Bufferbloat Systematic Analysis

Thiago B. Cardozo, Ana Paula C. da Silva, Alex B. Vieira, and Artur Ziviani

***Abstract*—There is a recent interest in the bufferbloat phenomenon as a possible explanation for the increased network latency observed in the Internet. The 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. Therefore, in this paper, we present a systematic analysis of the bufferbloat phenomenon considering the microscopic view of the insides in the buffer architecture of typical network devices. Our experimental results show that bufferbloat might not be a significant problem in practice. First, the phenomenon is only observed in specific cases. Second, changes made to the queues under the control of the operating system have negligible effects. Finally, recent versions of the Linux kernel avoid bufferbloat, even in the specific cases in which it was observed.**

***Index Terms*—Increased latency; network traffic; buffering**

I. Introduction

INTERNET has recently observed an increase in its end-to-end latency [1]. Recent literature points the bufferbloat phenomenon as one of the possible causes of this increase. 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 of time, degrading the network performance [2]. As a consequence, the past couple of years have witnessed a buzz in the networking community about bufferbloat [3]–[6].

Buffers in routers absorb traffic bursts, avoiding loss of data. Buffer sizing is indeed a recurrent challenge in the network area [7]. Excessively large queues 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 [8]. As buffers at routers are over-sized, the queue occupation increases without the TCP reducing its transfer rate. This behavior leads to large end-to-end delays caused by the extra overload in the network.

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 [2], [3]. They 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 [6] has recently presented the first systematic study of bufferbloat, investigating the problem from the *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 this paper, we present a systematic evaluation of the bufferbloat effect, considering the *microscopic* view of the insides in the buffer architecture of typical network devices. Different from previous works, we vary the size of the involved network buffers and evaluate key network metrics, such as the end-to-end latency and throughput. We conduct our experiments in a controlled testbed, using commercial hardware and free software. In common devices, there are two main queues: (i) a circular buffer associated to the network card (ring buffer) and (ii) a queue associated with the operating system (qdisc). Our microscopic view of the problem complements Allman's macroscopic bufferbloat study [6] because Allman focused on analyzing the bufferbloat phenomena without a deeper investigation on the internal queue mechanisms of the operational system and their impact on bufferbloat, which is the focus of our experimental study.

Our experimental results show that, in contrast to previous works [2]–[4], bufferbloat might not be a significant problem. In fact, Hohlfeld et al. [5] suggested that bufferbloat presents a minor effect on the Quality of Experience (QoE) of multimedia end users. Considering our study, we only notice a considerable impact on key network metrics while varying the ring buffer. In this case, when increasing this specific buffer, we experience an excessive latency, characterizing the bufferbloat phenomenon. In contrast, changes to qdisc do not significantly impact on metrics like end-to-end delay and transfer rate. Finally, we also present an analysis considering the mechanisms available in recent Linux kernels that were incorporated to mitigate the bufferbloat effects.

II. The Bufferbloat Phenomenon

Queues in packet switched networks are used to absorb packet arrival rates that can vary significantly in short periods of time [3]. Nowadays, due to the low cost of memory, routers often offer large amounts of buffering capacity, even in the simplest versions. The manufacturer's premise is that larger buffering capacities avoid packet loss, improving network quality of service to the end user.

However, network devices with large buffering capacity introduce the problem recently known as *bufferbloat* [2], [3]. 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 [9]. 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.

Thiago B. Cardozo and Artur Ziviani are with the National Laboratory for Scientific Computing (LNCC), Brazil (email: {thiagoc,ziviani}@lncc.br).

Ana Paula C. da Silva is with the Computer Science Department, Universidade Federal de Minas Gerais (UFMG), Brazil (email: ana.coutosilva@dcc.ufmg.br)

Alex B. Vieira is with the Computer Science Department, Universidade Federal de Juiz de Fora (UFJF), Brazil (email: alex.borges@ufjf.edu.br).

This work is partially supported by CNPq, FAPEMIG, FAPERJ, and PCI/MCTI.

One key side effect of the bufferbloat phenomenon is the malfunctioning of the TCP congestion control algorithm. The TCP congestion control algorithm adjusts its transfer rate in face of an increasing latency, avoiding an unnecessary path saturation [8]. This algorithm considers, at a first instance, packets drops as an implicit indication of network congestion. As router queues are getting larger recently, the path congestion occurs without a packet discard. As a consequence, the TCP congestion control algorithm does not adjust its transmission windows adequately, degrading the network performance.

In short, bufferbloat may be the key cause of the increasing end-to-end latency observed nowadays in the Internet. The large amount of buffer may reduce the TCP performance, which in turn impacts the quality of service of the vast majority of Internet applications. This potential negative impact on network performance motivates the in-depth investigation of the bufferbloat issue and its practical implications.

III. Experimental environment

A. Testbed

Our testbed is configured in a way that allows a systematic analysis of the bufferbloat phenomenon through the control of the buffer sizing at key network devices. Our experiments evaluate relevant metrics, such as end-to-end delay and throughput.

Figure 1 shows the testbed topology. The network consists of 5 MacMinis[1] running GNU/Debian 6.0[2] with Linux kernels 3.2 and 3.11. Each host has 1 GB of RAM and a Gigabit Ethernet interface. We have connected testbed hosts through 2 VLANs: the first one contains nodes A, B, C, and D while the second one contains the remaining hosts (D and E).

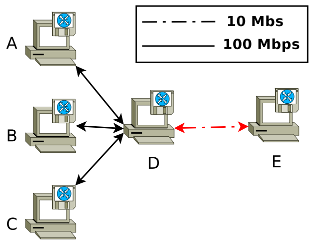


Fig. 1: Testbed configuration.

In our testbed, host D acts as a domestic router, being the main network device with excessively large queue issues [2]. Note that the link between hosts D and E is the network bottleneck. Every communication between two end hosts must pass through the central router. For example, data sent from host A to E must go through host D, forming a two hops path. To perform the routing task, host D 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, we only observe one bottleneck link. In other words, network end points have a lot of resources. Nevertheless, these end points are usually connected through one link with limited capacity. In this way, during our experiments, we assume that most network flows go through this single bottleneck.

B. Queue configuration

The key elements of the microscopic bufferbloat analysis we propose are the buffer sizing at the testbed network devices. Queues are everywhere in a network path, including end hosts, routers, and switches. In our work, we focus on Unix-based devices, since most network commercial equipments are developed based on this operating system. Further, there are many end hosts, servers, and even mobile devices that are Unix-based. Figure 2 describes a Unix-based network protocol stack. We are particularly interested in the system behavior while we vary the *qdisc* and *ring buffer* queues sizes.[4] These queues store packets in the network and link layers, respectively. Besides that, we focus our analysis on the output of the *ring buffer* and *qdisc* queues since their input queues commonly present sufficient packet processing capacity.

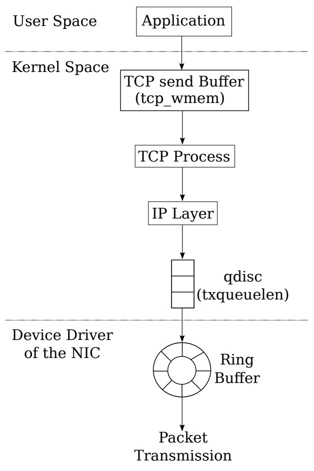


Fig. 2: Unix based queue architecture.[5]

More precisely, the output *qdisc* queue is located between the network and link layers. In Unix systems, the *qdisc* has a default capacity of 1,000 packets. 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. *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 (Network Interface Card) physical layer. The lower and upper bound *ring buffer* size limits are defined by the NIC driver, which are rarely configurable. In other words, the possibility of changing the *ring buffer* capacity using the command *ethtool* depends on the availability of such functionality by the NICs driver. The default *ring buffer* size in our testbed NICs is 512 packets.

IV. Experimental Results and Analysis

In our experiments, we have performed long-lived TCP flow transmissions, in a similar way to [11]. Our main goal is to induce the occurrence of bufferbloat in our testbed

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

[4]Typically, one refers to *qdisc* and *ring buffer* sizes as the capacity they have to store packets. In a practice, however, these queues actually store descriptors that point to the memory region that contains enqueued packets.
[5]Simplified version of picture 6-3 from [10].

(Figure 1). Node E, defined as a client, downloads files from three servers (nodes A, B, and C) using node D as a router. In each experiment, we have used a randomly generated file content of approximately 32 MB. We copy this file from servers to client E, simulating a long-lived TCP flow. The client downloads 10 files from each server, in a total of 30 concurrently downloads and about 1 GB of transferred data, ensuring that the bottleneck link is stressed.

For each experiment, we monitor the round trip time (RTT) with the ping tool as a probe flow for data collection. We also monitor achieved throughput between server nodes and the client node as well as the number of dropped packets. The results we present are the mean values of 10 experiments. Error bars show the standard error of the mean (SEM $= \sigma/\sqrt{n}$), where σ is the standard deviation and n the number of samples. Similarly to [6], we assume that the RTT variation is dominated by the queue occupation and consider negligible other fluctuations of delay unrelated to queuing.

A. *Experiments varying the ring buffer size*

In this section, we analyze the impact of the ring buffer size variation on the bufferbloat phenomenon, keeping the default qdisc size (1000 packets). This is a scenario equivalent to those of the original bufferbloat study [2].

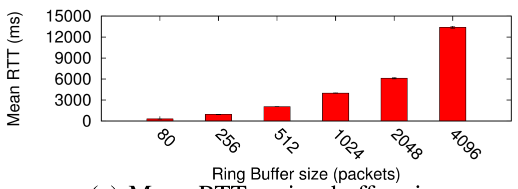
Figure 3(a) shows the RTT mean value while we vary the ring buffer size. We clearly notice that the RTT increases as the ring buffer size also increases. During our experiments, we have observed a negligible RTT value with small ring buffer sizes. Nonetheless, the RTT increases each time the ring buffer size is increased. In this case, we observe RTT values larger than 10 seconds with a ring buffer capacity of 4096 packets.

Figures 3(b) and 3(c) show that packet drops and retransmissions also suffer an impact with the increased size of the ring buffer. In fact, we observe an increasing number of retransmissions while the number of packet drops in the network router falls after a ring buffer of 80 packets. This happens because routers stop discarding packets due to lack of space since the TCP fast retransmit algorithm mistakenly starts the retransmission of packets that are still in the bottleneck queue. Our experimental results corroborate previous results discussed in [2]–[4]. Nevertheless, these previous results span only one dimension (the ring buffer size) of the packet flow over the network protocol stack on Unix-based equipments, as illustrated in Fig. 2. It is also interesting to observe the case when one keeps the ring buffer size on its default value and instead varies the size of the qdisc queue. Taking into account this new perspective, the system behavior changes and interesting results show up, as discussed in Section IV-B.

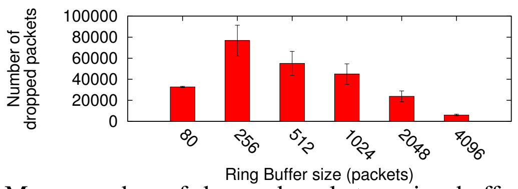
B. *Experiments varying the qdisc size*

We also analyze the impact of varying the qdisc size on network metrics. In this case, we keep the ring buffer default size unchanged (512 packets as defined by the NIC driver).

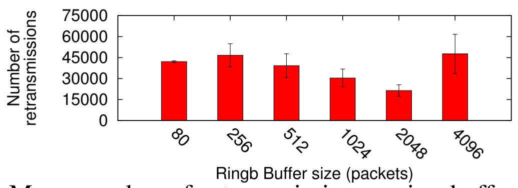
Figure 4(a) shows that as we increase the qdisc queue size, the RTT remains roughly stable. In contrast with the results shown in Section IV-A, there is no occurrence of *bufferbloat* and the RTT is one order of magnitude smaller. Moreover, as



(a) Mean RTT x ring buffer size.

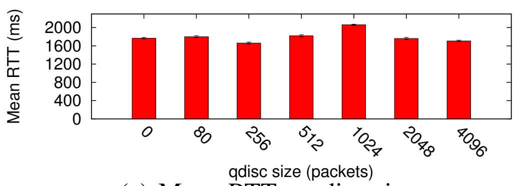


(b) Mean number of dropped packets x ring buffer size.

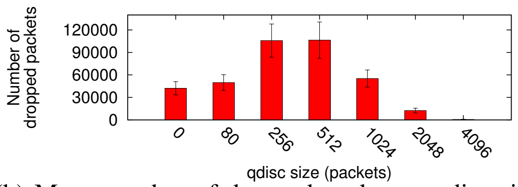


(c) Mean number of retransmissions x ring buffer size.

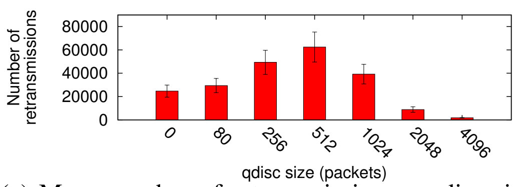
Fig. 3: Impact of the ring buffer size on network metrics.



(a) Mean RTT x qdisc size.



(b) Mean number of dropped packets x qdisc size.



(c) Mean number of retransmissions x qdisc size.

Fig. 4: Impact of the qdisc size on network metrics.

shown in Figures 4(b) and 4(c), the number of dropped packets and retransmissions increases up to a queue of 512 packets. After this point, both, packet drops and retransmissions fall to negligible values. This occurs because the qdisc is large enough to absorb traffic at the same rate it is being generated.

C. *Experiments varying both ring buffer and qdisc sizes*

Figure 5 shows the impact on RTT while we vary the size of both queues. As in Figure 4, queues larger than 4K packets are able to absorb all generated traffic. The results in Figure 5 are actually very similar to those presented in Figure 4. As previously shown, the qdisc size variation has

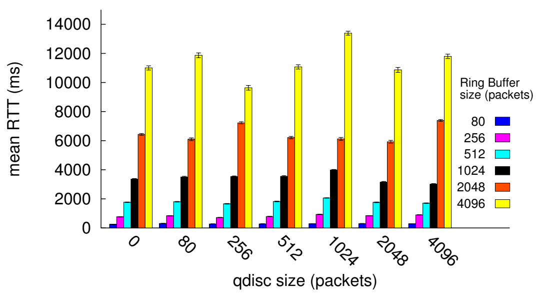


Fig. 5: Impact of buffer sizes on RTT.

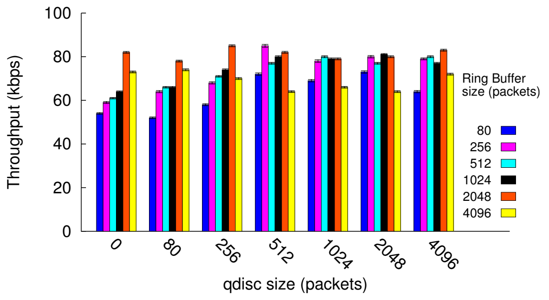


Fig. 6: Impact of buffer sizes on throughput.

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 also evaluate the impact of the ring buffer and qdisc sizes on throughput. Figure 6 shows that the mean throughput tends to be slightly stable 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 20%. In a similar way to the RTT, the variation of the qdisc buffer size does not significantly impact the throughput.

D. Advertised TCP window check

We conduct experiments to ensure that the receive buffer size is not limiting the throughput. Figure 7 shows the TCP advertised window during the transmission sequence. It can be observed that the advertised window grows while the transfer sequence advances, up to a maximum value of 4 MB. The TCP advertised window expansion, performed by *window scaling*[6], increases considerably the buffer capacity at the receiver, allowing senders to keep a high throughput during the experiment. Therefore, during our experiments, we can assume that the transfer throughput is not particularly limited by the TCP advertised window.

E. Byte Queue Limits (BQL) impact

In Linux kernel 3.3, a new functionality known as Byte Queue Limits (BQL) has been introduced. BQL is a self-regulating algorithm that intends to estimate how many bytes a network interface is capable of transmitting. Using BQL, the

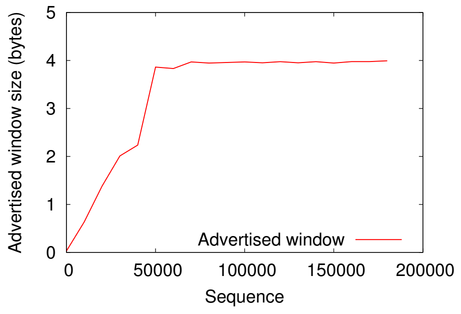


Fig. 7: TCP advertised window size.

amount of packets sent to the ring buffer is reduced, shifting the queuing to upper layers (e.g. to the qdisc). Buffering can then be controlled using efficient queuing disciplines available on qdisc, enabling a more sophisticated queue management and permitting latency reduction.

In this section, we present experiments aiming at the evaluation of the use of BQL in order to avoid bufferbloat on susceptible routers. We conduct these experiments using the same environment described in Section III-A, but now using ether Linux kernel 3.2 (without BQL) or 3.11 (with BQL).

To analyze the BQL impact on qdisc, we define that both qdisc and ring buffer together form a “virtual” queue with a fixed size of 5000 packets. We reduce the ring buffer size while increasing qdisc, keeping the “virtual” queue size unaltered. For example, with a 4096 packets ring buffer, qdisc is set to 904 packets. In this way, we can observe the system dynamics while we change the relative capacity of both queues.

Figures 8 and 9 show that BQL drastically reduces the effects of bufferbloat, even in the scenario in which we previously observed it (i.e., large ring buffer size). BQL usage reduces RTT up to two orders of magnitude. For example, in Figure 8, a ring buffer with a size of 2048 packets and a qdisc with 2952 packets, yields a RTT 95% percentile of 13 s. Meanwhile in Figure 9, the same queue configuration with BQL presents a RTT 95% percentile of 106 ms.

In contrast, as the BQL causes more packets to be held in the qdisc, we note a small 20 ms difference between the RTT 95% percentile in the case with a small qdisc (904 packets) and the case with a large one (4920 packets). In the cases qdisc size is determinant to latency increase, queue disciplines like CoDel[7] [3] can be deployed.

V. Discussion on existing solutions

There are a number of proposals in recent literature aiming at mitigating the bufferbloat impact. Typically, such solutions present implementations in the transport or network layers.

In the transport layer, solutions keep the network core unaltered. The adoption of such solutions is limited to updates at the end hosts, like kernel patches to operating systems or new firmwares on domestic routers. For example, in the transport layer, there is a delay-based congestion control, such as Low Priority Congestion Controls (LPCC) [4]. One of the most popular LPCCs is *LEDBAT* (Low Extra Delay

[6]Window scaling is an option available in recent TCP implementations that allows receiver advertised window to exceed its default limit of 65.525 bytes.

[7]CoDel is a recent Active Queue Management (AQM) algorithm motivated specifically by the bufferbloat phenomenon.

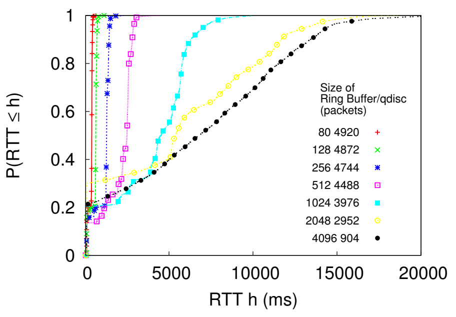


Fig. 8: RTT CDF on a router without BQL.

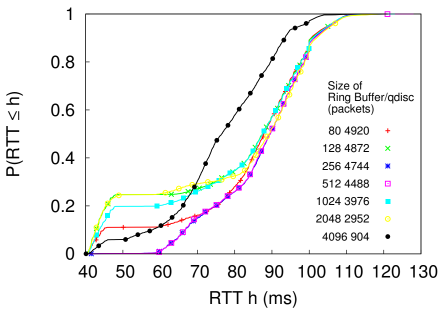


Fig. 9: RTT CDF on a router with BQL.

Background Transport), which is a TCP alternative protocol originally created for Bittorrent P2P networks. Chirichella and Rossi [4] 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 [12], is the unfairness for concurrent flows. These delay-based congestion control solutions may lead to an irregular bandwidth distribution among applications, and consequently to a reduction on quality of experience for the users.

In the network layer, Active Queue Management (AQM) algorithms, like Random Early Detection (RED) [13] and its variants, need to be implemented in routers. Moreover, they tend to be hard to configure and to adapt to constant changes in networks [2]. To fight bufferbloat, Nichols and Jacobson [3] proposed the CoDel queue management algorithm in a direct response to recent bufferbloat literature [2]. This algorithm essentially tries to detect bad queues, which are those that grow harmfully without signs of deflation. In face of a bad queue, CoDel intentionally drops packets to induce the activation of the TCP congestion control. CoDel is self-configurable, which is an advantage over traditional AQM solutions.

Finally, the coexistence of both AQM and LPCC-based solutions has been recently studied by Gong et al. [14]. Ideally, both approaches should be able to coexist transparently. Nevertheless, Gong et al. [14] present evidences that this coexistence may cause a problem called "*reprioritization*". This problem 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 without interfering with other flows that have low priority. 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 bufferbloat.

VI. Summary and Outlook

We present a systematic analysis of bufferbloat, considering a *microscopic* view of a typical network device architecture. We conduct our experiments in a controlled testbed, using commercial hardware and free software. Our experimental results show that, in contrast to previous works, bufferbloat might not be a significant problem. In fact, 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 highlight three key findings based on our experimental results: (i) small sizes of *ring buffer* allow the operating system to properly signal the presence of a "bad queue" (i.e., a persistently flooded buffer) and, as a consequence, the TCP inner controls still work properly; (ii) larger sizes of *qdisc* buffer allow the operating system to absorb burst traffic (i.e., a "good queue"), still minimizing packet drops and retransmissions; (iii) finally, the bufferbloat phenomenon is only noticed when one inadvertently changes the default *ring buffer* size (or the equivalent in other architectures). Further, we also show that recent versions of the Linux kernel avoid bufferbloat, even in the specific cases in which it was observed.

Overall, bufferbloat might not be the most plausible explanation for the recently observed increase in the end-to-end latencies in the Internet. Thus, we start considering other issues that may potentially increase network delays, such as the adoption of software emulated hardware and network virtualization. Investigation of such issues and their particular effects on the network delay is the target of our future work.

References

[1] D. Lee, K. Cho, G. Iannaccone, and S. Moon, "Has Internet Delay gotten Better or Worse?" in *Proc. 5th International Conference on Future Internet Technologies (CFI)*, Seoul, Korea, 2010.

[2] J. Gettys and K. Nichols, "Bufferbloat: Dark Buffers in the Internet," *Communications of the ACM*, vol. 55, no. 1, pp. 57–65, 2012.

[3] K. Nichols and V. Jacobson, "Controlling Queue Delay," *Communications of the ACM*, vol. 55, no. 7, pp. 42–50, 2012.

[4] 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.

[5] O. Hohlfeld, E. Pujol, F. Ciucu, A. Feldmann, and P. Barford, "BufferBloat: How Relevant? A QoE Perspective on Buffer Sizing," Technische Universität Berlin, Tech. Rep., 2012.

[6] M. Allman, "Comments on Bufferbloat," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 1, pp. 31–37, 2013.

[7] G. Appenzeller, I. Keslassy, and N. McKeown, "Sizing Router Buffers," in *Proc. ACM SIGCOMM*, 2004.

[8] V. Jacobson, "Congestion avoidance and control," *ACM SIGCOMM Computer Communication Review*, vol. 18, no. 4, pp. 314–329, 1988.

[9] J. Nagle, "On Packet Switches With Infinite Storage," *RFC 970*, 1985.

[10] K. Wehrle, F. Pahlke, H. Ritter, D. Muller, and M. Bechler, *Linux Networking Architecture*. Prentice Hall, 2004.

[11] H. Jiang, Y. Wang, K. Lee, and I. Rhee, "Tackling Bufferbloat in 3G/4G Networks," in *Proc. 2012 ACM IMC*, 2012.

[12] 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.

[13] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, 1993.

[14] 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*, vol. 65, pp. 255–267, 2014.