Actively monitoring bufferbloat, at a large-scale, from PlanetLab

Pellegrino Casoria[1,3], Dario Rossi[1], Jordan Auge[2], Marc-Olivier Buob[2], Timur Friedman[2] and Antonio Pescape[3]
[1] Telecom ParisTech, first.last@enst.fr
[2] UPMC Sorbonne Universites first.last@lip6.fr
[3] Universitá di Napoli Federico II first.last@unina.it

***Abstract*—Hosts in the current Internet are seldom experiencing large queueing delays, dubbed bufferbloat, that are self-inflicted by users due to TCP dynamics filling their access link buffers. While it is known that maximum bufferbloat possibly grows up to a few seconds, it is less understood how often this occur in practice. To answer this issue, we design a system based on distributed agents running on PlanetLab that aims at quantifying its extent. Specifically, we focus on continously probing at high frequency a fairly large number of hosts, to build a spatial and temporal map of Internet bufferbloat. In this paper, we report on the system architecture, its validation, and on preliminary results gathered through a measurement campaign we are currently running.**

I. Introduction

Internet delay and delay variation have been mesured over the years a lot of times, involving an almost countless number of active experiments that involve traceroute or ICMP echo request and reply packet pairs – some of which also made it to the Museum of Modern Art (MOMA) in New York [1]. Yet, in reason of the recent revival of the persistently full buffer problem, for which the bufferbloat buzzword was coined [16], late effort has focused on explicitly measuring, among other performance indicators, also the latency and queuing delay experienced by end-users.

With few exceptions [5], [11], [12], [15], most related effort [2]–[4], [8], [9], [13], [17]–[19], [23] employs active measurement techniques, on which we focus in the following (see a companion technical report [6] for passive measurement). Given the abudance of related work, it may seem at first sight redundant to focus on bufferbloat measurement via active measurements. Yet, this work nicely fit in a gap of the design space explored by the research community.

Summarizing our contributions, we first compare current efforts in quantifying bufferbloat, showing advantages and limits of each approach. Second, we present an architecture for large scale bufferbloat measurement, based on distributed agents running over the PlanetLab/TopHat infrastructure, that we plan to release as open source software. At the heart of our distributed agents, lays an efficient pinger module, able to contact about 15,000 (45,000) hosts every second (5 seconds). Finally, we present initial results of a measurement campaign we are currently running.

II. Design space taxonomy

Generally speaking, we can categorize work based on their methodology (e.g., active vs passive), network segment (e.g., core vs access), access type (e.g., fixed vs mobile), breadth of observation (e.g., focused vs large), maturity of the monitoring (e.g., continuous vs punctual effort), etc. Throghout this section, we compare our work to related effort with the help of the taxonomy presented in Tab. I.

A. Measurement systems

The best known examples of large-scale, continuosuly running, active monitoring systems that measure, among other observables, RTT delay are RIPE Archipelago [2] (that, from the early days of King/Skitter/Scamper, grew to about 1000 monitors in 2013) and Dimes [3] (with over 600 active agents). As deploying such system for a narrow goal as ours (bufferbloat measurement) would be cumbersome, the primary goal of these distributed measurement platforms is Internet-scale topology inference – yet they do not readily perform the kind of measurement we envision. For instance, on average each of the 600 Dimes agents perform 10,000 weekly traceroute measurement [22], or about 1 measurement every 3 minutes – while we aim at fine-grained temporal measurement on the order of 1 Hz, about two orders of magnitude.

B. End-user applications

Arguing that the best monitoring point to assess user quality of experience is as close as possible to the users, work such as [8], [18] leverages end-user application to perform active monitoring. Specifically, Netalyzr [18] is a Web browser based application (recently reimplemented as a browser plugin [13]) for troubleshooting user access network, that performed the first large scale characterization of bufferbloat in user home networks. Yet, Netalyzr requires user collaboration and measures the bufferbloat with an active methodology that saturates the up/downlink, so that *maximum* queuing delays are exposed, as opposite to the typical queuing suffered by users during their normal activities.

Authors in [8] instead exploit BitTorrent as a service for crowdsourcing ISP characterization from the edge of the network, via a plugin called Dasu. Owing to the plugin popularity, [8] reaches a quite large population of 500K peers in 3K networks. While [8] mainly focuses on achievable data rates for BitTorrent peers, [9] also considers latency and validates the significance of results gathered on BitTorrent peers against [23]. In more details, [9] uses traceroute to measure last mile latency, but limit to 100 the number of traceroute per peer. Hence, statistics may not be faithful of the full spectrum of user daily/weekly activities (additionally,

TABLE I: Comparison of related work based on active measurement, limited to their delay measurement aspects

[ref]	Access	From	To	Campaign	Scale	Comments/Bufferbloat
[18]	Fixed	Web browser	Custom servers	Ongoing	130K users	Controlled cross traffic
[8], [9]	Fixed	P2P	P2P	Ongoing	530K users	Last mile, e2e (100 probes)
[17]	Mobile	Smartphone	Custom servers	One shot	$<$10 devices	Controlled cross traffic
[23]	Fixed	Home gateway	Mlab servers	Ongoing	2K gateways	Last mile, e2e (unfrequent)
[19]	-	Server(s)	IPv4 hosts	One shot	10G target	139M responses (1 day)
[4]	Fixed	Botnet	IPv4 hosts	One shot	10G target	420M responses ($>$ 1 days)
[14]	-	1 Server	IPv4 hosts	One shot	10G target	45min for full Internet scan
[2]	-	70 agents	IPv4/6 hosts	Ongoing	70 agents	RTT distribution
[3]	-	600 agents	IPv4 hosts	Ongoing	600 agents	10,000 weekly traceroute/agent

experiments in [8] were scheduled to avoid interacting with user traffic).

C. End-user devices

Another class of work instead exploits custom end-user devices, such as for instance home gateway of fixed cable, DSL and FTTH access lines [23] or mobile terminals of cellular 3G/4G networks [17]. In more details, [17] exposes cellular bufferbloat by instrumenting devices to perform RTT measurement. Overall, measurements in [17] involve 9 devices from 3 locations from 4 ISPs, using 3 ICMP echo reply server. As they generate controlled cross-traffic with TCP, their methodology is similar to [18], as it exposes the maximum bufferbloat.

Authors in [23] exploit the SamKnows (and BISmark) user base consisting of 1914 (14) home gateways deployed in the households of different ISP networks, to perform bandwidth and latency measurement. For what concerns latency, the measurement frequency is however low compared to hours, as they check (last-mile and end-to-end) latency by sending 5 (12) ICMP packets per hour. Additionally SamKnows sends 600 UDP pkts/hr toward MLab hosts, resulting in one end-to-end sample every 6 seconds. While this frequency is quite fine grained, especially with respect to other related work, we point out that bufferbloat may go unnoticed for uploads lasting less than two measurement rounds, if the first sample falls during a initial Powerboost phase (typically 5MB worth of data [7]) where the uplink rate is significantly higher. Finally, also SamKnows (and Bismark) also measures every 30min (2hr) latency under up/downlink load, with an approach similar to [18] that tends to expose maximum bufferbloat (between 800ms and 10s) rather than the typical one.

D. Internet scan

Another class of closely related work is represented by large-scale Internet census efforts exploiting common servers [19] or even Botnets [4]. However, scanners aim at maximize the spatial reach [4], or jointly maximize the reach while scheduling probes so as to minimize intrusiveness [14], [19]. However, in order to gather a full Internet snapshot of the smallest possible duration (down to one day [19] or hour [4] only few probes are exchanged with any host [4], [14], [19] (e.g., one ICMP packet to measure reachability), which limits the ability to monitor delay variations.

An interesting tradeoffs emerge from comparison of the Carna BotNet [4] vs zmap [14], that both peform full IPv4-Internet scanning: while the first employs a large number of very slow devices (i.e., over 500k compromised home gateways with low CPU and limited bandwidth), the latter instead employs a single one (i.e., high-end server in academic environment with Gbps connection). Our effort precisely sits in between these two extremes.

E. Our contribution

In this paper, we focus on a more noarrow goal (i.e., bufferbloat measurement), developing an efficient system for large-scale high-frequency scanning, customized to periodically report very detailed per-host statistics (e.g., percentiles). As individual probes are capable of scanning about 10K hosts in a second, it follows that using 100 PlanetLab nodes we could in principle follow about 1 million hosts every second or, trading space for time, cover the whole Internet in about one hour.

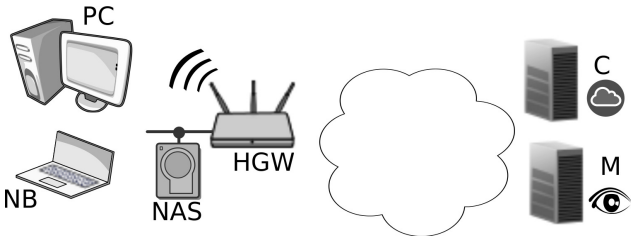
Due to architectural similarities with scanners [4], [19] and systems based on distributed agents [2], [3], our approach allows to achieve spatial scales larger than [17], [23] (already in this paper) and [8], [9] (prospectively). Additionally, our efficient implementation allows much higher scan frequency than [2], [3], [8], [9], [17], [23], where the frequency of background latency measurement is typically too sparse to offer an adequate bufferbloat characterization from the user perspective.

Finally, in terms of the delay statistics, we avoid to measure (maximum) latency under controlled load as in [17], [18], [23], and rather gather the (typical) delay by continuous host measurement, hence sampling the user load during their normal activities

III. Bufferbloat scanner

A. Architecture

Building over TopHat [10], we design a distributed architecture for Internet bufferbloat scanning. We target at a simple yet flexible design that can e.g., perform high-frequency sampling of large target sets hosts for prolonged durations (e.g., sending 1 packet/second to 10^6 targets for a week), or Internet-scale low-frequency sampling (e.g., a periodic scan of a significant fraction of Internet hosts every hour). We divide measurement periods (of 5 minutes by default), at



Case	NIC	IP	Cross traffic
A	Eth	public	PC to C through HGW (Cloud)
B	Eth	private	PC to C through HGW (Cloud)
C	WiFi	private	PC to C through HGW (Cloud)
D	WiFi	private	NB to NAS over WiFi (local)

Fig. 1: Testbed scenario

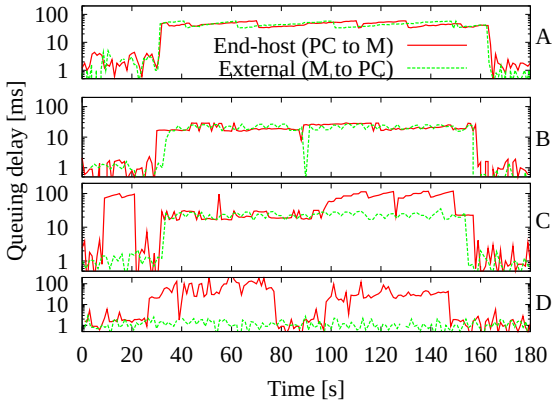


Fig. 2: Bufferbloat scanner: testbed validation

the beginning of which each measurement server ask for instructions (essentially, a list of destinations/subnets and the sampling frequency, 1 Hz by default).

During the measurement period, hosts are probed, in parallel, by each server. Upon reception of any new samples from an host, beside the usual minimum, maximum, average and standard deviation, we also compute per-host percentiles of the queuing delay distribution. At the end of each measurement period, per-hosts stastistics (and possibly the raw log, if debug is enabled) are collected through the TopHat facility for further post-processing.

For each target, we gauge the queuing delay via ICMP measurements as $q_i = RTT_i - \min_{j \leq i} RTT_j$. At first sight, this may be biased due to recent work [21] assessing the unsuitability of delay measurement via ICMP: indeed, ICMP may provide a biased RTT view with respect to application flows, since flows may follow different paths depending on their flow identifier. However, we have to stress that we are not using ICMP to provide *absolute* RTT measurement – that are indeed biased, as from the RTT difference across multiple flows, one could wrongly attribute to queuing delay variations, on the order of tens of milliseconds, that are instead due to differences of data plane paths. Instead, our methodology is unbiased, as we use *relative* RTT measurement with respect to the RTT baseline gathered by flows having the same flow-ID – that are unbiased for any flows, since all packets of a flow follow the same path, and are thus representative of the queuing delay.

Our methodology couples the state of the queues, as it measures the sum of the forward and backward queuing delays: yet, by ensuring that queuing does not happens at the measurement servers, we can however correctly infer the remote queuing delay. Notice that (i) this is easily done when servers are under control (ii) the decision that a measurement round is affected by server-side queuing (and should thus be disregarded) can be taken locally (i.e., whenever a server observes a sudden increase of several queuing delay samples of spatially unrelated hosts probed by the same server).

We notice that the generally measurement are *initiated* by the end-host [8], [9], [17], [18] (SamKnows/BISmark [23] slightly differ in that measurement starts from the HGW). In our case, measurement are instead *targeting* the end-hosts: this is common in large-scale census studies [4], [19] (of which we inherit the scalability property) but has not been explored so far, to the best of our knowledge, for bufferbloat measurements.

B. Testbed validation

To build and validate our methodology, we first carefully inspect a number of cases, that we describe with the help of Fig. 1. We consider a measurement server M, sending ICMP echo request probes to a non cooperative host H (as the is on our control, we verify queuing delay at M to be zero). We send ICMP echo requests both from the end-hosts PC toward the measurement server M (End-host), as well as from M to PC (External).

We consider the typical household scenario where the host can be any device (e.g., a fixed PC, a mobile notebook NB, a shared NAS disk) connected to the Internet through a home-gateway HGW, that may also serves as WiFi access point. For the sake of simplicity, let fix H=PC in the household scenario. We then consider several cases, denoted with an upper case letter A-D, where:

- the PC host has either a public or a private IP address (in the latter case, HGW works as a NAT box);
- the PC host is connected to HGW with a different NIC (Ethernet vs WiFi);
- the cross traffic is either generated by the PC host connected to a server C, or cross traffic is generated by other hosts in the same household, such as a remote backup between a notebook NB and a storage device NAS (emulating cloud vs local traffic).

Measurement results of this small scale Internet testbed are shown in Fig. 2. It can be seen that if the host has a public IP address (A), ICMP measurements are in perfect agreement, so that external measurement are as accurate as end-host techniques. Knowing that [20] about 90% of DSL lines use NAT gateways (with 50% having multiple hosts, that are active at the same time in about 10% of the cases), this means that the methodology is accurate for at least 10% of the non-NATted hosts.

	T	1s	2s	5s	10s
nPing	S	10k	15k	20k	30k
TopHat	S	15k	25k	45k	75k

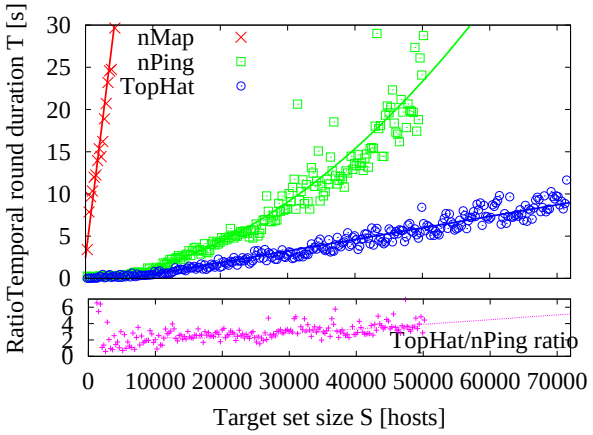


Fig. 3: Bufferbloat scanner: tool scalability

It is thus more interesting to observe behavior of NATted hosts. More precisely, in case of a NATted host connected through wired (B) or wireless (C) link to its HGW, we see that both methodologies generally agree so as to *when* there is queueing delay. At the same time, two remarks are worth pointing out. First, the ICMP echo requests sent by the external monitor now terminates at the gateway: in case there are outstanding TCP segments, the ICMP echo reply generally get serviced just after one TCP segment. This implies that the queuing sampling process will infer that *some* amount of queuing is present, which is furthermore confirmed by all consecutive samples, despite it cannot tell for sure the *exact* queing delay amount (see discussion in Sec. V). Additionally, when the host is connected through WiFi it may happen that fading, interference, sudden bitrate changes further add complexity to the picture (see discrepancies in C).

Finally, (D) considers an increasingly common scenario in current households, determined by the appearance of "intelligent" devices requiring a fair amount of background synchronization. In this case, both PC and NB are connected through the WiFi access points, while a storage device NAT is wired to (or colocated with) the HGW. Whenever NB access content stored in NAT through WiFi, the performance of PC will be affected due to contention, and possibly bufferbloat, at the WiFi AP queue. However, since the traffic is entirely local, no packet is outstanding into the outgoing modem queue: as a results, an external process will see no delay to HGW, despite the PC host is bufferbloated. Is worth pointing out that in this case, while purely end-host solution [9], [18] will correctly detect bufferbloat, queuing delay will go undetected by both external server and gateway-based solutions [23] (unless the latter also probes the local household network).

Summarizing, we find that external-server solution appears to be an interesting alternative worth exploring to current end-host or gateway based solutions: indeed, such techniques inherits from Internet scan methodologies the appeal for potentially large-scale study. At the same time, large scale possibly requires some accuracy compromise: while for a minority of non-NATted hosts it is possible to accurately quantify queuing delay, in the case of NATted hosts it should still be possible to provide more coarse indicators (e.g., a binary bufferbloat flag). Finally, we also point out that bufferbloat due to congestion in the household can also be unnoticed under other currently deployed techniques.

C. Scalability

At the core of our scanner, lay an efficient tool to ping a large amount of hosts with the least possible resources. Our pinger is implemented with two threads employing one raw socket: one thread sends ICMP echo request packets, the other listen for incoming ICMP echo reply (or error) packets. Once echo reply packets are received, per-host statistics are updated online. In case error packets are received, their feedback is used to dynamically build *graylist* for unactive hosts (that will be probed again in the next period) and *blacklists* for administratively prohibited hosts (that will not be probed again).

To give a rough idea of the tool performance, and speculate on the achievable measurement scale, we report in Fig. 3 the time needed (T, y-axis) by our tool to contact a growing number of hosts (S, x-axis). Performance are compared to nmap and nPing, a tool for large-scale ping scanning based on the Nmap Scripting Engine. Additionally, the picture tabulates the spatial set size S that can be achieved in rounds having fixed temporal duration $T \in [1-10]$ sec.

We already gather that our ICMP module introduces non-negligible gains with respect to nPing. More precisely, the gain is lower for smaller round durations and grows with the target set size. Shown in the picture, fitting of the experimental data exhibit a linear dependency between R and S for TopHat. This instead does not hold for nPing, where for $S{>}50K$ target sizes, the duration of a cycle exploses. Hence, the gains for $S{>}50K$ shown in the lower plot of Fig. 2 are conservative in that they are based on real data gathered for $S{<}50K$.

With back of the envelope calculation, we see that the core ICMP scales to about 10^6 probes per second considering a set of 100-500 OneLab/PlanetLab machines targeting 10K-2K hosts each – which fits our initial goals of high-frequency (1 packet/sec to 10^6 hosts), or Internet-scale (1 packet/hour to $3.6 \cdot 10^9$ hosts) scans.

IV. Measurement campaign

To validate the tool on the wild, we report results on a preliminary measurement campaign. We focus on moderate number of hosts O(10^4) on the same ISPs, that we continuously probe at 0.5 Hz frequency from 2 separate PlanetLab nodes for a period of about 8 continuous hours. According to Fig. 2, TopHat is able to handle about 25K ICMP probes per second, and we conservatively set a target set size of 75% this bound (i.e., 18,750 hosts, about one order of magnitude more than [23]). Overall, we receive replies to 47% of our sent packets, for a total of O(10^8) valid samples – using only two PlanetLab servers, we already achieve a quite significant scale in terms of spatial reach and temporal frequency.

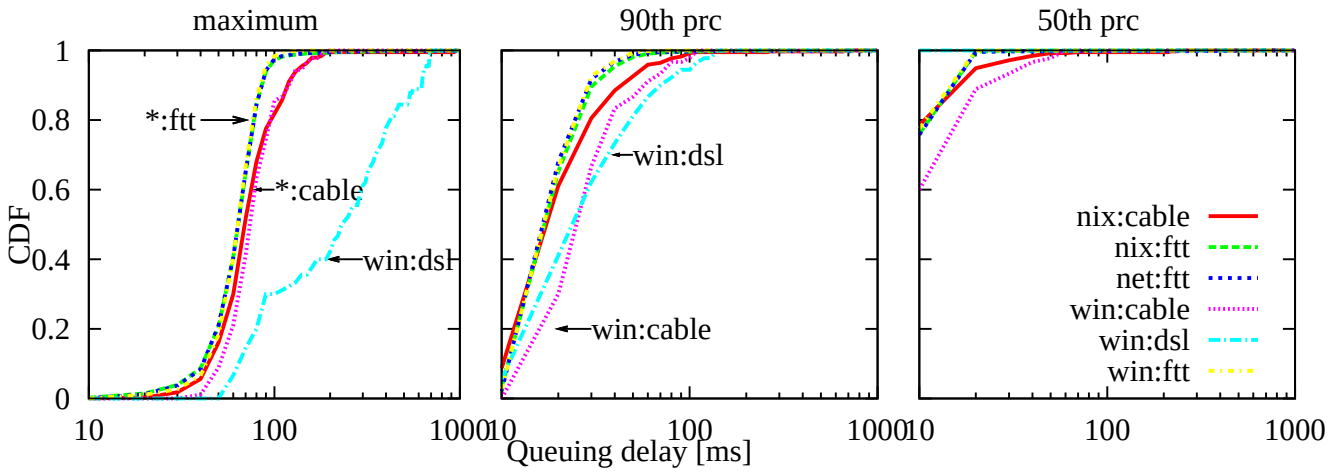


Fig. 4: Validation of the Internet measurement campaign: CDF of maximum (left), 90-th percentile (center) and median (right) delay, for hosts with reliable reverse DNS and OS fingerprint information.

A. *Internet validation*

Yet, we still need to ensure that the methodology can yield to some significant inights in terms of the gathered statistics in the wild Internet. For validation purposes, we therfore infer (i) the access type (AT) of our target hosts by issuing reverse DNS queries, as well as (ii) the remote operating system (OS) through `nmap` fingerprinting. As for the access type, the ISP we are targeting offers DSL, FTTH and cable access: similarly to [11], we expect the breakdown of queuing delay along AT to yield an intuitive validation of the observed statistics. As for the OS, we do not argue queuing delay performance to be (strongly) tied to the OS (despite we still expect difference to arise due to, e.g., different default TCP congestion control flavors across OSs). Rather, we use the OS information as an indication so as to whether we are in case A, B, C of the testbed of Fig. 1. Specifically, we argue that in case the remote OS is reliably found to be a Windows OS, then we can very likely rule out the case of NATted access: more likely, we hit a private end-host or a server of a small-size business professional[1] Overall, we manage to infer both AT and OS information for 2546 hosts: as for the OS breakdown, the majority of hosts are reported to be some Linux variants (most of which are likely NAT devices), followed by known home gateway boxes (denoted with *net*), and 12 are Windows servers; as for the AT, the majority of hosts has fiber access (denoted with *ftt*), followed by cable and 6 DSL lines. Hence, this highly unbalanced validation subset does not allow to report any statistically meaningful per-AT or per-OS conclusions – but nevertheless allows to validate our methodology.

As described in Sec. III, we collect per-host percentiles during 5 minutes windows: Fig. 4 reports the Cumulative Distribution Function (CDF) of a few per-host queuing delay statistics, gathered over all hosts and measurement rounds (in other words, each host yields a sample for each CDF during a different 5 min round). In more details, top plot reports the maximum queuing delay CDF: as expected, from the picture clearly emerges that (a) fiber access suffers the lowest delays irrespectively from the OSs, (b) cable delays are only slightly higher, whereas (c) DSL end-hosts may seldom suffer from delays close to 1 sec. We further report the 50th (bottom) and 90th (middle) percentiles CDF. Notice further that (d) from practical purposes, the 90th percentile is lower than 100 ms under any combination of OS and AT – including end-hosts behind DSL. Moreover, since the *win:cable* and *win:dsl* lines now clearly separate from the others, we infer that the methodology needs to be refined as it likely underestimates bufferbloat delay for NATted hosts – although observation (d) suggests bufferbloat not to be relevant.

[1]In case the Windows OS runs an HTTP server, we empirically verify our intuition by manually browsing to the Webpage for a handful of cases. Residential users could instead be classified via Spamhaus/PBL.

B. *Consistency*

As a further validation, we contrast the queuing delay statistics relative to the the overall set of 18,750 hosts, probed during the same timeframe by two independent PlanetLab nodes. In principle, we expect bufferbloat measurement to be the same irrespectively of the measurement server. Yet, we need to rule out the fact that the mixture of applications running on other slices of the same PlanetLab nodes negatively affect the measurement.

To conduct rigorous analysis, we not only compare scalar queuing delay statistics (i.e., mean and variance, over all hosts and rounds, of the queuing delay percentiles, tabulated in Fig. 5) but also report CDF of the actual probe rate (in million packets per round) and effective inter-probe time (in seconds). Results confirm that, despite individual agents may run on PlanetLab nodes running a mix of different applications, (i) queuing delay statistics agree on a 2 ms accuracy and (ii) TopHat agents running on PlanetLab are able to send 1.5-2 millions ICMP probes per 5 minutes round, with an inter-probe gap for the same host of 2-2.5 seconds.

Queuing delay Percentiles	Srv_A		Srv_B	
	Mean	Var	Mean	Var
50th	19.3	2.8	17.5	2.9
75th	27.1	8.0	25.1	8.2
90th	38.4	17.9	36.2	17.6

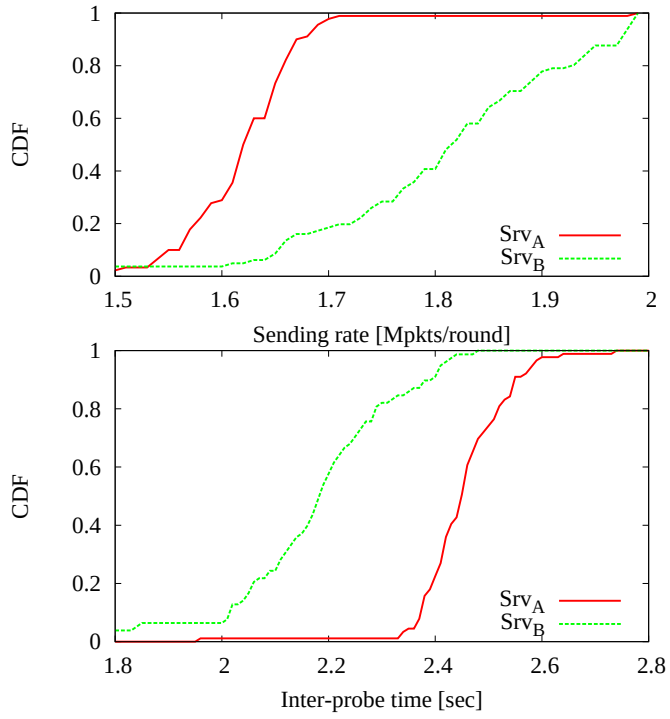


Fig. 5: Validation of the measurement campaign: queuing delay statistics and performance indicators for two different servers (same target set, same time).

V. Summary

This work fills a gap between the plethora of available active methodologies for active bufferbloat measurement. We develop an efficient tool that, trading crawling space vs frequency, is able to continuously estimate queuing delay of Internet hosts at fairly large scales.

Our ongoing work follows two main lines. On the one hand, we are currently performing a larger scale measurement campaign. At the same time, we have shown the tool to be accurate when hosts have public IPs, whereas it may underestimate the queuing delay of NATted hosts. Our second direction is thus to coarsely, but reliably, estimate queuing delay indicators in such cases. We envision an heuristics approach based on (i) observation of multiple consecutive samples with steady queuing delay, to gather a binary indication that one (or more) packets are in the uplink buffer with (ii) knowledge about the NAT device, such as make and model (e.g., via active device fingerprint or UPnP measurements) from which the buffer size could be extrapolated. Knowledge of (i) allows to know that queuing happens, while (ii) bounds queuing delay in a range specific to each host. While this approach does not allow to quantify the precise amount of queuing delay of each hosts, it should however allow to gauge, at large scale, whether end-user queuing happens in nowadays Internet, and how frequently.

Acknowledgements

This work has been carried out during Pellegrino Casoria 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://www.sdsc.edu/News%20Items/PR022008_moma.html.

[2] http://www.caida.org/projects/ark/.

[3] http://www.netdimes.org/new/.

[4] http://internetcensus2012.bitbucket.org/.

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

[6] A. Araldo and D. Rossi. Passive inference of bufferbloat root cause. Technical report, http://www.enst.fr/~drossi/papers/araldo-tr.pdf, Telecom ParisTech, 2013.

[7] S. Bauer, D. Clark, and W. Lehr. Powerboost. In *ACM SIGCOMM Workshop on Home networks*. ACM, 2011.

[8] Z. Bischof, J. Otto, M. Sánchez, J. Rula, D. Choffnes, and F. Bustamante. Crowdsourcing ISP characterization to the network edge. In *ACM SIGCOMM Workshop on Measurements Up the STack (W-MUST'11)*, 2011.

[9] 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 Workshop on Measurements Up the STack (W-MUST'12)*, 2012.

[10] T. Bourgeau, J. Augé, and T. Friedman. Tophat: supporting experiments through measurement infrastructure federation. In *TridentCom*, 2010.

[11] 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.

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

[13] 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.

[14] Z. Durumeric, E. Wustrow, and J. A. Halderman. Zmap: Fast internet-wide scanning and its security applications. In *Proceedings of the 22nd USENIX Security Symposium*, 2013.

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

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

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

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

[21] C. Pelsser, L. Cittadini, S. Vissicchio, and R. Bush. From paris to tokyo: on the suitability of ping to measure latency. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 427–432. ACM, 2013.

[22] Y. Shavitt and U. Weinsberg. Quantifying the importance of vantage points distribution in internet topology measurements. In *INFOCOM 2009*, 2009.

[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.