Computer Networks 56 (2012) 3087–3098

Contents lists available at SciVerse ScienceDirect

Computer Networks

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

A spike-detecting AQM to deal with elephants ☆,☆☆

Dinil Mon Divakaran

Department of Electrical and Computer Engineering, National University of Singapore, Singapore 117576, Singapore

ARTICLE INFO

Article history:
Received 14 October 2011
Received in revised form 22 March 2012
Accepted 17 April 2012
Available online 11 May 2012

Keywords:
AQM
QoS
Elephants
Flows
Markov

ABSTRACT

The current TCP/IP architecture is known to be biased against flows of small sizes—small flows (or mice)—in the network, thereby affecting the completion times of small flows. A common approach taken to solve this problem is to prioritize small flows over large flows (elephants) during the packet-scheduling phase in the router. Past studies have shown that such 'size-based' priority schedulers improve the completion times of small flows with negligible affects on the completion times of large flows. On the flip side, most approaches are not scalable with increasing traffic, as they need to trace flows and estimate ongoing sizes of active flows in the router.

In this context, this work attempts to improve the performance of small flows using an active queue management (AQM) system, without needing to track sizes of flows. The core idea is to exploit a TCP property in detecting large 'spikes' and hence large flows, from which packets are dropped, and importantly, only at times of congestion. In this way, we use only a single queue, diverting from the multi-queueing systems used in size-based schedulers. We propose two spike-detecting AQM policies: (i) SDS-AQM that drops packets deterministically, and (ii) SDI-AQM that drops packets randomly. Using a simple Markov Chain model, we compare these new policies with the well-known RED AQM, highlighting the loss behavior. We also perform simulations, and using a number of metrics, compare the performance of (mostly) small flows obtained under the new AQMs against that obtained under the traditional drop-tail buffer, RED as well as a size-based flow-scheduler PS + PS. Surprisingly, RED is seen to give better performance than the size-based flow-scheduler developed specifically for improving the response times of small flows. Further, we find that the spike-detecting AQM policies give better performance to small flows than any other policy (including RED). Of the three scenarios we consider, two experiment with different buffer sizes—one with large buffer size (BDP) and another with small size (fraction of BDP). The third scenario considers the case where slow and fast flows compete. The results show that the spike-detecting AQM policies, unlike other policies, consistently give improved performance to small flows in all three scenarios. Of the two, the SDI-AQM performs better with respect to some metrics.

1. Introduction

Internet flow-size distribution exhibits strong heavy-tail behavior. This means that a small percentage of flows contribute to a large percentage of the Internet's traffic volume [2]. It is also known as the 80–20 rule, as 20% of the flows carry 80% of the bytes. It has become customary to call the large number of small flows *mice* flows, and the

☆ This work was done when the author was affiliated with IIT Mandi.
☆☆ This article is an extended version of the paper published in IEEE IPCCC 2011 [1]. In comparison to the conference paper, Section 2 discusses elaborately on related works, Section 4 is new, providing insights using a model based on Markov Chain, and Section 6 presents more results from simulations.

E-mail address: eledmd@nus.edu.sg

1389-1286/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.comnet.2012.04.025

small number of large flows *elephant* flows. Examples of mice flows include tweets, chats, web-search queries, HTTP requests, etc., for which users expect very short response times[1]; while elephant flows are usually the downloads that run in the background (say, a kernel image or a movie, involving MBs and GBs of data), the response times for which are expected to be higher than that for the mice flows by orders of magnitude.

The current Internet architecture has an FCFS server and a drop-tail buffer at most of its nodes. This, along with the fact that most of the flows in the Internet are carried by TCP [3], hurt the response times of mice flows adversely. Specifically, some of the important reasons for the biases are:

- As mice flows do not have much data, they almost always complete in the slow-start phase, never reaching the congestion-avoidance phase; and thus typically having a small throughput.
- A packet-loss to a small flow most often results in a time-out due to the small congestion-window (*cwnd*) size; and time-outs increase the completion times of small flows many-fold. On the other hand, a large flow is most probably in the congestion-avoidance phase, and hence has congestion-windows of large sizes. Therefore, for a large flow, packet-losses are usually detected using duplicate ACKs, instead of time-outs, thereby being able to recover faster.
- The increase in round-trip-time (RTT) due to large queueing delays hurts the small flows more than the large flows. Again, for the large flows, the large *cwnd* makes up for the increase in RTT.

The biases against small flows become more relevant today—recent studies show an increase in the mice-elephant phenomenon, with a stronger shift towards a 90–10 rule [4]. Most solutions to this problem can be grouped into a class of priority-scheduling mechanisms that schedule packets based on the ongoing sizes of the flows they belong to.

The priority schedulers, which we hereafter refer as *size-based* schedulers, give priority to 'potential' small flows over large flows, thereby improving the response time of small flows. They range from SRPT [5] to LAS [6] to MLPS scheduling policies [7]. The different size-based schedulers need to identify flows and distinguish between small and large flows. Most of these mechanisms have multiple queues with different priorities, and use the information of ongoing flow-sizes to decide on where to queue an incoming packet. Other works give priority to packets of small flows in space, that is in buffer. We observe that most of such works dealing with giving preferential treatment (either in space and/or in time) based on the size assume that the router keeps track of sizes of all flows. This assumption is challenged by the scalability factor, since tracking flow-size information requires flow-table update for every arriving packet. Given that the action involves lookup, memory access and update, this will require fast access as well as high power. Besides, as the number of flows in progress can grow to a large value under high load, this can also become an overhead. Hence, most existing solutions face a roadblock when it comes to implementation.

The spike-detecting AQM proposed here is inspired from the TLPS/SD (two-level-processor-sharing with spike-detection) system proposed in [8]. In TLPS/SD, a large flow is served in the high-priority queue, until it is detected as 'large,' which happens when its congestion-window is large enough ($>2^{\eta}$) to 'cause' congestion (buffer length $>\beta$) in the link (for pre-determined values of η and β). Such detected large flows are de-prioritized by serving them in a low-priority queue thereafter.

In this paper, we present spike-detecting AQMs (SD-AQMs in short). The major difference between SD-AQMs and the existing works that improve the response times of small flows (in comparison to the drop-tail buffer with FCFS scheduler), is that, SD-AQMs do not need to identify small and large flows. Second, these new AQMs do not need to track sizes of flows. Third, they use a single queue, removing the need for two or more virtual queues. Based on the core simple idea of detecting spikes, we present two policies: (i) *SDS-AQM* that drops packets deterministically, and (ii) *SDI-AQM* that drops packets randomly. We use '*SD-AQM policies*' to refer to both these policies.

As far as we know, there is also no existing work that study the performance of small flows under the well-known RED (random-early-drop) AQM policy [9]. Therefore, in this work, we also analyze performance of the RED AQM. Using a simple Markov Chain model we compare the performance of RED and SD-AQM policies, and highlight why small flows gain under SD-AQM policies in comparison to RED.

We then use simulations to study the performance of small flows under SD-AQM policies, RED, the traditional drop-tail (with FCFS scheduler) and PS + PS scheduler—a size-based scheduling strategy developed specifically to improve the response times of small flows. We perform studies using various metrics under three scenarios: (i) with bottleneck buffer size equal to BDP, (ii) with small bottleneck buffer (of size 1000 packets), and (iii) where slow and fast flows compete. In general, our observations reveal that small flows perform worst under the drop-tail approach. In comparison to drop-tail, the affects on large flows are negligible under other policies. The results in the large-buffer scenario show that while small flows under RED get similar performance as under PS + PS, medium- and large-size flows are less penalized in RED; whereas in the small-buffer scenario, RED performs better than PS + PS even for small flows. In both the scenarios, the SD-AQM policies are observed to perform better than all other policies. Not only small flows, but also medium- and large-size flows get better performance in SD-AQM policies. Between the two SD-AQM policies, SDI-AQM is a better choice, as it induces lesser number of timeouts on the overall traffic. The performance of SDI policy becomes more evident in the third scenario, where the flows taking the path with larger RTTs face a much lesser number of timeouts and congestion-window cuts in comparison to other policies.

The remaining of this paper is organized as follows. Next section discusses the previous works on mitigating

[1] We often use 'completion time' to refer to 'response time'.