Simulation Modelling Practice and Theory 80 (2018) 1–18

Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat

An analysis of the impact of network device buffers on packet schedulers through experiments and simulations

Pasquale Imputato*, Stefano Avallone

Università degli Studi di Napoli "Federico II", Dipartimento di Ingegneria Elettrica e delle Tecnologie, dell'Informazione, Via Claudio, 21, Napoli, 80125, Italy

ARTICLE INFO

Article history:
Received 2 May 2017
Revised 27 September 2017
Accepted 30 September 2017

Keywords:
Experimental evaluation
Network simulations
Dynamic queue sizing
Active queue management

ABSTRACT

Keeping the delay experienced by packets while travelling from a source to a destination below certain thresholds is essential to successfully deliver a number of Internet services nowadays. Most of the packet delay can be usually ascribed to the time spent in the many queues encountered by the packet. In this context, the term bufferbloat has been recently coined to denote the uncontrolled growth of the queuing time due, among others, to the excessive size of the buffers and the attitude of TCP to increase the sending rate until a packet is dropped. In this paper, we focus on the queues employed by the *traffic control* infrastructure and by the network device drivers. Reducing the queuing time due to the former is the objective of a plethora of scheduling algorithms developed in the past years and referred to as Active Queue Management (AQM) algorithms. Conversely, the impact of the additional queuing in the buffer of the network device driver on performance and on the effectiveness of AQM algorithms has instead received much less attention. In this paper, we report the results of an experimental analysis we conducted to gain a better insight into the impact that network device buffers (and their size) have on performance. We also give an in-depth presentation of Dynamic Queue Limits (DQL), an algorithm recently introduced in the Linux kernel to dynamically adapt the size of the buffers held by network device drivers. The experiments we conducted show that DQL not only enables to reduce the queuing time in the network device buffers, which is essential to ensure the effectiveness of AQM algorithms, but also enables to keep latency stable, which is important to reduce the jitter. In order to faithfully reproduce through simulations the dynamics revealed by the experimental study we conducted, we implemented DQL for the popular ns-3 network simulator. In this paper, we describe the design of such implementation and report the results of a simulation study we conducted to show the ability of the simulator to accurately reproduce the experimental results.

1. Introduction

Network devices make a wide use of buffers to temporarily store packets waiting to be transmitted. The size of such buffers heavily influences network performance, primarily packet delay and loss probability. The buffer size should be large enough to accommodate packet bursts, but it should not be excessively large, in order to avoid that the latency experi-

* Corresponding author.
E-mail addresses: pasquale.imputato@unina.it (P. Imputato), stefano.avallone@unina.it (S. Avallone).

https://doi.org/10.1016/j.simpat.2017.09.008
1569-190X/

enced by packets grows in an uncontrolled manner, a problem which is commonly referred to as bufferbloat [1]. A common rule-of-thumb is to make the buffer size proportional to the link bandwidth (i.e., the so called Bandwidth-Delay Product). However, the use of such a rule has been questioned, e.g., in [2], where authors claim that much smaller buffers can be employed. In practice, the availability of memory chips at low cost often leads to the use of oversized buffers.

Limiting the queuing delay resulting from the use of large buffers has been the objective of a plethora of Active Queue Management (AQM) algorithms that have been defined by the research community in the last decades. RED (Random Early Drop) [3] has been one of the first AQM algorithms to be proposed and has been followed by a number of variants, such as Adaptive RED [4], Gentle RED [5] and Nonlinear RED [6]. RED and its derivatives can be classified [7] as queue-based schemes, since they aim to maintain the queue length at a target level. Other algorithms, such as BLUE [8] and AVQ (Adaptive Virtual Queue) [9] use instead the packet arrival rate as a congestion measure and hence can be classified as rate-based schemes. A combination of the two approaches is proposed by algorithms such as REM (Random Exponential Marking) [10]. More recently, CoDel (Controlled Delay) [11] has been proposed to specifically address the bufferbloat problem. CoDel aims to keep the queuing delay below a target value and drops packets that exceed such threshold *after* they have been dequeued. Furthermore, a few AQM algorithms, such as SFB (Stochastic Fair Blue) [12], AFCD (Approximated-Fair and Controlled Delay) [13] and FQ-CoDel [14], were proposed with the aim of additionally providing fairness among different flows.

The interaction with the TCP congestion control mechanism has been taken into account in the design of some AQM algorithms. The PI (Proportional Integral) [15] controller is among the earliest of such approaches and is based on a linearized dynamic model for a TCP/AQM system. PIE (Proportional Integral controller Enhanced) [16] computes the drop probability based on both the departure rate and the queue length. The stability of PIE is demonstrated by using a TCP fluid model. PI^2 [17] extends PIE with the objective to improve the coexistence between *classic* congestion controls (TCP Reno, Cubic, etc.) and *scalable* congestion controls (Data Center TCP). In [18], authors evaluate the interaction among AQM algorithms (such as CoDel) and low-priority congestion control techniques (such as LEDBAT [19]) through both experiments and simulations. A theoretical analysis is instead presented in [20] to study the interaction between Compound TCP and REM.

AQM algorithms have been implemented in real systems such as the Linux operating system as *queuing disciplines* within the Traffic Control (TC) infrastructure [21]. As such, AQM algorithms manage packets before they are handed to the device driver. The latter employs its own buffer, usually called *transmission ring*, to avoid the starvation of the network interface. Thus, packets are enqueued again after being scheduled by the queuing discipline. Also, a flow control strategy is implemented to regulate the passing of packets from the queuing discipline to the transmission ring. While the interactions between TCP congestion control and AQM algorithms have been addressed by a number of papers, the impact of such additional queuing and the related flow control remains largely unexplored. To the best of our knowledge, only [22] recognizes the presence of the transmission rings, but authors just model them as FIFO queues to evaluate their impact on the service guarantees of fair-queuing schedulers.

The goal of this paper is to fill this gap by evaluating the impact of transmission rings and flow control on network performance, in general, and on the effectiveness of AQM algorithms, in particular. We conducted a thorough experimental campaign which provided us with a number of insights. For instance, if the queuing discipline does not differentiate among traffic flows and its capacity is not saturated, the size of the transmission ring has no impact on network performance; when the queuing discipline assigns different priority levels to flows, the latency experienced by prioritized flows increases with the size of the transmission ring. It turns out, however, that sizing the transmission ring is not trivial: a too small ring causes a throughput loss, a too large ring causes a high packet delay. To complicate things further, for a given transmission ring size, the packet delay is affected by multiple factors, including packet size and CPU load. Also, rather unexpectedly, latency may exhibit a large variance and exceed by far the waiting time in the queuing discipline and in the transmission ring.

Another contribution of this paper is an in-depth description and experimental evaluation of Dynamic Queue Limits (DQL), a mechanism that has been recently introduced in the Linux kernel with the goal of dynamically computing the number of packets to store in the transmission ring in order to prevent starvation. Our experiments showed that DQL is effective in keeping latency low when AQM algorithms or priority packet schedulers are used, while it has no impact when a queuing discipline that enqueues all the packets in a single queue is used below its capacity. Also, when DQL is used, latency turns out to be rather stable and equal to the waiting time in the queuing discipline and in the transmission ring.

Furthermore, in order to allow for simulation studies to faithfully reproduce the dynamics revealed by our experimental campaign, we implemented DQL inside the ns-3 network simulator[1] and present the implementation design in this paper. This work builds upon our previous effort to introduce a traffic control layer resembling the Linux TC infrastructure within ns-3 [23]. With the introduction of the traffic control layer and DQL, ns-3 is now a reliable tool to evaluate the performance of AQM algorithms and higher layer protocols in terms of delay, packet loss and throughput. To this end, in this paper we present a simulation study we conducted to show that ns-3 simulations reproduce the experimental results rather accurately. We note that having DQL implemented in ns-3 enables to evaluate its performance also in contexts, such as wifi networks, that could not be tested with real experiments because of the lack of support from the Linux drivers.

The rest of this paper is structured as follows. Section 2 provides some background information about how queuing works in Linux, while Section 3 presents an in-depth description of DQL. Section 4 illustrates the design of our implementation

[1] Available since the ns-3.26 release.