Rethinking Buffer Management in Data Center Networks

Aisha Mushtaq[1], Asad Khalid Ismail[2], Abdul Wasay[1],
Bilal Mahmood[1], Ihsan Ayyub Qazi[1], Zartash Afzal Uzmi[1]
[1]SBASSE, LUMS, Lahore, Pakistan, [2]UC, Santa Barbara, USA
aishaa.mushtaq@gmail.com, asad@cs.ucsb.edu, 14100117@lums.edu.pk,
mahmood.b07@gmail.com, ihsan.qazi@lums.edu.pk, zartash@lums.edu.pk

ABSTRACT

Data center operators face extreme challenges in simultaneously providing low latency for short flows, high throughput for long flows, and high burst tolerance. We propose a buffer management strategy that addresses these challenges by isolating short and long flows into separate buffers, sizing these buffers based on flow requirements, and scheduling packets to meet different flow-level objectives. Our design provides new opportunities for performance improvements that complement transport layer optimizations.

1. INTRODUCTION

Many popular applications such as web search, advertising, and recommendation systems require low latency and high throughput from the underlying data center network [1]. Unfortunately, the completion times for latency-sensitive short flows in existing TCP-based deployments can be orders of magnitude larger than optimal [2]. This happens due to the *mixing* of short and long flows that have disparate performance expectations from the network fabric. As a result, short and long flows compete for a shared set of network resources (e.g., link capacity and switch buffers) in an adversarial manner. In particular, the buffer-filling nature of long-lived TCP flows often causes short flows to get queued up behind packets from long flows. In addition, the presence of long flows reduces the ability of switch buffers to absorb application-induced packet bursts that are common in data centers.

We propose MulBuff, a buffer management strategy that addresses the root cause of these problems that stem from the mixing of short and long flows. MulBuff uses three key design principles: (i) *segregation* of long flows and short flows into separate buffers, (ii) *sizing* of buffers based on flow requirements, and (iii) *scheduling* packets from these buffers to meet different flow-level objectives. We show that these principles complement each other and all are needed to achieve high performance across a wide range of scenarios.

Designs like MulBuff provide new opportunities for performance improvements that complement transport layer optimizations [1, 4, 5]. In particular, using MulBuff in conjunction with existing transports like DCTCP can provide significant performance benefits due to isolation of flows and by explicitly accounting for scenarios like incast. Our initial results show that TCP with MulBuff improves performance over DCTCP and achieves similar performance compared to pFabric [2].

SIGCOMM'14, August 17–22, 2014, Chicago, IL, USA.
ACM 978-1-4503-2836-4/14/08.
http://dx.doi.org/10.1145/2619239.2631462.

2. FRAMEWORK DESIGN

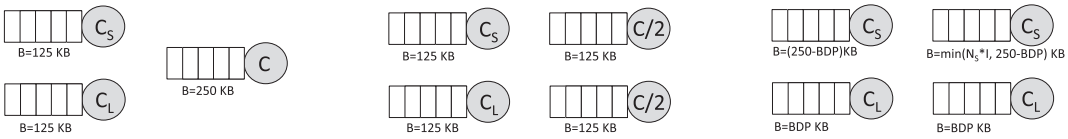
MulBuff's key design insight is a principled handling of flows with heterogeneous requirements at a switch. End hosts using MulBuff mark each packet as belonging to the short flow queue or the long flow queue[1]. The MulBuff switch maps flows to queues, updates sizes of the short flow queue (to keep a bounded loss rate and low delay) and the long flow queue (to keep high utilization), and adjusts queue capacities based on the number of flows in each queue[2]. We now study the impact of buffer organization, sizing, and scheduling on MulBuff's performance and show that they must be considered in unison to maximize performance because a lack of principled consideration of one factor can override the benefits of the other.

Buffer Organization: MulBuff's segregation of flows with different requirements into seperate queues allows flows to achieve their desired performance without impacting other flows. Figure 2(a) shows the average flow completion time (AFCT) with TCP for two buffering policies as a function of network load. Observe that having a single common buffer for both short and long flows significantly increases the AFCT. This happens because of the increased queuing delays caused by long flows' packets filling up the buffer. On the other hand, using multiple buffers isolates the impact of long flows on short flows leading to lower AFCTs.

Scheduling: Allocating fixed capacities to queues can degrade performance when more flows get mapped to one queue compared to the other or when flows have different performance requirements (e.g., to minimize completion times, short flows should be prioritized). Thus to meet different flow-level objectives (e.g., minimize FCTs, fair sharing), each queue should be allocated capacity proportional to the number of ongoing flows and operator assigned weights. Figure 1(b) shows the AFCT for processor sharing (PS) across queues and weighted processor sharing (WPS) as a function of load. Observe that FCTs increase rapidly with load under PS. This happens because PS assigns equal capacity to both the queues irrespective of the number of flows in each queue, which causes the FCT of short flows to increase. However, with WPS, as rates are assigned based to the number of flows in each queue, it results in a graceful increase in FCT (see Figure 2(b)).

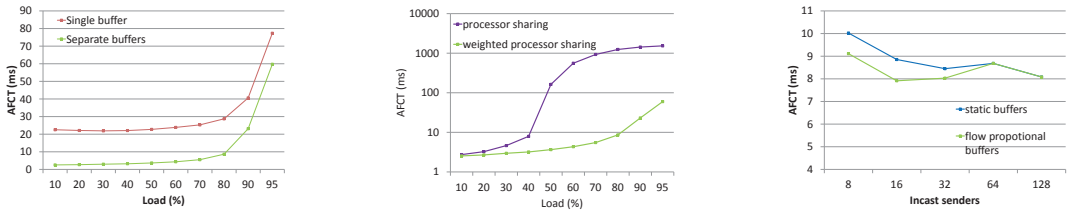
Buffer Sizing: The size of a buffer plays a key role in meeting the requirements of flows. For instance, latency-sensitive short flows often suffer TCP retransmission timeouts due to synchronized flow

[1]This is based on flow size information. If flow size is not known, it can be estimated using the amount of data sent so far [4].
[2]Note that MulBuff can use any number of queues.



(a) Scenario A (b) Scenario B (c) Scenario C

Figure 1: (a) Scenario A shows comparison of a single buffer with two buffers, (b) Scenario B shows comparison of static and dynamic link capacities (scheduling), and (c) Scenario C shows comparison of static and dynamic buffer sizing. Note that B is the buffer size, C=1 Gbps is the link capacity, C_S and C_L are the capacities for the short and long buffers, respectively, and N_S is the number of flows in the short buffer.



(a) Scenario A (b) Scenario B (c) Scenario C

Figure 2: For each scenario in Fig. 1, comparison of AFCT for short flows as a function of load. For scenarios (a) and (b), flow sizes were drawn from the interval [2 KB,98 KB] using a uniform distribution while keeping two long-lived flows in the background as done in [4]. In case of (c), a 1 MB file was divided equally among the senders.

arrivals. To prevent these timeouts and achieve a bounded loss rate, requires apportioning a certain amount of buffer space for each flow [3]. Thus MulBuff sizes buffers proportional to the number of flows for the short flow queue. For long flows to fully utilize the allocated capacity, the buffers must be large enough to prevent any queue underflow. Consequently, MulBuff sets the buffer size for the long flow queue to the bandwidth-delay product (BDP). Scenario C in Figure 1(c) compares the use of static and dynamic buffer sizing strategies. The static buffers setup uses BDP buffers for the long flows and (250-BDP) KB of buffer for the short flows. The other setup uses dynamic buffering for the short flow queue (for every new flow, the buffer size is increased by three packets and when a flow departs, it is decreased by three packets and the maximum size is capped at (250-BDP) KB. For these two setups, Figure 2(c) shows the AFCT results as a function of the number of senders in an ensuing incast. Observe that using dynamic buffering leads to improved FCTs especially when the number of senders are small as dynamic buffering helps in keeping small queues.

Overall, these results motivate the need to consider all these factors (buffer organization, scheduling, sizing) in unison.

Evaluation: Figure 3 shows the AFCT and the 99th percentile FCT for TCP with MulBuff, DCTCP, and pFabric on a 3-level tree topology in the network simulator (ns-2). Observe that MulBuff considerably improves performance over DCTCP and achieves similar performance compared to pFabric across a range of loads. Under incast, MulBuff's segregation ensures more headroom for short bursty flows, sizing allows dynamic allocation of buffers as flows arrive, and scheduling allows them to finish quickly (see Fig. 4(a)). Moreover, using MulBuff in conjunction with DCTCP provides up to 34% improvement in AFCT over DCTCP as shown in Fig. 4(b).

3. CONCLUSION AND FUTURE WORK

We make the following contributions in this paper: (a) We presented the design considerations needed for a buffer management strategy to achieve high performance across a range of data center workloads, (b) we presented a simple but novel solution that tackles buffer organization, sizing and scheduling, (c) we conducted evaluation of MulBuff in a typical data center environment. We

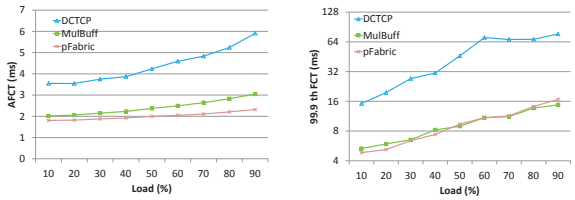


Figure 3: Comparison of TCP-MulBuff with DCTCP and pFabric in terms of (a) AFCT, (b) 99.9th perc. FCT under the all-to-all scenario.

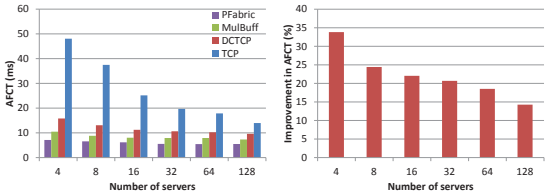


Figure 4: (a) Performance under the (intra-rack) incast scenario and (b) Impact of using DCTCP with MulBuff.

are currently implementing MulBuff in the Click modular router. In the future, we plan to evaluate the effects of choosing arbitrary scheduling policies, and how MulBuff behaves in the presence of legacy switches. In addition, we plan to explore how MulBuff can help address problems like TCP Outcast.

4. REFERENCES

[1] ALIZADEH, M., GREENBERG, A., MALTZ, D., PADHYE, J., PATEL, P., PRABHAKAR, B., SENGUPTA, S., AND SRIDHARAN, M. Data center tcp (dctcp). In *SIGCOMM'10*.

[2] ALIZADEH, M., YANG, S., SHARIF, M., KATTI, S., MCKEOWN, N., PRABHAKAR, B., AND SHENKER, S. pfabric: Minimal near-optimal datacenter transport. In *SIGCOMM'13*.

[3] MORRIS, R. Scalable tcp congestion control. In *INFOCOM'00*.

[4] MUNIR, A., QAZI, I. A., UZMI, Z. A., MUSHTAQ, A., ISMAIL, S. N., IQBAL, M. S., AND KHAN, B. Minimizing Flow Completion Times in Data Centers. In *INFOCOM'13*.

[5] VAMANAN, B., HASAN, J., AND VIJAYKUMAR, T. N. Deadline-aware datacenter tcp (d2tcp). In *SIGCOMM'12*.