP calls).

VI. Discussion

This paper propose a methodology to passively observe Internet bufferbloat. Though preliminary, this work already convey several useful insights (e.g., ranging from guidelines on settings of monitoring tools to avoid bufferbloat underestimation, to a per-application assessment of likely QoE impact, to rule inference, etc.) and especially allows to pinpoint its root cause.

While in this work we adopt a minimalistic approach, each delay sample carries additional information beyond the application label, concerning the amount of its activity during the window (i.e., number of packets, bytes and number of valid data-ack pairs). As part of our future work, we plan to leverage this information to refine the quality of our inference (e.g., samples could be weighted by the number of packets; the number of valid data-ack pair, raw or normalized over the number of packets in the window, could be used as an indication of the confidence of each sample; the istantaneous sending rate over the last few windows correlates with the queuing delay of the subsequent windows, possibly assisting root cause analysis; etc.).

More importantly, as part of our future work, we plan to deploy the modified version of Tstat on operational networks, in order to gather more statistically significant results from both spatial (over several ISPs and traffic mixes) and temporal perspectives (comparing the older dataset which we focus on in this paper with online monitoring results).

Acknowledgements

This work has been carried out during Andrea Araldo intership 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://internetcensus2012.bitbucket.org/.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *VLDB*, 1994.

[3] M. Allman. Comments on bufferbloat. *SIGCOMM Comput. Commun. Rev.*, 43(1), Jan 2012.

[4] 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*, 2011.

[5] Z. S. Bischof, J. S. Otto, and F. E. Bustamante. Up, down and around the stack: ISP characterization from network intensive applications. In *ACM SIGCOMM W-MUST*, 2012.

[6] J.-C. Bolot. End-to-end packet delay and loss behavior in the internet. *ACM SIGCOMM Computer Communication Review*, 23(4):289–298, 1993.

[7] P. Casoria, D. Rossi, J. Auge, M.-O. Buob, T. Friedman, and A. Pescape. Distributed active measurement of internet queueing delays. Technical report, http://www.enst.fr/~drossi/papers/casoria-tr.pdf, Telecom ParisTech, 2013.

[8] K.-T. Chen, C.-Y. Huang, P. Huang, and C.-L. Lei. Quantifying skype user satisfaction. 36(4):399–410, 2006.

[9] S. Cheshire. It's the latency, stupid! http://rescomp. stanford.edu/~cheshire/rants/Latency.html, 1996.

[10] 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)*, 2013.

[11] C. Chirichella, D. Rossi, C. Testa, T. Friedman, and A. Pescape. Remotely gauging upstream bufferbloat delays. In *PAM*, 2013.

[12] M. Dhawan, J. Samuel, R. Teixeira, C. Kreibich, M. Allman, N. Weaver, and V. Paxson. Fathom: a browser-based network measurement platform. In *ACM IMC*, 2012.

[13] L. DiCioccio, R. Teixeira, M. Mayl, and C. Kreibich. Probe and Pray: Using UPnP for Home Network Measurements. In *PAM*, 2012.

[14] A. Finamore, M. Mellia, M. Meo, M. Munafo, and D. Rossi. Experiences of internet traffic monitoring with tstat. *IEEE Network Magazine*, May 2011.

[15] S. Gangam, J. Chandrashekar, I. Cunha, and J. Kurose. Estimating TCP latency approximately with passive measurements. In *PAM*, 2013.

[16] J. Gettys and K. Nichols. Bufferbloat: Dark buffers in the internet. *Communications of the ACM*, 55(1):57–65, 2012.

[17] O. Holfeld, E. Pujol, F. Ciucu, A. Feldmann, and P. Barford. BufferBloat: how relevant? a QoE perspective on buffer sizing. Technical report, 2012.

[18] V. Jacobson, R. Braden, and D. Borman. TCP Extensions for High Performance. IETF RFC 1323, 1992.

[19] H. Jiang, Y. Wang, K. Lee, and I. Rhee. Tackling bufferbloat in 3G/4G networks. In *ACM IMC*, 2012.

[20] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson. Netalyzr: Illuminating the edge network. In *ACM IMC*, 2010.

[21] D. Leonard and D. Loguinov. Demystifying service discovery: implementing an internet-wide scanner. In *ACM IMC*, 2010.

[22] G. Maier, F. Schneider, and A. Feldmann. Nat usage in residential broadband networks. In *PAM*, 2011.

[23] S. Sundaresan, W. de Donato, N. Feamster, R. Teixeira, S. Crawford, and A. Pescapè. Broadband internet performance: a view from the gateway. In *ACM SIGCOMM*, 2011.

[24] C. Testa and D. Rossi. The impact of uTP on BitTorrent completion time. In *IEEE P2P*, 2011.