On the Interoperability of Bufferbloat Solutions

Thiago Cardozo, Alex Borges Vieira, Artur Ziviani, Ana Paula Couto da Silva

***Abstract*— Bufferbloat phenomenon is related to the excessive packet queuing in over-sized buffers inside the network that may lead to network performance degradation. In this context, we observe a lack of experimental results considering the practical aspects of off-the-shelf network devices. In this paper, we present a systematic analysis of the bufferbloat phenomenon considering the microscopic view of the buffer architecture of typical network devices. Moreover, we evaluate the most common fighting bufferbloat solutions recently proposed in the literature, such as BQL, CoDel, FQ-CoDel, and PIE. In fact, we have only observed bufferbloat effects under particular configurations. Second, all approaches we have tested are efficient against bufferbloat, although presenting different performances. Finally, mechanism we have evaluated can interoperate. We only notice a bad performance under a misconfigured BQL; actually results suggest a misconfigured BQL is worse than no BQL at all.**

***Keywords*— Bufferbloat, AQM, CoDel, BQL, FQ-CoDel, PIE**

I. Introduction

An increase on Internet end-to-end latency has been observed in recent years [17]. The bufferbloat phenomenon is usually pointed out as one of the possible causes of this increasing latency. In fact, the past couple of years has witnessed many works in the networking community about bufferbloat [19], [14], [5], [11], [1]. In short, bufferbloat occurs as a consequence of excessively large buffers at network devices, specially routers. These large buffers create a significant delay, which persists for long periods, thus degrading the network performance [7].

Queues in network routers absorb traffic bursts avoiding data loss, but buffer sizing is a recurrent challenge in the network area [2]. Excessively large buffers may defeat the fundamental congestion avoidance algorithm of TCP (Transmission Control Protocol), the most common Internet transport protocol. More precisely, TCP implicitly detects congestion when noticing packet loss [12]. As buffers at routers are over-sized, the queue occupation increases without the TCP connections reducing their transfer rates. Such a behavior leads to large end-to-end delays caused by the extra overload in the network.

The buffer sizing problem, and its consequences to excessive delay, has been recognized as early as 1985 [18]. Currently, there is a number of solutions addressing the bufferbloat issue [19], [9]. Most of these solutions are new Active Queue Management (AQM) policies, which enable an Internet router to inform the senders that they need to reduce the sending rate before the buffer becomes full, allowing routers to maintain relatively small queues and thus avoiding bufferbloat [13].

Thiago Cardozo Computer Science Department, UFF, Brazil. Alex Borges Vieira Computer Science Department, UFJF, Brazil. Artur Ziviani Computer Science Department, LNCC, Brazil. Ana Paula Couto da Silva Computer Science Department, UFMG. Brazil. E-mails: boubee.thiago@gmail.com,alex.borges@ufjf.edu.br, ziviani@lncc.br, ana.coutosilva@dcc.ufmg.br. This work was partially supported by CNPq, FAPEMIG and FAPERJ.

In this paper, we present a systematic evaluation of different bufferbloat solutions. The solutions we consider are BQL (Byte Queue Limits) [9], CoDel [19], PIE [20], and FQ-Codel [10], the most common bufferbloat fighting approaches. While BQL limits TCP transfers, the remainder are new AQM approaches. We also evaluate the interoperability of these solutions and possible mutual effects. All those solutions are available on Linux-based devices, running Linux kernels from version 3.10 onwards.

We have assessed the effects of these proposed solutions on the bufferbloat phenomenon, considering a typical network device architecture. We have evaluated key network metrics, such as the end-to-end latency and throughput while varying the size of the involved network buffers. Our experiments have been conducted in a controlled testbed, using commercial hardware and open-source software.

Our experimental results show that BQL, CoDel, PIE, and FQ-CoDel are effective in fighting the bufferbloat phenomenon. When we disable these solutions we experience an excessive latency, characterizing the bufferbloat. For example, comparing a typical bufferbloat scenario with the system's standard configuration, we observe an increase of up to 585% in the end-to-end delay. In contrast, using BQL or CoDel, no significant impact in the analyzed metrics are observed. In this case, the resulting end-to-end delay is around 0.5% larger than the latency under the same system's conditions without the analyzed solutions.

Moreover, we also present an analysis considering the coexistence of the solutions available in recent kernels of Linux operating systems that were incorporated to mitigate the bufferbloat effects. In this case, end-to-end delay in a typical bufferbloat scenario can be negligible. However, a misconfiguration of BQL can reduce CoDel efficiency and, in this case, end-to-end latency is 17K% higher when compared with a scenario with an auto-tuned BQL.

II. Bufferbloat

Queues in packet-switched networks are used to absorb bursts on packet arrival rates that can vary significantly in short periods of time [19]. Nowadays, due to the low cost of memory in routers, it is usual to find large amounts of buffering capacity in network devices, even in the simplest versions. The manufacturer's premise is that a large buffering capacity avoids packet loss, improving the network quality of service to the end user.

However, the scenario where network devices present large amounts of buffering introduces the problem recently known as the bufferbloat phenomenon [7], [19]. The excessive packet buffering results in large end-to-end latency as well as throughput degradation. In fact, large buffers (and queues) in routers is not a recent problem [18]. In a network with infinite queues

and packets with limited lifetime, the throughput tends to zero. This occurs because packets are enqueued for a long time and they expire their lifetimes before (or just after) they are forwarded by routers.

More precisely, the vast majority of Internet applications, including the Web, uses TCP as transport protocol. The TCP congestion control algorithm adjusts its transfer rate in face of an increasing latency, avoiding an unnecessary path saturation [12]. This algorithm considers, at a first instance, packet drops as an implicit indication of network congestion. This probing approach, combined with large FIFO-like buffers, leads to a significant buffer filling. As packet drops occur only when buffers are completely full, the use of large buffers may result on malfunctioning of the TCP congestion control algorithm. As a consequence, the TCP congestion control algorithm does not adjust its transmission rate adequately, degrading network performance.

In short, bufferbloat may be the key cause of the increasing end-to-end latency observed nowadays in the Internet. Large buffering may reduce the TCP performance, thus impacting the quality of the vast majority of Internet applications. This potential negative impact on network performance motivates the further investigation of the bufferbloat issue and its practical implications.

III. Bufferbloat Recent Solutions

There is a number of proposals in recent literature aiming at reducing the bufferbloat impact. Typically, such solutions work on the transport or network layers.

Solutions in the transport layer keep the network core unaltered. The adoption of such solutions is limited to updates on end hosts, such as applying operating systems kernel patches or updating domestic router firmwares. Typical solutions in the transport layer target delay-based congestion control, such as Low Priority Congestion Controls (LPCC) [5] algorithms. One of the most popular LPCCs is *LEDBAT* (Low Extra Delay Background Transport), which is a TCP alternative protocol originally created for Bittorrent P2P networks. Chirichella and Rossi [5] show that LEDBAT prevents the increased delay caused by queuing for most of Bittorrent users. A recurrent issue involving delay-based congestion controls, like LEDBAT or even the more traditional TCP Vegas [3], is the unfairness in concurrent flows. These delay-based congestion controls may lead to an irregular bandwidth distribution among applications, reducing the user quality of experience.

Solutions in the network layer involve AQM algorithms, such as Random Early Detection (RED) [6] and its variants, and need to be implemented in routers. Moreover, they tend to be hard to configure and to adapt to constant traffic changes in networks [7]. To fight bufferbloat, Nichols and Jacobson [19] proposed the CoDel queue management algorithm in a direct response to the recent bufferbloat phenomenon [7]. CoDel essentially intends to detect bad queues, which are those that grow harmfully without signs of deflation. In face of a bad queue, CoDel intentionally starts a packet drop phase in order to induce the activation of the TCP congestion control. CoDel is auto-configurable, which is an advantage over traditional AQM solutions. Despite its advantages, the use of CoDel during the TCP slow start phase is not expected to be effective [21]. This issue may be a problem because nowadays a vast number of TCP connections are very short.

Recent Linux kernels (i.e. newer than 3.3.0) present a new feature that attempts to solve the buffer sizing problem of network devices. This mechanism, known as *Byte Queue Limits* (BQL) [9], limits the size of the transmission hardware queue on a NIC by the number of bytes. In short, BQL is a self-regulating algorithm that intends to estimate how many bytes a network interface is capable of transmitting. Using BQL, the amount of packets sent to the *ring buffer* (the NIC related buffer – see Sec. IV-B) is reduced, shifting packet queuing to the upper layers (e.g. to the *qdisc*, a OS related queue). Then, buffering can be controlled using efficient queuing disciplines available on *qdisc*, enabling a more sophisticated queue management and allowing for latency reduction. The main goal is to reduce latency caused by excessive queuing in hardware without sacrificing throughput.

FQ-CoDel [10] and PIE (Proportional Integral controller Enhanced) [20] are recent queue management schemes. FQ-CoDel combines CoDel with stochastic fair queuing [16]. According to its authors, PIE can effectively control the average queueing latency to a reference value, with low overhead.

Despite the excessive buffer capacity in network devices and the consequent possibility of increased overall latencies, most of the existing studies focus on the *potential* negative impact of bufferbloat [7], [19]. These studies do not actually present a systematic study evaluating the impact of buffer size and its correlation with bufferbloat. In fact, as far as we know, Allman [1] presented the first systematic study of the bufferbloat problem, investigating the problem from a *macroscopic* view of large-scale network measurements. The conclusion is that, although the bufferbloat phenomenon may happen, the magnitude of the problem in real network traffic is rather modest.

In our previous work [4], we have presented a systematic analysis of bufferbloat considering a microscopic view of the buffer architecture of typical network devices. We have varied the sizes of two main network queues on Unix-based systems and we have notice the bufferbloat phenomenon only in specific cases. For example, we only notice a considerable impact on network latency when increasing the *ring buffer* size, a NIC queue. In contrast, changes to *qdisc*, a network related queue under OS control, do not significantly impact on TCP performance. Despite the analysis in this previous work, we have not conducted a systematic analysis of the most common approaches to deal with bufferbloat phenomenon, as contributed in this work.

Jarvinen and Kojo [13] have evaluated CoDel and PIE. Authors have simulated (on NS2) and compared the performance of both mechanisms against an aggressive RED variant called HRED (Harsh RED). They indicate that neither CoDel nor PIE really handle well load transients, whereas HRED outperforms both in load transient handling. Moreover, CoDel auto-tuning does not scale well with the load.

The coexistence of both AQM and LPCC based solutions has been recently studied by Gong et al. [8]. Ideally, both approaches should coexist transparently. Nevertheless, Gong et al. present evidences that such a coexistence may cause a problem called "*reprioritization*". Reprioritization occurs because

AQM queues try to limit the excessive usage of bandwidth to protect new and short duration flows. LPCCs, on the other hand, attempt to use the available bandwidth in low priority without interfering with other flows. As a consequence, LPCCs under the influence of AQMs have their low priority ignored and their flows become as aggressive as normal TCP flows, reducing the benefit o LPCCs usage to avoid network overload.

IV. Experimental Environment

A. Testbed

We have configured a testbed in a way to allow a systematic analysis of the buferbloat phenomenon and the interoperability of the existing solutions. We are then able to control important system parameters, such as the queue sizes at the network devices and OS modules. In short, our experiments allow the validation o end-to-end delay and throughput metrics.

Fig. 1 shows our testbed topology. Network consists of three machines, where two are MacMinis[1] acting as endpoints (A and C), running Ubuntu 13.10[2] with Linux kernel 3.11. Each MacMini has 1 GB of RAM and a Gigabit Ethernet interface. The last machine (B) is a PC with Ubuntu 14.04 and Linux kernel 3.13. This PC has 48 GB of RAM and 2 Gigabit Ethernet interfaces. We have connected testbed hosts through a VLAN containing all three nodes.

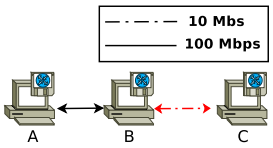


Fig. 1. Testbed configuration.

In our testbed, device B acts as a domestic router, the main network device with excessively large queue issues [7]. The link between hosts B and C is the network bottleneck. Every communication between the two end hosts must pass through this central router B. To perform the routing task, B runs the Quagga[3] routing software. Quagga is an open source software that implements the most important routing algorithms.

Without loss of generality, our testbed represents a typical home network scenario. Usually, between a domestic router and an ISP router, or even between small office networks, only one bottleneck link is observed. In other words, network end points have plenty of resources. Nevertheless, these end points are usually connected through one link with limited capacity.

B. Queue configuration

Devices queues and their configuration are the key elements of the bufferbloat microscopic analysis we propose in this paper. Queues are everywhere along a network path, including end hosts, routers, and switches. Hence, it is important to have in mind the general scenario that describes packet flows from that application layer to the link layer.

In this work, we focus on Unix-based devices, since most commercial network equipments are developed based on this OS. Besides, it is well-known that there are many Unix-based end hosts, servers and, even mobile devices.

Fig. 2 describes a Unix-based network protocol stack. We are particularly interested in the system behavior while we vary *qdisc* and *ring buffer* queue sizes.[4] These queues store packets in the network and link layers, respectively. Further, we focus our analysis on the output from the *ring buffer* and *qdisc* queues since their input queues commonly present sufficient packet processing capacity.

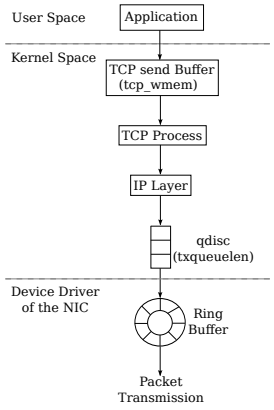


Fig. 2. Unix-based queue architecture.[5]

More precisely, the output *qdisc* queue is located between the network and link layers. In Unix systems, *qdisc* has a standard 1,000 packets storage capacity. We are able to configure this storage capacity using the *ifconfig* command, through the *txqueuelen* parameter. By default, *qdisc* uses FIFO as a queuing discipline. On the other hand, *ring buffer* is a queue located between the link and physical layers. It receives packets that arrive from the *qdisc* and delivers them to the NIC physical layer. The lower and upper bounds on the *ring buffer* size are defined by the NIC driver, which is rarely configured. In other words, the possibility of changing the *ring buffer* capacity using the command *ethtool* depends on the availability of such functionality by the NIC driver. The default *ring buffer* size in our testbed NICs is $\approx$ 512 packets.

V. Experimental Results

A. Methodology

During our experiments, we have performed long-lived TCP flow transmissions, as Jiang et al. [15]. We attempt to induce the occurrence of the bufferbloat phenomenon in our testbed (Figure 1) to analyze two main issues: (i) the impact of *ring buffer* and *qdisc* sizes on bufferbloat (Sec. V-B); and (ii) the impact of BQL and CoDel on bufferbloat as well as their coexistence (Sec. V-C).

For each experiment, we monitor the round trip time (RTT) with the *ping* tool for long TCP transfers from node A to node C. As Allman [1], we assume that the RTT varies proportionally to the variation of the queue occupation. Delay

[1] www.apple.com
[2] www.ubuntu.com
[3] www.nongnu.org/quagga/

[4] Typically, we refer *qdisc* and *ring buffer* queue sizes as the packet buffering capacity. In a practical implementation, however, these queues actually store descriptors that points to the real memory region that contains packets.

[5] Simplified version of picture 6-3 from [22].

fluctuations not caused by queuing are thus considered negligible. We also monitor the *throughput*. For all metrics in this scenario, the results are the mean values of 10 experiments. We also show the standard error of the mean (SEM $= \sigma/\sqrt{n}$), where σ is the standard deviation and n the number of samples.

To study existing solutions, such as CoDel and BQL, we used the *netperf-wrapper* tool[6] to perform experiments and collect data. In a given set of experiments, we have used the "*real time response under load*" *netperf-wrapper* option to ensure that the bottleneck link is stressed. During our experiments, *netper-wrapper* performs 50 downloads and a single upload. We have considered in the testbed different scenarios combining varied BQL limits, enabled/disabled CoDel, and the use of distint bufferbloat solutions (e.g. FQ-CoDel or PIE):

- BQL and CoDel disabled: There is no bufferbloat countermeasure;
- Auto-tuning BQL and CoDel enabled: Both solutions coexist to fight bufferbloat;
- BQL disabled and CoDel enabled: This is the default configuration for the considered Linux version;
- BQL with high limit and CoDel enabled: A badly configured BQL with CoDel;
- BQL with high limit and CoDel disabled: A badly configured BQL without CoDel;
- Auto-tuning BQL with CoDel, FQ-CoDel, and PIE;
- BQL with high limit with CoDel, FQ-CoDel, and PIE;
- BQL disabled with CoDel, FQ-CoDel, and PIE.

B. Impact of ring buffer and qdisc sizes

We first evaluate the impact of queue sizes on bufferbloat. Due to space constraints, we only show results while varying both *ring buffer* and *qdisc* sizes. The detailed impact of each one of these queues, with independent size variation, has been reported on our previous work [4].

According to Fig. 3, the variation of *qdisc* size has negligible impact on RTT. In contrast, the variation of the *ring buffer* size leads to a high RTT and, consequently, to the occurrence of the bufferbloat phenomenon. We clearly note that, for any *qdisc*, a 4k *ring buffer* imposes up to 14s for the RTT, whereas a small *ring buffer* results in a RTT in the order of 100ms.

We have also evaluated the impact of the *ring buffer* and *qdisc* sizes on throughput. Fig. 4 shows that the mean throughput tends to be slightly smaller with the increase of the *ring buffer* size. For instance, considering the range of sizes related to the *qdisc* buffer (0, 80, 2^8, 2^9, 2^{10}, 2^{11}, 2^{12}), the difference between the throughput reached by the smallest ring buffer size (80 packets) and the one reached by the largest ring buffer size (4096 packets) is lower than 20kbps for any *qdisc* size. Similarly to the RTT behavior, the variation of the *qdisc* buffer size does not significantly impact the throughput.

Finally, we have monitored the TCP advertised window during the file transfer. We have checked that, while the transfer sequence goes on, the advertised window also grows up, reaching a maximum value of 4MB. In short, the buffer storage capacity of the receiving host was not a limiting factor during transfers. In this case, we may conclude that RTT and throughput results have been only impacted by the bufferbloat phenomenon.

[6] github.com/tohojo/netperf-wrapper

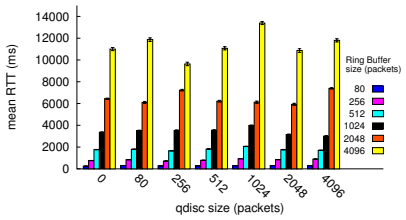


Fig. 3. Impact of buffer sizes on RTT.

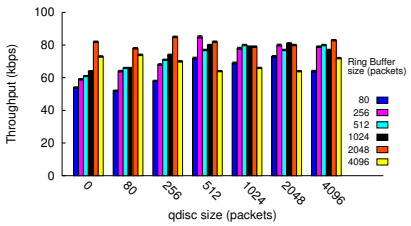


Fig. 4. Impact of buffer sizes on throughput.

C. BQL and CoDel Impact

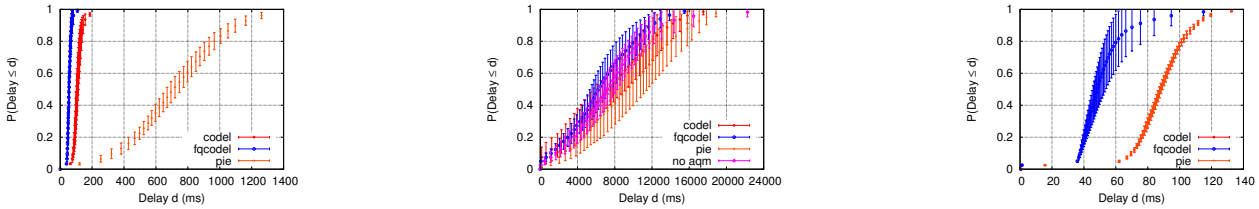
In this section, we evaluate the use of BQL and CoDel to avoid bufferbloat on susceptible routers.

Fig. 5 shows the CDF of the RTT delay on our testbed. We present data we gather from 10 experiments and the SEM of each percentile as error bars. We clearly note a high end-to-end latency with both BQL and CoDel disabled (Fig. 5(a)). In this case, less than 10% of all packets experienced a RTT lower than 3s. Such RTTs are far larger than expected. In fact, with both BQL and CoDel enabled (Fig. 5(b)), RTTs are no larger than 100ms. With only CoDel enabled (Fig. 5(c)), maximum RTT values are slightly larger than a system with both BQL and CoDel enabled. RTT in Fig. 5(c) presents a higher variance as compared with the scenario with both BQL and CoDel enabled (Fig. 5(b)). Both curves present some overlapping, suggesting they are actually statistically equivalent.

In short, the use of BQL and CoDel clearly mitigates bufferbloat. Furthermore, results shown in Fig. 5 might suggest that CoDel is the key mitigating actor. Nevertheless, under the presence of a non-autotunning BQL, CoDel is incapable of mitigating the bufferbloat effects. In fact, Fig. 5 shows that a virtually unlimited BQL turns RTT as high as the scenario where both, CoDel and BQL were disabled. In this case, it would be better to fully disable BQL than enabling it with a misconfiguration.

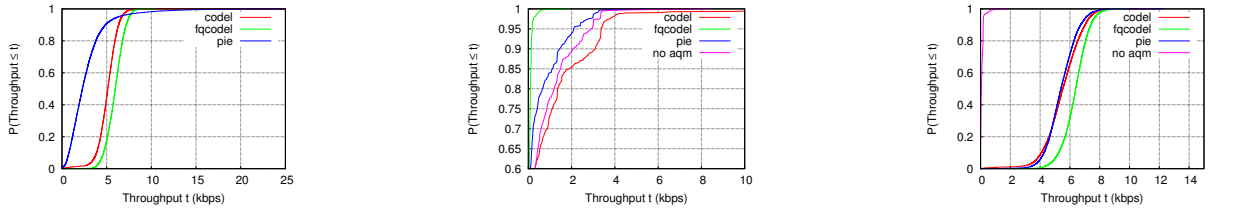
Finally, Fig. 6 shows the CDF of the throughput on our testbed with different BQL configurations. Fig. 6(a) shows auto-tunning BQL in use with FQ-CoDel, CoDel, and PIE. As shown in Fig. 6(a), the concurrent use of BQL and FQ-CoDel is slightly better than BQL and CoDel; e.g. the mean throughput is about 7 kbps in there former case, which is almost 40% better than the latter case. BQL with PIE presents a lower throughput, with mean values around 2.5 kbps.

As we noticed when analyzing RTT, despite the good performance of BQL on its standard configuration, if one turns off its auto-tunning mechanism, it might become useless.



(a) Auto tuning BQL with AQMs. (b) High limit BQL with AQMs. (c) Zero limit BQL with AQMs.

Fig. 5. Impact of BQL and AQMs on a typical bufferbloat scenario.



(a) Auto tuning BQL with AQMs. (b) High limit BQL with AQMs. (c) Zero limit BQL with AQMs.

Fig. 6. Impact of BQL configuration on throughput.

For instance, according to Fig. 6(b), all mechanisms working together with a virtually unlimited BQL present throughput worst than a system without any AQM approach. Only CoDel has presented a slightly better performance, yet, worse than a system without BQL (Fig. 6(c)).

We remark that scenarios with BQL disabled present better results, for all concomitant approaches, if compared with a misconfigured BQL. In this case, FQ-CoDel outperforms other approaches and, as shown in Fig. 5(c), latencies are low.

VI. Conclusions

In this paper, we present a systematic evaluation of bufferbloat, considering a typical network architecture. We conduct our experiments in a controlled testbed, using the most common bufferbloat solutions proposed in the recent literature, namely BQL, CoDel, FQ-CoDel, and PIE. Moreover, we also evaluate the interoperability of these bufferbloat solutions and the mutual impact they impose on each other.

Our experimental results show:

- Despite the recent buzz around the bufferbloat phenomenon, our results indicate that bufferbloat *may* only happen under very specific and unusual configurations of the network buffers. We only notice bufferbloat when one inadvertently changes the default ring buffer size.
- Recent versions of the Linux kernel avoid bufferbloat, even in the specific cases in which it was observed. In this case, any mechanism we have evaluate was efficient.
- They can work together without a noticeable interference. We only notice a bad performance under a a misconfigured BQL.

Acknowledgements

Authors thank Dave Täht (Bufferbloat.net) for suggestions concerning experiments with the netperf-wrapper tool.

References

[1] M. Allman. Comments on Bufferbloat. *ACM SIGCOMM Computer Communication Review*, 43(1):31–37, 2013.
[2] G. Appenzeller, I. Keslassy, and N. McKeown. Sizing Router Buffers. In *Proc. ACM SIGCOMM*, 2004.
[3] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson. TCP Vegas: New techniques for congestion detection and avoidance. In *Proc. ACM SIGCOMM*, 1994.
[4] T. B. Cardozo, A. P. C. da Silva, A. B. Vieira, and A. Ziviani. Bufferbloat systematic analysis. In *ITS 2014*.
[5] C. Chirichella and D. Rossi. To the Moon and back: are Internet bufferbloat delays really that large? In *Proc. IEEE INFOCOM Workshop on Traffic Measurement and Analysis*, 2013.
[6] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Trans. on Net.*, 1(4):397–413, 1993.
[7] J. Gettys and K. Nichols. Bufferbloat: Dark Buffers in the Internet. *Communications of the ACM*, 55(1):57–65, 2012.
[8] Y. Gong, D. Rossi, C. Testa, S. Valenti, and M. Täht. Fighting the bufferbloat: on the coexistence of AQM and low priority congestion control. *Computer Networks*, 2014.
[9] T. Herbert. Byte Queue Limits @lwn.net, nov 2011.
[10] T. Hoeiland-Joergensen, P. McKenney, D. Taht, J. Gettys, and E. Dumazet. Flowqueue-codel. *IETF Internet Draft*, 2014.
[11] O. Hohlfeld, E. Pujol, F. Ciucu, A. Feldmann, and P. Barford. BufferBloat: How Relevant? A QoE Perspective on Buffer Sizing. Technical report, Technische Universität Berlin, 2012.
[12] V. Jacobson. Congestion avoidance and control. *ACM SIGCOMM Computer Communication Review*, 18(4):314–329, 1988.
[13] I. Jarvinen and M. Kojo. Evaluating codel, pie, and hred aqm techniques with load transients. In *IEEE LCN*, 2014.
[14] H. Jiang, Z. Liu, Y. Wang, K. Lee, and I. Rhee. Understanding Bufferbloat in Cellular Networks. In *ACM SIGCOMM Workshop on Cellular Networks: Operations, Challenges, and Future Design*, 2012.
[15] H. Jiang, Y. Wang, K. Lee, and I. Rhee. Tackling Bufferbloat in 3G/4G Networks. In *Proc. 2012 ACM IMC*.
[16] N. Khademi, D. Ros, and M. Welzl. The new aqm kids on the block: Much ado about nothing? *http://urn.nb.no/URN:NBN:no-38868*, 2013.
[17] D. Lee, K. Cho, G. Iannaccone, and S. Moon. Has Internet Delay gotten Better or Worse? In *Proc. 5th CFI*, 2010.
[18] J. Nagle. On Packet Switches With Infinite Storage. *RFC 970*, 1985.
[19] K. Nichols and V. Jacobson. Controlling Queue Delay. *Communications of the ACM*, 55(7):42–50, 2012.
[20] R. Pan, P. Natarajan, C. Piglione, M. S. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg. Pie: A lightweight control scheme to address the bufferbloat problem. In *HPSR, 2013 IEEE 14th*, pages 148–155.
[21] D. Täht. Inside Codel and Fq Codel. Standford University Networking Seminar, January 2013.
[22] K. Wehrle, F. Pahlke, H. Ritter, D. Muller, and M. Bechler. *Linux Networking Architecture*. Prentice Hall, 2004.