Enabling Service Queue Isolation in Multi-Tenant Data Centers

Gyuyeong Kim, *Student Member, IEEE,* and Wonjun Lee, *Senior Member, IEEE*

***Abstract*—To isolate service queues in data centers, recent solutions assume Explicit Congestion Notification (ECN)-enabled end-hosts. However, this assumption is not valid in multi-tenant environments where tenants own the network stack. This letter presents BarberQ, a multi-queue management scheme that isolates service queues without dependency on transport protocols. At the heart of BarberQ lies dynamic buffer sharing through packet eviction. Our benchmark results demonstrate that BarberQ not only achieves minimum guarantees and work conservation, but also provides low latency.**

***Index Terms*—Data center networks, Multi-tenancy, Clouds**

I. Introduction

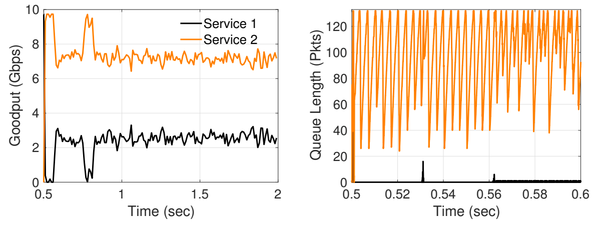
MODERN data center networks are shared by many services, which require low latency for small messages and high throughput for bulk transfers. Meanwhile, today's commodity switches support 4-8 class of service queues per port. Operators map traffic to service queues to provide differentiated network performance. The operator often groups service traffic into a class based on network policy. The switch identifies the service queue of the packet by referring DSCP value at the IP header. To isolate service queues, the switch uses work-conserving packet schedulers. However, as shown in Fig. 1, service queue isolation can be violated because the current best-effort buffer sharing policy allows a queue to monopolize the entire port buffer on the bottleneck.

Recent solutions [1], [2] utilize Explicit Congestion Notification (ECN), which enables the switch to limit the queue length around the marking threshold K. One underlying assumption of these solutions is that all end-hosts are ECN-enabled. However, this assumption does not hold in many multi-tenant environments where the network stack of end-hosts is owned by tenants. In multi-tenant data centers, VMs may not support ECN. Tenants may use recent non-ECN transport protocols, which deliver better performance than ECN-based protocols. Tenants may customize the network stack to accelerate their services as well. Fundamentally, the dependency on a specific transport setting makes it hard to adapt the evolution of network stack. Even if an optimized network stack is designed, it cannot be realized with the current inflexible end-hosts.

Manuscript received June 20, 2019; revised August 9, 2019; accepted September 2, 2019. Date of publication MM DD, 2019; date of current version MM DD, 2019. This research was partly supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (Ministry of Science and ICT) (No. 2019R1A2C2088812), Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (No. 2017M3C4A7083676). The associate editor coordinating the review of this letter and approving it for publication was Walter Cerroni. *(Corresponding author: Wonjun Lee)*

G. Kim and W. Lee are with the Network and Security Research Lab., School of Cybersecurity, Korea University, Seoul, South Korea. E-mail: {gykim08,wlee}@korea.ac.kr

Digital Object Identifier XX.XXXX/LCOMM.2019.XXXXXXX



(a) Goodput (b) Queue Length

Fig. 1. Goodput and queue length when two services share the bottleneck. Service 1 generates a flow at the beginning and service 2 generates 8 flows at 0.5 seconds. Although each service has the same weight and the bandwidth is expected to be guaranteed by packet schedulers, service 1 cannot grab a half of bandwidth due to excessive buffer occupancy of service 2.

The goal of this letter is *to isolate service queues without transport dependency for multi-tenant data centers.* A good solution should satisfy the following requirements:

- *Generic transports*: Our work should not rely on a specific end-host transport setting like ECN.
- *Bandwidth guarantees*: Whenever needed, a service queue should be able to obtain guaranteed bandwidth.
- *Work conservation*: Link capacity should be fully utilized. If there is spare bandwidth, a service queue should occupy bandwidth larger than the guaranteed share.
- *Low latency*: The flow completion time (FCT) of latency-sensitive small flows should be minimized for better user experience.

This letter presents BarberQ, a multi-queue management scheme that isolates service queues without dependency on transport protocols for multi-tenant environments. BarberQ achieves the goal without the help of end-hosts by trimming the tail of the longest service queue. Specifically, to enqueue packets of a unsatisfied service queue when the buffer has no rooms, the switch evicts the tail packet of the satisfied service queue whose the queue length is the longest. This dynamic buffer sharing enables the switch to isolate service queues with generic transport protocols. Our benchmark results demonstrate that BarberQ achieves the goal.

II. The BarberQ Design

A. Impact of Buffer Sharing on Service Queue Isolation

Consider an output port having buffer size B. M services share the link capacity C. We suppose that the switch maps a long-lasting flow of service i to queue i. For packet scheduling, we consider weighted fair schedulers like Weighted Round Robin (WRR) that packets in queue i are scheduled with the weight of w_i every rounds. The output rate[1] of queue i at time

[1]We use ‘output rate’ and ‘bandwidth’ interchangeably.

t can be given by $\mu_i(t) = \min(\lambda_i(t), \alpha_i(t))$ where $\lambda_i(t)$ is the input rate of queue i at time t. $\alpha_i(t)$ represents the weighted fair share rate for queue i at time t, which is determined by the buffer sharing policy and the packet scheduling algorithm.

To achieve line-rate throughput, the buffer size B must be larger than or equal to the Bandwidth Delay Product (BDP) that $B \geq C \times RTT$ where RTT denotes the base RTT. Note that today's commodity switches satisfy this. Let the minimum guaranteed bandwidth for queue i as $\mu_i^G = \frac{w_i}{\sum_{j=1}^{M} w_j} C$ where $\frac{w_i}{\sum_{j=1}^{M} w_j}$ is the normalized weight for queue i. Ideally, queue i should enjoy at least the guaranteed output rate on a congested link where $\sum_{j=1}^{M} \lambda_j(t) > C$ and $\lambda_i(t) \geq \mu_i^G$. For this case, $\mu_i(t) = \alpha_i(t)$. To ensure bandwidth guarantees for queue i, a solution must satisfy the following inequality that

$$\mu_i(t) \geq \mu_i^G \tag{1}$$

With the BDP, we have the required buffer size for queue i to satisfy μ_i^G on the bottleneck as $\gamma_i = \frac{w_i}{\sum_{j=1}^{M} w_j} C \times RTT$. Let $q_i^{\max}(t)$ denote the maximum queue length of queue i at time t. With γ_i, we have the following inequality that a solution must satisfy for Eq. (1) that

$$q_i^{\max}(t) \geq \gamma_i \tag{2}$$

The best-effort buffer sharing policy does not provide bandwidth guarantees because it fails to satisfy Eq. (2). Let $q_i(t)$ denote the queue length of queue i at time t. The policy leaves for a case of $B'(t) < \gamma_i$ as shown in Fig. 1 where $B'(t) = B - \sum_{j=1}^{M-1} q_j(t)$ denote the free port buffer size at time t. $\sum_{j=1}^{M-1} q_j(t)$ is the overall buffer occupancy except queue i. Thus, Eq. (1) cannot be satisfied as well.

A strawman solution to ensure bandwidth guarantees is to reserve a static weighted buffer share for queue i that

$$\tau_i = \frac{w_i}{\sum_{j=1}^{M} w_j} B \tag{3}$$

In this case, Eq. (2) is satisfied since $q_i^{\max}(t) = \tau_i$ at any time and $\tau_i > \gamma_i$ due to the BDP. Naturally, the solution satisfies Eq. (1) as well.

Unfortunately, the strawman is not work-conserving. The available bandwidth at time t can be expressed as $C'(t) = C - \sum_{j=1}^{M-1} \mu_j(t)$ where $\sum_{j=1}^{M-1} \mu_j(t)$ is the sum of output rates of all service queues except queue i. To achieve work conservation, the output rate of queue i at time t when $C'(t) \geq \mu_i^G$ must be $\mu_i(t) = C'(t)$. This results in $C = C'(t) + \sum_{j=1}^{M-1} \mu_j(t) = \sum_{i=1}^{M} \mu_i(t)$.

Similar to γ_i, the required buffer size for queue i to satisfy $\mu_i(t)$ at time t is

$$\kappa_i(t) = C'(t) \times RTT \tag{4}$$

To ensure work conservation, a solution must satisfy the following inequality when $B'(t) \geq \kappa_i(t)$ that

$$q_i^{\max}(t) \geq \kappa_i(t) \tag{5}$$

With the strawman, it is possible to see a case of $\tau_i < \kappa_i(t)$ depending on w_i and B. Recall that $q_i^{\max}(t) = \tau_i$ for the strawman even when $B'(t) > \tau_i$ at time t. In addition, the FCT can be lengthened because the shallow buffer is vulnerable to bursty packet loss [3].

Algorithm 1 The pseudocode of BarberQ

```
1: if $size(P_i) + \sum_{i=1}^{M} q_i > B$ then                ▹ Buffer overflow
2:     if $(size(P_i) + q_i) < \tau_i$ then                    ▹ Unsatisfied queue $i$
3:         $j \leftarrow 0$                                    ▹ Initialize victim queue index
4:         for $k \leftarrow 0$ to $M$ do                       ▹ ind victim queue
5:             if $q_k \geq \tau_k$ and $q_k > q_j$ then          ▹ Longer satisfied queue?
6:                 $j \leftarrow k$                            ▹ Update index
7:             end if
8:         end for
9:         while $size(P_i) + \sum_{i=1}^{M} q_i > B$ do
10:            if $(q_j - size(P_{j,tail})) < \tau_j$ then
11:                return Drop($P$)                  ▹ Drop if queue $j$ can be penalized
12:            end if
13:            Evict($P_{j,tail}$)                   ▹ Evict tail packet $P_{tail}$ in queue $j$
14:        end while
15:    else                                                    ▹ Satisfied queue $i$
16:        return Drop($P_i$)                                  ▹ Drop packet $P_i$
17:    end if
18: end if
19: return Enqueue($P_i$)                                      ▹ Enqueue packet $P_i$
```

B. Isolating Service Queues by Trimming the Tail

We aim at achieving bandwidth guarantees, work conservation, and low latency, without transport dependency. BarberQ achieves the requirements by trimming the tail of the longest service queue. The switch allows a queue to occupy the entire port buffer like the best-effort policy. However, when a queue of the arriving packet is unsatisfied, the switch reallocates per-queue buffer share by evicting buffered packets. This buffer replacement through packet eviction is the key to achieve both bandwidth guarantees and work conservation because it allows the switch to reallocate buffer shares among service queues dynamically. The idea of dynamic replacement by evicting buffered data for a new data is studied intensively for caching and paging in computing area. In networking area, there exist a few works that use buffer replacement (e.g. microburst absorption [3]). However, no work concerns service queue isolation in multi-tenant data centers. The BarberQ algorithm in Alg. 1 consists of two steps, victim queue search and victim packet eviction as follows.

Victim Queue Search. When buffer overflow occurs (Line 1), the switch checks whether queue i of the arriving packet P occupies the buffer less than the weighted share τ_i with P (Line 2). If the condition is true (i.e. queue i is unsatisfied), the switch begins reallocating per-queue buffer share. The victim queue is queue j where the queue length is the longest among satisfied queues (Lines 3-8). This is to maintain guaranteed bandwidth of queue j that q_j should be larger than or equal to τ_j after packet eviction.

Victim Packet Eviction. After searching the victim queue j, the switch evicts the buffered tail packet of queue j (Line 13). To prevent throughput loss of the victim queue, we stop the operation if $q_j < \tau_j$ is expected after eviction (Lines 10-12). Otherwise, the switch evicts packets until the enough buffer space is obtained (Line 9). The switch finishes the operation by enqueueing the arriving packet P (Line 19). BarberQ evicts multiple packets for a case of $size(P_i) > size(P_{j,tail})$. However, we rarely see multiple evictions because the switch observes typically MTU-sized packets. Note that even if end-hosts use Large Send Offload (LSO), the NIC splits the batch into the MTU-sized packets before transmitting them.

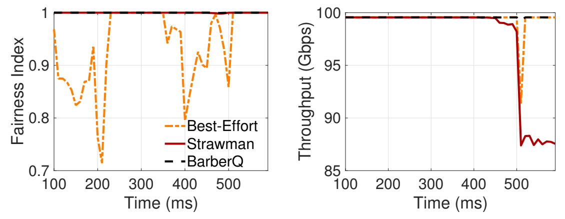
To prevent eviction of latency-sensitive small flow packets, we can configure a combination of Strict Priority Queueing (SPQ) and a weight fair scheduling like WRR where small flows packets are enqueued into the SPQ queue regardless of services. Since most of bytes are from large flows [4], the SPQ queue is rarely selected as the victim queue.

BarberQ is work-conserving because it allows queue i to occupy the buffer up to $B'(t)$ at time t like the best-effort. BarberQ also provides bandwidth guarantees by obtaining τ_i even when the buffer has no vacancy. Thus, we can express the maximum buffer size that queue i can occupy at time t in BarberQ as $q_{i,BQ}^{\max}(t) = \max(\tau_i, B'(t))$ where $B'(t) \geq \kappa_i(t)$ and $\tau_i \geq \gamma_i$ satisfy Eq. (4) and Eq. (2). With $q_{i,BQ}^{\max}(t)$, we can express the output rate of queue i at time t on the bottleneck when we use BarberQ as $\mu_{i,BQ}(t) = \max\left(\mu_i^G, C'(t)\right)$ where μ_i^G ensures bandwidth guarantees in Eq. (1) and $C'(t)$ preserves work conservation in $\mu_i(t)$.

Hardware Implementation. BarberQ operations are at the end of ingress pipeline. In the following, we describe the hardware implementation cost of BarberQ operations. We consider hardware running at a typical clock frequency of 1 GHz where 1 clock cycle is $1ns$. Finding the longest service queue through a linear search requires n clock cycles where n is the number of service queues. Since commodity switches generally support 4~8 service queues, the complexity are bounded to 4~8 clock cycles. Evicting the tail packet in the longest service queue is the same as dequeueing operation except that ours happen at the tail. Therefore, the operation can be implemented using `pop` primitive with 1 clock cycle. The processing overhead of BarberQ is relatively small because per-packet processing delay is generally hundreds of ns. For example, Broadcom Trident 3 100G ASIC provides a minimum per-packet processing delay of $800ns$. For this case, only 0.63%~1.13% (5~$9ns$) of additional processing delay is required to use BarberQ.

Discussions. We now discuss state-of-the-art mechanisms supported in modern switches. Some switches provide a parameter to limit the queue length of a port. Unfortunately, the parameter only concerns buffer sharing among ports, not among service queues in a port. In a similar vein, the strawman in this paper limits the queue length of each service queue. Although the strawman can ensure bandwidth guarantees, it is not work-conserving because a service queue cannot occupy large than the dedicated buffer space, which can be less than the BDP. The switch can limit bandwidth share of a service queue as well. However, without buffer management, a service queue can occupy excessive buffer space regardless of bandwidth limiting. In addition, it is not work-conserving since a service queue cannot utilize the whole link capacity.

Meanwhile, some vendors provide deep buffered switches having a large off-chip DRAM buffer (e.g. Arista 7280R). However, they are too slow to meet latency requirements of modern applications due to lack of cut-through switching support and delay to communicate with the ASIC. It can also cause excessive buffering of packets. Similarly, the switch can assign a large buffer to a single port using a shared buffer mechanism. However, it can harm fairness among ports by taking excessive buffers, which can be assigned to the other



(a) Per-Queue Fairness (b) Aggregate Throughput

Fig. 2. Time series of fairness index and aggregate throughput of service queues. We observe that BarberQ is the only solution achieving bandwidth guarantees and work conservation at the same time.

ports. A large queueing delay caused by excessive buffering can happen as well. Compared to the above mechanisms, BarberQ is the only solution that ensures bandwidth guarantees and work conservation at the same time by evicting buffered packets, which makes free buffer space on demand without harming per-port fairness and excessive buffering.

III. Performance Evaluation

A. Microbenchmarks

1) Methodology: We compare BarberQ with the following schemes: Best-Effort and Strawman. We build a star topology where multiple servers are connected to a single switch to simulate a compute rack. We consider a switch with Broadcom Trident 3 ASIC that supports 100Gbps and 32MB of buffers among 32 ports where the switch can assign 1MB of buffer per port. The port buffer is completely shared among service queues except Strawman. The base RTT is $40\mu s$ where $9\mu s$ is the host delay and $1\mu s$ is the link delay. The Jumbo frame is enabled. For packet scheduling, we use WRR that drains packets from each queue with the quantum of 9000 bytes. We use TCP for the transport protocol. We set RTO_{min} to 5ms.

In this benchmark, 8 services are mapped to 8 service queues whose weights are equal. Each service i generates $2 \times i$ flows at the beginning. Services $2 \sim 8$ stop flows every $50ms$ from $200ms$ in order. Therefore, service 1 is the only active service at $500ms$. We measure the throughput of each queue every $10ms$, and calculate Jain's fairness index and the aggregate throughput. Jain's fairness index measures how service queues share link capacity fairly. If a solution provides bandwidth guarantees and work conservation, the fairness index and aggregate throughput should be high.

2) Results: Fig. 2 shows the results. We find that BarberQ provides bandwidth guarantees and work conservation simultaneously. Since Best-Effort allows a service with many flows to occupy more buffer space, bandwidth cannot be shared fairly as shown in Fig. 2 (a). We also observe that Strawman is not work-conserving when few services are active in Fig. 2 (b) because a service queue cannot occupy the buffer larger than the assigned dedicated buffer space. Best-effort also results in throughput degradation for a moment at $500ms$. However, unlike Strawman, it is due to the time to fill up the bandwidth and buffer, not the lack of buffer space.

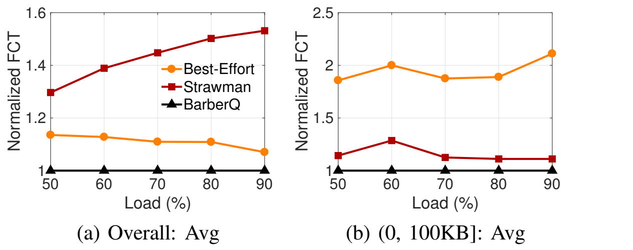


Fig. 3. FCT comparison with non-ECN schemes. We can see that BarberQ offers the best performance between non-ECN schemems.

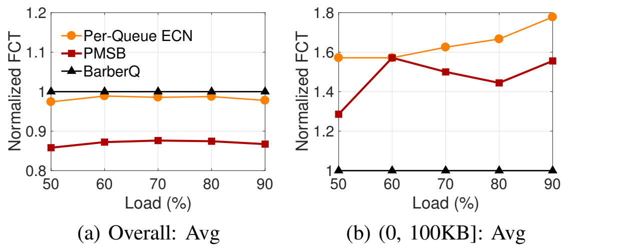


Fig. 4. CT comparison with ECN schemes. The gap in the FCT of overall flows is due to the impact of ECN and packet eviction.

B. Macrobenchmark

1) Methodology: We build a non-blocking leaf-spine topology, which has 12 leaf (ToR) switches and 12 spine (Core) switches. Each leaf switch has 12×100Gbps of downlinks and 12×00Gbps uplinks. We have 144 hosts and the base RTT is $44\mu s$. We use ECMP as the load balancing scheme. We use the same settings as in the microbenchmarks except packet schedulers, SPQ/DWRR. We reserve one queue for SPQ and the other queues for DWRR queues on the lowest priority. To use service queues, we suppose that services employ the two-level PIAS [5] where priority demotion threshold is 100KB. Therefore, packets of flows whose bytes sent are less than 100KB are enqueued into the shared SPQ queue while the remaining packets are enqueued into the DWRR queues.

We use 4 workloads [1] derived from production data centers running a web search, a data mining, a Hadoop, and a caching service. We make a communication pair with randomly chosen source and destination hosts. We evenly classify the 144×143 pairs into 7 services and each service has its own service queue. We open persistent connections between the hosts of each pair. We generate 10K flows whose inter-arrival time follow a Poisson process by varying traffic load from 50% to 90%. We consider the average FCT of overall flows and small flows (≤ 100KB). The FCT of small flows is crucial to user experience of modern services [4]–[6]. Therefore, our solution should minimize the FCT of small flows. The FCT of overall flows is generally similar to the FCT of large flows since most of bytes are from large flows. The results are normalized to the values of BarberQ. We also consider Per-queue ECN and PMSB [2]. For Per-queue ECN, each service queue has the same marking threshold of 550KB. The switch marks the packet when the queue length of the service queue exceeds the threshold. PMSB marks the packet when the port queue length is large than the 550KB of port threshold and the queue length of the service queue exceeds the weighted per-queue threshold (≈ 69KB) simultaneously.

2) Results: We make the following observations.

Comparison with Non-ECN Schemes. Fig. 3 shows the results. For the average FCT of overall flows, BarberQ achieves the best performance. The gaps stem from the lack of bandwidth guarantees (Best-Effort) or work conservation (Strawman). For example, Strawman causes more packet losses than BarberQ due to the lack of buffer space that a service queue can occupy. For the average FCT of small flows, it is easy to see that BarberQ outperforms the other schemes. This result suggests that BarberQ is effective to improve user experience of latency-sensitive applications.

Comparison with ECN-based Schemes. We now evaluate the performance of BarberQ against ECN-based schemes. We use ECN* as the transport protocol [7] for the ECN-based schemes to see the net effect of ECN. Fig. 4 reports the FCT results. The results for overall flows in Fig. 4 (a) show that BarberQ underperforms the comparisons. For example, BarberQ is worse than PMSB by up to 0.87×. This is because ECN avoids packet loss by notifying congestion proactively and BarberQ evicts buffered packets, causing more packet loss. The reason why the gap is not too much is that the impact of packet loss decreases as the flow size increases.

In Fig. 4 (b), we find that BarberQ is better than the ECN-based schemes for the average FCT of small flows. Our solution beats PMSB by 1.46× on average. This is because BarberQ rarely evicts small flow packets in the SPQ queue owing to the heavy-tailed flow size distribution in data centers [4]. The victim queue is mostly one of DWRR queues where most of bytes are buffered.

IV. Conclusion

This letter presented BarberQ, a multi-queue management scheme to enable service queue isolation in multi-tenant environments. BarberQ trims the tail of the longest service queue to reallocate buffer share dynamically. Through extensive benchmarks, we showed that BarberQ achieves bandwidth guarantees, work conservation, and low latency.

References

[1] W. Bai, L. Chen, K. Chen, and H. Wu, "Enabling ecn in multi-service multi-queue data centers," in *Proc. of USENIX NSDI*, pp. 537–549, 2016.
[2] Y. Pan, C. Tian, J. Zheng, G. Zhang, H. Susanto, B. Bai, and G. Chen, "Support ecn in multi-queue datacenter networks via per-port marking with selective blindness," in *Proc. of IEEE ICDCS*, pp. 33–42, July 2018.
[3] G. Kim and W. Lee, "Absorbing microbursts without headroom for data center networks," *IEEE Communications Letters*, vol. 23, pp. 806–809, May 2019.
[4] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," in *Proc. of ACM SIGCOMM*, pp. 63–74, 2010.
[5] W. Bai, K. Chen, H. Wang, L. Chen, D. Han, and C. Tian, "Information-agnostic flow scheduling for commodity data centers," in *Proc. of USENIX NSDI*, pp. 455–468, May 2015.
[6] K. Alhazmi, A. Shami, and A. Refaey, "Optimized provisioning of sdn-enabled virtual networks in geo-distributed cloud computing datacenters," *Journal of Communications and Networks*, vol. 19, pp. 402–415, August 2017.
[7] H. Wu, J. Ju, G. Lu, C. Guo, Y. Xiong, and Y. Zhang, "Tuning ecn for data center networks," in *Proc. of ACM CoNEXT*, pp. 25–36, 2012.