ABM: Active Buffer Management in Datacenters

Vamsi Addanki* — TU Berlin
Maria Apostolaki* — Princeton University
Manya Ghobadi — MIT
Stefan Schmid — TU Berlin
Laurent Vanbever — ETH Zurich

ABSTRACT

Today's network devices share buffer across queues to avoid drops during transient congestion and absorb bursts. As the buffer-per-bandwidth-unit in datacenter decreases, the need for optimal buffer utilization becomes more pressing. Typical devices use a hierarchical packet admission control scheme: First, a Buffer Management (BM) scheme decides the maximum length per queue at the device level and then an Active Queue Management (AQM) scheme decides which packets will be admitted at the queue level. Unfortunately, the lack of cooperation between the two control schemes leads to *(i)* harmful interference across queues, due to the lack of isolation; *(ii)* increased queueing delay, due to the obliviousness to the per-queue drain time; and *(iii)* thus unpredictable burst tolerance. To overcome these limitations, we propose ABM, Active Buffer Management which incorporates insights from both BM and AQM. Concretely, ABM accounts for both total buffer occupancy (typically used by BM) and queue drain time (typically used by AQM). We analytically prove that ABM provides isolation, bounded buffer drain time and achieves predictable burst tolerance without sacrificing throughput. We empirically find that ABM improves the 99th percentile FCT for short flows by up to 94% compared to the state-of-the-art buffer management. We further show that ABM improves the performance of advanced datacenter transport protocols in terms of FCT by up to 76% compared to DCTCP, TIMELY and PowerTCP under bursty workloads even at moderate load conditions.

CCS CONCEPTS

• **Networks** → **Data path algorithms**; **Data center networks**.

KEYWORDS

Datacenter, Shared Buffer, Buffer Management, Queue Management.

ACM Reference Format:
Vamsi Addanki, Maria Apostolaki, Manya Ghobadi, Stefan Schmid, and Laurent Vanbever. 2022. ABM: Active Buffer Management in Datacenters. In *ACM SIGCOMM 2022 Conference (SIGCOMM '22), August 22–26, 2022, Amsterdam, Netherlands.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3544216.3544252

*Both authors contributed equally to this research.

SIGCOMM '22, August 22–26, 2022, Amsterdam, Netherlands
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9420-8/22/08…$15.00
https://doi.org/10.1145/3544216.3544252

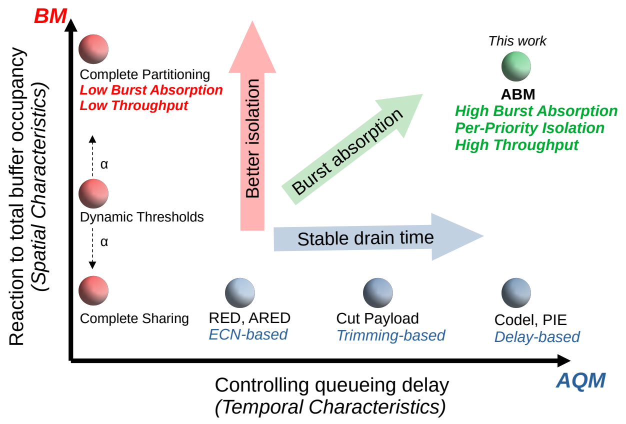


Figure 1: BM and AQM are orthogonal in their goals, and the hierarchical scheme fundamentally limits the burst absorption capabilities of the buffer.

1 INTRODUCTION

Network devices are equipped with a buffer to avoid drops during transient congestion and to absorb bursts. To reduce cost and maximize utilization, the on-chip buffer is shared across its queues. This sharing naturally leads to various problems. Concretely, the excessive growth of a queue might harm the performance of another queue, which might be starved, deprived of throughput, etc. Worse yet, such harmful interference might occur across queues that are seemingly independent e.g., queues that are mapped to different ports or queues that are formed by independent applications.

Network devices typically employ a hierarchical packet admission control to orchestrate the use of the shared space. First, a Buffer Management (BM) algorithm [16, 22, 33] dynamically splits the buffer space across queues. Second, an Active Queue Management (AQM) algorithm [24, 32, 42] manages the buffer slice that BM allocates to each individual queue by selectively admitting the incoming packets. Historically, BM and AQM evolved independently with orthogonal goals. We visualize this in Figure 1. BM aims at achieving *isolation* across queues by managing the *spatial* allocation of the buffer at the device level. Intuitively, BM's goal is to avoid long-lived queue starvation, effectively enforcing fairness in the steady state. For instance, Dynamic Thresholds [22] aims at weighted fairness across multiple queues in a device. In contrast, AQM's goal is to maintain stable queuing delays by managing the *temporal* allocation of the buffer at the queue level. Intuitively, AQM prevents bufferbloat by avoiding packets stay in the buffer for "too long" [24, 32, 42]. For instance, ECN-based AQM such as RED [24] control the queue lengths via ECN marking; delay-based

AQM such as Codel [32, 42] control the queueing delays at a fixed reference value.

While this decoupling has been reasonable and successful in the past as it allowed BM and AQM schemes to evolve further, two recent datacenter trends make the need for coordination between them pressing. First, buffer size is not keeping up with the increase in switch capacity [20, 27]. In effect, BM no longer has enough buffer available to provide isolation to each queue. Second, as traffic becomes more bursty and incast scenarios more prevalent [18, 43, 50], the transient state of the buffer needs to be controlled at the device-level [20]. To keep up with these trends, a buffer-sharing scheme needs to provide isolation, bounded drain time and high burst tolerance.

In this paper, we show that today's BM and AQM schemes are fundamentally unable to independently satisfy these requirements. Driven by this insight, we propose ABM, an Active Buffer Management algorithm that incorporates the insights from both BM and AQM to achieve high burst absorption without sacrificing throughput. Concretely, ABM leverages both total buffer occupancy at the device level and the individual queue drain time. Essentially, ABM is a function of both spatial (used by BM) and temporal (used by AQM) characteristics of the buffer, effectively providing the best of both worlds as shown in Figure 1. We analytically show that unlike state-of-the-art, ABM achieves strong isolation properties and maintains stable buffer drain time. This allows ABM to provide high and predictable burst absorption while achieving high throughput. We consider ABM practical, as it operates using statistics that are already used by BM or AQM algorithms, thus ABM is well within the capabilities of today's devices.

Our results from large-scale simulations show that ABM improves the flow completion times for short flows by up to 94% compared to existing BM and AQM schemes. Moreover, ABM is not only compatible with advanced congestion control algorithms (e.g., TIMELY, DCTCP and PowerTCP) but improves their performance in terms of tail FCTs by up to 76% under bursty workloads. Finally, we show that unlike traditional buffer management schemes, ABM works well on various buffer sizes, including shallow buffers (e.g., Tomahawk [1, 3]).

We view our work as the beginning towards a new class of ABM algorithms which react to both total buffer occupancy and the queuing delay.

In summary, our contributions in this paper are:

- We reveal the fundamental limitations of BM and AQM schemes that prevent optimally absorbing bursts (§ 2.2).
- We analytically show the critical limitations of the state-of-the-art buffer management scheme (§ 2.3).
- We design Active Buffer Management (ABM), an algorithm that achieves high burst absorption and maintains high throughput by leveraging both total buffer occupancy and queue drain time (§ 3).
- An extensive evaluation that demonstrates the benefits of ABM in the datacenter context (§ 4).
- As a contribution to the research community, to ensure reproducibility and facilitate future work, we made all of our artifacts publicly available at https://github.com/inet-tub/ns3-datacenter.

This work does not raise any ethical issues.

2 MOTIVATION

In this section, we make a case for cooperation between Buffer Management (BM) and Active Queue Management (AQM) to reap the best out of the precious but limited on-chip buffer space. Hereafter, we say **buffer-sharing** scheme to refer to any scheme within the two dimensions of BM and AQM as shown in Figure 1. After describing our model, we explain the desirable properties of a buffer-sharing scheme (§ 2.1). We then discuss the limitations of existing approaches (§ 2.2). Finally, we analytically reveal the pitfalls of the state-of-the-art BM scheme, namely Dynamic Thresholds (§ 2.3).

Model: We consider an output-queued shared-memory packet switching chip. Two schemes affect the allocation of the shared buffer. First, a BM scheme dynamically decides the maximum length of each queue. Second, an AQM scheme decides whether an incoming packet will be enqueued, marked, trimmed, or dropped. We consider that packets are grouped into a small set of priorities $\mathcal{P}$. Each priority exclusively uses a separate queue at each port.

2.1 Desirable Properties

To maximize the benefits of the shared buffer, a buffer-sharing scheme needs to satisfy three key properties: *(i)* isolation; *(ii)* bounded drain time; and *(iii)* predictable burst tolerance. Next, we describe these properties and motivate them through intuitive examples.

2.1.1 *Isolation*

As the buffer is shared across multiple queues, the excessive use of buffer by a small set of queues in aggregate might interfere with the ability of other queues in the switch to use the shared buffer. Such interference can be particularly harmful if the competing queues belong to different traffic priorities. As an intuition, consider the case illustrated in Figure 2. A queue serving a traffic priority that is particularly critical to the operator (e.g., loss-sensitive traffic) starves because the buffer is occupied by queues formed on other ports and even by best-effort traffic. To avoid such harmful interference across queues, different traffic priorities must be isolated. Concretely, we require that each priority must be allowed to occupy a configurable minimum amount of buffer at any given time, independently of the buffer state.

Formally, let $T_p(t)$ denote the total allocated buffer to a priority $p \in \mathcal{P}$ in the set of all priorities $\mathcal{P}$ at any time t. Then $T_p(t)$ must always be greater than B_p^{min}, a configurable static value. However, since the total buffer space, B is limited, it is necessary that the total allocation is within B. Thus, each priority must also be upper bound in its allocation (B_p^{max}) to prevent monopolizing the buffer.

$$\text{Isolation (Minimum Guarantee):} \qquad T_p(t) \geq B_p^{min} \qquad (1)$$

$$\text{Isolation (Preventing Monopoly):} \qquad T_p(t) \leq B_p^{max} \qquad (2)$$

2.1.2 *Bounded drain time*

Queueing delays are the root cause of high flow completion times for short flows in a datacenter [14, 36, 39]. Hence, various schemes including ECN-based AQM aim at reducing the queue lengths. Indeed, queuing delay also affects the buffer's drain time, namely, how fast the occupied buffer can be made available for incoming traffic (e.g., a burst or an incast). Importantly, though, queueing delay is not equivalent to a queue's length. In fact, two queues with

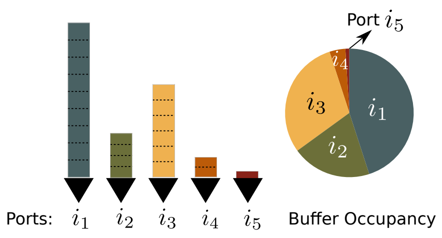


Figure 2: The queue at port i_5 is starved (deprived of buffer) due to other queues that are using the buffer.

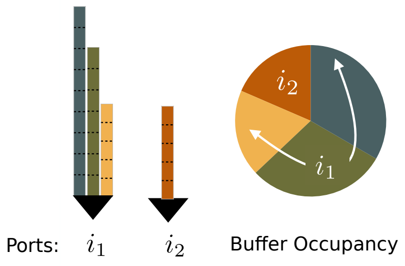


Figure 3: While one of the queues at port i_1 has the same length as the queue at port i_2, it has 3x queuing delay.

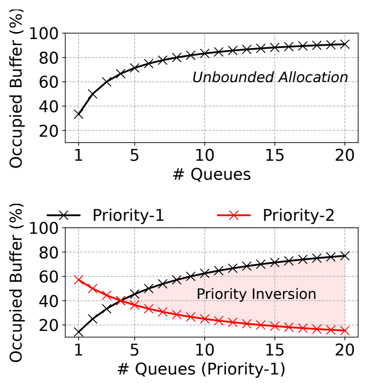


Figure 4: As the number of congested queues increases (Equation 6), DT allocates a unboundedly large amount of buffer (top) and cannot address isolation (bottom).

the same queue length might experience different queueing delays. This is possible as a variable number of queues can share the bandwidth of a single port. Consider the example shown in Figure 3: a queue in port i_1 experiences 3x queueing delay compared to the queue of equal length in port i_2. This is the case as the former queue can use $\frac{1}{3}$ of the port's bandwidth while the latter can use the full bandwidth of the port. Even within the same port, queues of equal length might experience different queueing delay depending on the underlying scheduling algorithm (e.g., weighted round-robin).

To avoid the harmful consequences of high queuing delays, a buffer-sharing scheme needs to bound the per-queue drain time. Concretely, a buffer-sharing scheme needs to bound the occupied buffer $q(t)$ of a queue with service rate $\mu(t)$ by a configurable static value Γ. In fact, the following condition summarizes the bufferbloat problem [26, 32, 42]: the desired property is to avoid packets staying "too long" in the buffer.

$$\textit{Bounded drain time:} \quad \frac{q(t)}{\mu(t)} \leq \Gamma \tag{3}$$

2.1.3 *Predictable burst tolerance*

Incast and bursty traffic are key challenges in a datacenter. Dropping packets of a burst results in costly timeouts and is thus undesirable. We define burst tolerance as the maximum burst of packets that the buffer-sharing scheme can store in the buffer until they can be transmitted via the corresponding port.

Intuitively, a buffer-sharing scheme can absorb an incoming burst either if there is burst-size amount of empty buffer available upon arrival (relates to isolation) or if the occupied buffer can drain fast enough to accommodate the burst (relates to drain time). While these conditions are sufficient to absorb a burst, they could have adverse effects. On the one hand, maintaining burst-size amount of empty buffer at all times deprives queues of the precious buffer, resulting in potential loss of throughput. On the other hand, allocating all newly released buffer (from draining queues) to an incoming burst will starve the draining queues (drop each of their incoming packets). In conclusion, to provide predictable burst tolerance while avoiding buffer waste, a buffer-sharing scheme needs a combination of both isolation and bounded drain time.

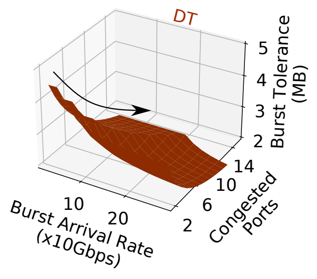
2.2 Limitations of Existing Approaches

Next, we show that the existing approaches in BM and AQM cannot satisfy the desirable properties of a buffer-sharing scheme discussed in § 2.1. We focus on BM and AQM as these "actively" control the worst-case (maximum) queue lengths.

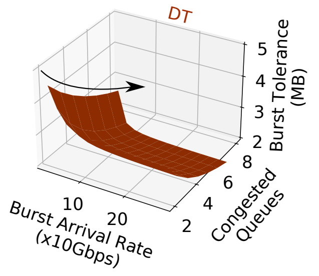
We first generalize the steady-state workings of both BM and AQM using a fluid-flow model. We are in steady state when the load conditions and buffer occupancy remain unchanged. We define the threshold of a queue as the value of queue length after which incoming packets are not admitted in the buffer. Thus, the steady-state of both BM and AQM can be generalized as shown in Equation 4, where T_p^i is the steady-state length of a queue at port i and of priority p. In the following, Ψ is a function that any BM scheme uses to calculate the per-queue thresholds based on buffer-wide metrics i.e., the instantaneous total buffer occupancy Q, while Φ is a function that any AQM scheme uses to calculate the per-queue threshold based on per-queue metrics e.g., length q and queue drain rate μ_p^i. The effective threshold per queue T_p^i is the minimum between the two thresholds,

$$T_p^i = \min\left(\underbrace{\Psi_p^i(Q)}_{BM} , \underbrace{\Phi(q, \mu_p^i)}_{AQM} \right) \tag{4}$$

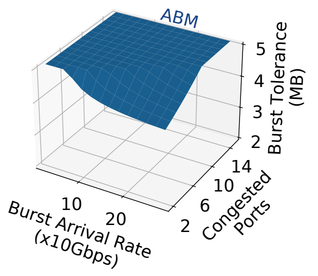
Limitations of Active Queue Management (AQM): AQM schemes control the queue lengths/delay at a fixed reference value. For instance, RED [24] queues used in DCTCP [14] would set $\Phi = K$, where K is the marking threshold (a constant). Delay-based AQM such as PIE [42] queues would set $\Phi = K \cdot \frac{\mu_p^i}{b}$ where μ_p^i is the drain rate of the queue and b is the port bandwidth; $\frac{K}{b}$ is the delay reference value. In principle, delay-based AQM schemes which set $\Phi = K \cdot \frac{\mu_p^i}{b}$ can maintain constant queue drain time. As discussed



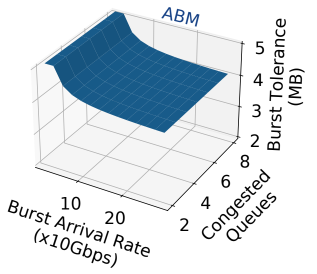
(a) DT's burst tolerance reduces with the number of congested ports due to its unbounded buffer allocation.



(b) DT's burst tolerance reduces with the number of congested per-port queues due to its obliviousness to drain rate.



(c) ABM offers predictable burst tolerance by bounded allocation regardless of the number of congested ports.



(d) ABM offers predictable burst tolerance by dynamically adapting allocations to the buffer drain time (number of queues per port).

Figure 5: DT's burst tolerance is unpredictable as it depends on unpredictable factors. On the contrary, ABM's burst tolerance remains high even at times of high load.

earlier in § 2.1, bounded drain time helps in absorbing bursts, since the occupied buffer can rapidly react to accommodate the incoming burst. However, the total shared buffer occupancy with AQM controlled queues is $n \cdot \Phi$ where n is the total number of queues using the buffer. AQM has no visibility over other queues using the buffer and thus cannot bound the overall buffer occupancy. As a result, AQM cannot satisfy the isolation property we describe in § 2.1.

Takeaway. *While AQM can, in principle, guarantee bounded drain time to help with burst tolerance, AQM cannot fundamentally satisfy the isolation property due to its inability to control the shared buffer.*

Limitations of Buffer Management (BM): Buffer Management schemes assign thresholds to every queue on a device. Thus, a BM scheme can, in principle, achieve isolation across queues. For instance, complete partitioning sets $\Psi_p^i(Q) = \frac{B}{N}$ where N is the total number of queues in the buffer. This naturally isolates each queue, but at the cost of extremely low buffer utilization. More dynamic schemes, such as Dynamic Thresholds (DT) [22] improve buffer utilization but sacrifice isolation. We analytically show this and other limitations of DT in § 2.3. Further, BM schemes are oblivious to the drain time, since they only react to total buffer occupancy i.e., BM schemes cannot ensure that packets leave the buffer "fast enough". As a result, BM schemes cannot satisfy the drain time and burst-tolerance property.

Takeaway. *BM schemes can, in principle, achieve isolation but are fundamentally limited in burst tolerance as a result of being oblivious to buffer drain time.*

2.3 Drawbacks of the State-of-the-Art Buffer Management Scheme

In this section, we shed light on the important drawbacks of the state-of-the-art buffer management algorithm used in today's datacenter switches, namely Dynamic Thresholds (DT) [22]. DT is the most common buffer management today [8, 16, 29, 37, 41]. We first explain how DT calculates its thresholds. Next, we describe why it is fundamentally unable to achieve the desirable properties described in § 2.1.

DT's workings: DT dynamically adapts the instantaneous maximum length of each queue of priority p, namely its threshold according to the remaining buffer and a configurable parameter α_p often configured per priority.[1] Formally, let B be the total shared buffer and $Q(t)$ be the total buffer occupancy at time t. DT calculates the threshold $T_p^i(t)$ for a queue of priority p at port i as follows:

$$T_p^i(t) = \alpha_p \cdot \underbrace{(B - Q(t))}_{\text{Total Remaining}} \tag{5}$$

The α parameter of a queue affects its maximum length relative to the other queues. An operator is likely to set higher (resp. lower) α values for high-priority (resp. low-priority) traffic. Different vendors and operators reportedly use different α values. For instance, Yahoo uses $\alpha = 8$ [29] while Cisco uses $\alpha = 14$ [8] and Arista $\alpha = 1$.

In the following, we analytically show that Dynamic Thresholds (DT) suffers from low and unpredictable burst tolerance. The key reasons for this limitation are: *(i)* the unbounded buffer allocation; and *(ii)* the obliviousness to the buffer drain time.

Unbounded buffer allocation: To explain why DT is fundamentally unable to bound its allocation, we walk through the workings of DT in steady-state. We refer to steady-state as the time during which the load conditions, the total buffer occupancy, and the thresholds remain stable (unchanged) i.e., $\dot{T}_p^i(t) = 0; \quad \dot{Q}(t) = 0$. The threshold calculated by DT for a queue of priority p at port i in steady-state is given by,

$$T_p^i(t) = \frac{\alpha_p \cdot B}{1 + \sum_{p \in \mathcal{P}} n_p \cdot \alpha_p} \tag{6}$$

where $\mathcal{P}$ is the set of priorities using the buffer and n_p is the number of congested queues of priority p. As we see in Equation 6, DT's threshold for a queue of priority p is dependent on the configurable α_p value and the number of congested queues of each priority using the buffer. As the number of congested queues n_p increases, the threshold decreases arbitrarily close to zero. In effect, DT cannot

[1]While α can be configured per queue, it is often configured per priority.

offer any minimum available buffer to any priority, i.e., cannot offer any isolation.

Figure 4, illustrates this effect. First, the remaining buffer tends to zero as the number of congested queues increases. Second, the threshold for a queue of certain priority (e.g., loss-sensitive AF) drops as the number of congested queues of another priority (e.g., best-effort BE) increases. Observe that this occurs even though the α_p value is higher for the loss-sensitive priority.

Unpredictable burst tolerance: Intuitively, DT's threshold calculation results in unpredictable burst tolerance for two reasons. First, DT's unbounded buffer allocation (which we described before) cannot predict the amount of buffer available for an incoming burst. Second, DT is oblivious to drain time, thus unaware of the rate at which the buffer can be made available for an incoming burst.

To formally verify this intuition, we analyze the state of the buffer throughout the arrival of a packet burst that arrives in a queue at port i, priority p and with a drain rate of μ_p^i. We describe the burst as incoming traffic with an arrival rate r. Before the burst arrives, given the number of congested ports, we denote the aggregate drain rate as μ. Upon the arrival of a burst at time $t = 0$, the buffer enters a transient-state: the total buffer occupancy, remaining buffer and consequently the thresholds change until they stabilize to a steady-state.

For time $t = 0^+$, the queue (hosting the burst) starts to grow, and its threshold changes. The change in the queue's threshold depends on the change in the total remaining buffer (see Equation 5). To study the rate of change of the threshold, we take its derivative with respect to time and obtain the following:

$$\dot{T}_p^i(t) = -\alpha_p \cdot \dot{Q}(t) \tag{7}$$

We integrate on both sides of Equation 7 in the interval $t = 0$ to a time τ when the queue hosting the burst reaches its threshold i.e., $T_p^i(t) = Q_p^i(t)$. We substitute the initial conditions for $t = 0$ from the steady-state occupancy (Equation 6) and solve for the time τ. In essence, the queue experiences zero loss until τ and the corresponding queue length at τ i.e., $(r - \mu_p^i) \cdot \tau$ indicates the burst tolerance of the buffer given the initial steady-state occupancy. Depending on the arrival rate r of the burst, we split into two cases.

First, if the arrival rate r is such that the threshold of each queue at $t = 0$ drops at a rate less than the corresponding queue's drain rate, then all the queues are able to drain according to the changes in the thresholds. In this case, the burst occupies the steady-state allocation corresponding to its queue (Equation 6). Notice that DT's burst tolerance even when the arrival rate r is low, critically depends on the number of congested queues of each priority due to its unbounded buffer allocation. Figure 5a illustrates the poor burst tolerance of DT across different burst arrival rates and initial steady-state conditions.

Second, if the arrival rate r is such that the threshold of each queue at $t = 0$ drops at a rate greater than the corresponding queue's drain rate, then the queues cannot keep up with the changes in the threshold and the burst experiences drops at time τ before reaching its steady-state allocation. In this case, we obtain the burst tolerance of DT $(r - \mu_p^i) \cdot \tau$ as follows:

$$\overbrace{(r - \mu_p^i) \cdot \tau}^{\text{Burst Tolerance}} = \frac{\alpha_p \cdot B}{\left(1 + \sum_{p \in \mathcal{P}} n_p \cdot \alpha_p\right) \cdot \left(1 + \alpha_p \cdot \frac{(r - \mu_p^i) - \mu}{r - \mu_p^i}\right)} \tag{8}$$

Observe that DT's burst tolerance critically depends not only on the number of congested queues n_p of each priority but also on the difference between the burst arrival rate r and the aggregate buffer drain rate μ. Even worse, DT's obliviousness to drain rate allows queues to increase in lengths irrespective of their drain time — effectively increasing n_p even with low aggregate drain rate μ. As a result, DT significantly suffers from low burst tolerance. Figure 5b illustrates the consequences of the obliviousness of drain time in buffer allocation to burst tolerance.

We have so far proved our intuition, namely that the burst tolerance depends on two factors *(i)* the total buffer occupancy at $t = 0$ in the steady-state (Equation 6) and *(ii)* the drain rate of the buffer in the transient-state (Equation 7).

3 ACTIVE BUFFER MANAGEMENT

Driven by our observations in § 2.3, we design a buffer-sharing scheme that systematically combines the insights of both BM and AQM while avoiding the pitfalls of existing schemes. Our goal is to satisfy the properties we identified in § 2.1:

- Provide isolation ▹ Theorem 1, 2
- Maintain bounded drain time ▹ Theorem 3
- Achieve predictable burst tolerance ▹ Equation 11

3.1 The ABM Algorithm

ABM assigns thresholds to each queue considering both spatial, buffer-wide and temporal, per-queue statistics. Formally, ABM assigns a threshold $T_p^i(t)$ to a queue of priority p at port i according to Equation 9 i.e., using a configurable value α_p, the port's bandwidth b and three dynamically changing factors: *(i)* the number of congested queues of priority p (n_p) that contributes to isolation property; *(ii)* the drain rate of the queue (μ_p^i) that contributes to maintain bounded drain time property; and the remaining buffer space ($B - Q(t)$).

$$T_p^i(t) = \overbrace{\alpha_p \cdot \frac{1}{n_p} \cdot (B - Q(t))}^{Buffer\ Management} \cdot \overbrace{\frac{\mu_p^i}{b}}^{AQM} \tag{9}$$

$\boxed{\alpha_p}$ is the only parameter the operator needs to configure in ABM. Similarly to DT, a higher α_p value in ABM results in a higher allocation *on average*. Unlike DT though, the α_p defines the minimum and maximum buffer available to each priority, as we show in § 3.2

$\boxed{n_p}$ denotes the number of congested queues of priority p. ABM considers a queue congested if the queue length is close to the corresponding threshold. In our evaluation, we consider a queue as congested if its length is greater than or equal to 0.9 of its threshold.

$\boxed{\frac{\mu_p^i}{b}}$ denotes the normalized drain rate, where μ_p^i is the drain rate of the queue and b is the bandwidth per port.[2] In effect, $\frac{\mu_p^i}{b}$ is the portion of the port's bandwidth that is available to the particular queue. In our analysis, we assume that the drain rate μ_p^i is a continuous value that changes according to the offered load and the scheduling policy. For instance, each of four queues that are mapped to the same port will have $\frac{\mu_p^i}{b} = 0.25$ if the scheduling is Round Robin. To ensure practicality, in our evaluations, we measure μ_p^i periodically and use the measurement for the threshold calculation. We also perform a sensitivity analysis on the periodic update intervals.

$\boxed{(B - Q(t))}$ denotes the unused and unreserved buffer space. Importantly, this factor is identical to what DT uses.

3.2 ABM's Properties

We now show how ABM satisfies the desirable properties, namely isolation, bounded drain time, and predictable burst absorption. We also provide formal arguments for these properties, deferring the full proof to the Appendix A.

ABM offers isolation across priorities. Concretely, ABM *(i)* bounds the total buffer occupied by each priority; and *(ii)* offers minimum buffer guarantees to each priority. In effect, no priority can monopolize the buffer and starve others. ABM achieves this by considering the number of congested queues per priority when calculating the per-queue thresholds. As an intuition, the per-queue threshold of a given priority decreases as more queues of that priority are congested.

Concretely, Theorem 1 addresses the isolation property of minimum guarantee, while Theorem 2 addresses the isolation property of preventing monopoly. We formally define and discuss the need for those properties in § 2.1.

Theorem 1 (Isolation - Minimum guarantee). *The total amount of buffer available for any priority p is lower bounded by B_p^{min} given by,*

$$B_p^{min} \geq \frac{B \cdot \alpha_p}{1 + \sum_{p \in \mathcal{P}} \alpha_p}$$

Theorem 2 (Isolation - Preventing monopoly). *The total amount of buffer available for any priority p is upper bounded by B_p^{max} given by,*

$$B_p^{max} \leq \frac{B \cdot \alpha_p}{1 + \alpha_p}$$

Notably, both bounds depend only on the α_p parameter which the operator configures, and not on the instantaneous state of the buffer as in DT.

ABM bounds drain time by allocating buffer space proportionately to the drain rate of each queue. In effect, ABM bounds queuing delay and total buffer drain time irrespective of the number of congested queues in a port, or the scheduling policy. We formally express the drain time properties of ABM in Theorem 3.

Theorem 3 (Bounded drain time). *The thresholds assigned by ABM upper bounds the drain time Γ for any queue of priority p given by,*

$$\Gamma \leq \frac{B \cdot \alpha_p}{(1 + \alpha_p) \cdot b}$$

Notably, this bound is only dependent on the α_p which the operator configures, and b which is the switch port bandwidth which is static at run-time.

Intuition for proof of Theorem 1, 2, 3: We first define Ω (formally defined in Definition 1) as $\Omega_p^i = \alpha_p \cdot \frac{1}{n_p} \cdot \frac{\mu_p^i}{b}$. Notice that ABM's allocation scheme (Equation 9) is essentially $T_p^i(t) = \Omega_p^i \cdot (B - Q(t))$. We can then view Ω as an adaptive α parameter according to DT's allocation (Equation 5)[3]. We derive an important property that the sum of Ω_p^i values of all congested queues of a certain priority p is upper bounded by α_p (Lemma 1). The buffer allocation for a priority p in steady state turns out to be $\frac{\sum_i \Omega_p^i \cdot B}{1 + \sum_i \sum_{p \in \mathcal{P}} \Omega_p^i}$ where $\mathcal{P}$ is the set of priorities using buffer. Based on the property of Ω, we then derive lower and upper bounds (Theorem 1, 2) for the buffer allocated to a certain priority. Similarly, using the upper bound for buffer allocation and dividing by the drain rate μ_i^p, we bound the drain time in Theorem 3.

ABM's allocation scheme in Equation 9 has its roots in the properties of Ω that offer isolation and bounded drain time properties. For full proofs, we refer the interested reader to our complete analysis in Appendix A.

ABM offers predictable burst tolerance thanks to the previous properties. Intuitively, ABM's bounded allocation makes it ready to absorb small bursts and the stable drain time property further enhances ABM's burst tolerance as the occupied buffer can readily react to incoming bursts. To formalize ABM's burst tolerance, we analyze the state of the buffer throughout the arrival of a packet burst that arrives on a queue at port i, priority p and with a drain rate of μ_p^i. Similar to our analysis of DT in § 2.3, we describe the burst as incoming traffic with an arrival rate r. Upon the arrival of a burst at time $t = 0$, the buffer enters a transient state.

First, if the arrival rate r is such that the threshold of each queue at $t = 0$ reduces at a rate less than the corresponding queue's drain rate, then the aggregate buffer drains according to the changes in the thresholds and the burst occupies its steady-state allocation as follows:

$$\overbrace{(r - \mu_p^i) \cdot \tau}^{\text{Burst Tolerance}} = \frac{\alpha_p \cdot \frac{1}{n_p} \cdot B \cdot \frac{\mu_p^i}{b}}{1 + \sum_{p \in \mathcal{P}} \alpha_p} \tag{10}$$

Notice that ABM's burst tolerance is independent of the number of the congested queues of *other* priorities. Rather, the burst tolerance only reduces due the number of congested queues of the *same* priority.

Second, if the arrival rate r is such that the threshold of each queue at $t = 0$ reduces at a rate greater than the corresponding queue's drain rate, then the aggregate buffer cannot drain according to the changes in the thresholds. In this case, we obtain ABM's burst tolerance given by Equation 11, where μ is the aggregate drain rate of the buffer at time $t = 0$.

[2]If the switch ports are not symmetric in bandwidth, b is the bandwidth of the port with the highest bandwidth.

[3]Setting $\Omega_p^i = \alpha_p$ reduces to DT's allocation scheme.

$$\overbrace{(r-\mu_p^i)\cdot\tau}^{\text{Burst Tolerance}} = \frac{\alpha_p\cdot\frac{1}{n_p}\cdot B\cdot\frac{\mu_p^i}{b}}{\left(1+\sum_{p\in\mathcal{P}}\alpha_p\right)\cdot\left(1+\alpha_p\cdot\frac{1}{n_p}\cdot\frac{\mu_p^i}{b}\cdot\frac{(r-\mu_p^i)-\mu}{r-\mu_p^i}\right)} \tag{11}$$

Finally, notice from Equation 11 that ABM's burst tolerance on a queue of a certain priority remains independent of the number of congested queues of other priorities. Further, unlike DT, ABM accounts for the buffer drain time and achieves a high burst tolerance even with low aggregate drain rate. In essence, ABM's burst tolerance is a function of the burst arrival rate r. Observe from Equation 11 that the burst tolerance of a certain priority still depends on the number of congested queues of the *same* priority. The buffer occupancy of a priority self-inflicts its own burst tolerance. In § 3.3, we further optimize ABM's thresholds to prevent this effect.

Figure 5 illustrates ABM's burst tolerance properties under various buffer states. Observe that ABM's burst tolerance for a priority remains high and only depends on *rate* at which the burst arrives regardless of the number of ports that are congested (Figure 5c) and of the number of congested queues per port (Figure 5d). Recall that DT's burst tolerance depends on both these factors and is thus unpredictable as we observe in Figure 5a, 5b.

3.3 Optimizing for Datacenter Workloads

As mentioned above, although ABM alleviates the dependency on other priorities in its burst tolerance of a given priority, the buffer occupancy of each priority self-inflicts its own burst tolerance. To prevent this effect, we further optimize ABM's thresholds to maximize its burst tolerance.

To this end, ABM prioritizes all unscheduled packets by using a higher α_p value in allocating buffer for two reasons. First, bursty traffic in a datacenter originates mainly from unscheduled (first RTT) packets of a flow, since congestion control cannot fundamentally act within the first RTT. Second, short flows that finish within the first RTT are of utmost importance in a datacenter. Even a single packet loss could lead to costly timeouts and long flow completion times.

Specifically, even the unscheduled packets are destined to a specific queue at each port based on their default traffic priority. However, ABM uses a higher α_p value in its thresholds while admitting such packets to the buffer. We assume that unscheduled packets arrive with a tag attached by the end-hosts. Observe that such a tag can also be dynamically obtained if the switch is programmable. In essence, by prioritizing unscheduled packets, ABM prevents the self-inflicting effect as seen in Equation 11 i.e., a higher α_p for unscheduled packets diminishes the effect of self-inflicting n_p factor. Note that unscheduled packets relate to the transient state of the buffer, and prioritizing such packets does not affect the steady-state properties of ABM. As a result, ABM maximizes the burst tolerance properties and remains independent of the number of congested ports, as well as the number of congested queues at each port.

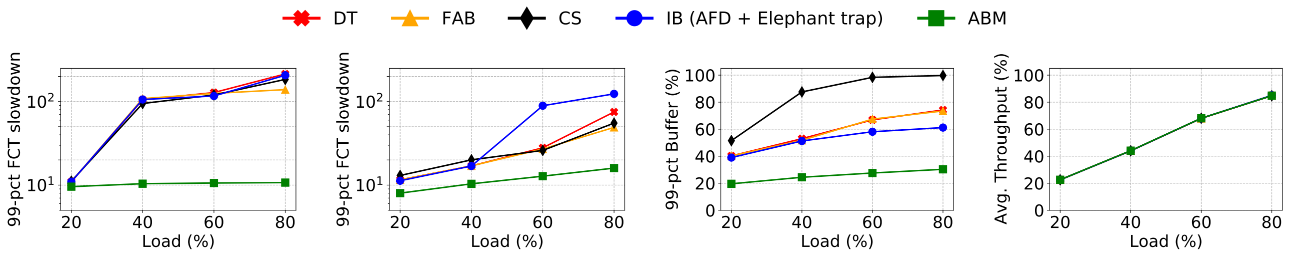
3.4 ABM's Practical Considerations

ABM is attractive in practice for three reasons.

ABM uses statistics that are available to today's switches. ABM cannot be easily implemented today, as the MMU is not programmable even on programmable devices [47]. Still, it is important to note that ABM only uses statistics that are used either by BM or by AQM schemes, thus is not fundamentally impossible. $\boxed{\alpha_p}$ and $\boxed{(B-Q(t))}$ are used by DT [22] which is implemented on most datacenter switches [8, 29, 37, 41]. $\boxed{n_p}$, i.e., the number of congested queues of priority p only requires visibility over queue lengths, which is provided to both AQM and BM schemes to decide whether a packet can to be admitted. $\boxed{\frac{\mu_p^i}{b}}$ depends on the port's bandwidth, the scheduling algorithm, and the number of congested queues mapped to the same port. The two former do not change. Thus, if the number of congested queues mapped to the same port is static, then $\frac{\mu_p^i}{b}$ also becomes static. If the number of congested queues (i.e., exceeding a threshold) mapped to the same port changes over time, then we only need this number to calculate the rate. For example, if scheduling is round-robin and there are two congested queues in a port, then the normalized drain rate is $\frac{\mu_p^i}{b} = 0.5$. Further, several congestion control algorithms implemented in real datacenters already use in-band telemetry and insert μ_p^i in packet headers (e.g., HPCC [35]). Finally, ABM's threshold also requires a floating point operation similar to DT which calculates $\alpha_p\cdot(B-Q(t))$.

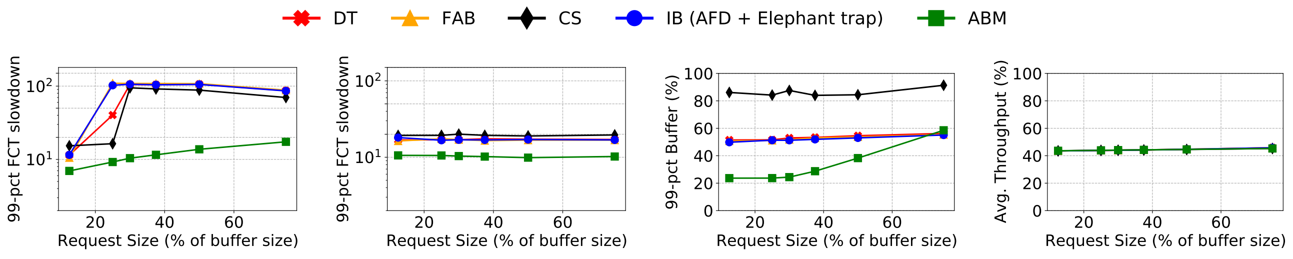
ABM teaches essential lessons on how to configure α values. Although ABM and DT have major differences in their properties, their thresholds are in fact similar (cf. Equations 6, vs 9). Thus, ABM's insights can help an operator configure DT. Concretely, the operator could divide α_p values to the number of queues mapped to the same port or to the number of congested ports they expect to have. Finally, an operator can use our mathematical analysis (also illustrated in Fig. 5) to find an approximation of the burst tolerance of their DT configuration leveraging their insights about the usual state of the buffer (i.e., number of congested queues per port and the number of congested ports).

One can approximate ABM on top of DT. As ABM's threshold (Equation 9) is so similar to DT's formula (Equation 5) an operator could approximate ABM using DT. To that end, the operator would need to implement a control-plane function that periodically pulls queue statistics and reconfigures α_p per queue according to ABM 's thresholds. Observe that most vendors today expose queue statistics that are required to calculate $\frac{\mu_p^i}{b}$ and n_p i.e., queue lengths [2, 6, 7, 11]. How close this approximation would be to ABM depends on two factors *(i)* how frequently a device can be reconfigured; and *(ii)* how dynamic the traffic patterns are. In § 4.4, we evaluate the effect of such an approximation by varying the time intervals in which α_p values are updated. We find that the approximation performs similarly to ABM for small update intervals, but performs similarly to DT at high update intervals.



(a) Flows of incast traffic (b) Short flows of web-search (c) Buffer occupancy (d) Throughput

Figure 6: Buffer Management under various loads. ABM achieves lower tail FCT (a) for flows contributing to bursts (incast traffic) and (b) for short flows (web-search) compared to other BM schemes across various loads. In doing so ABM (c) uses less buffer; and (d) does not sacrifice throughput.



(a) Flows of incast traffic (b) Short flows of web-search (c) Buffer occupancy (d) Throughput

Figure 7: Buffer Management under various request sizes. ABM achieves lower tail FCT (a) for flows contributing to bursts (incast traffic) and (b) for short flows (web-search) compared to other BM schemes across various request sizes. (c) ABM's tail buffer occupancy increases with the request size, while (d) throughput stays untouched.

4 EVALUATION

We evaluate the performance of ABM and compare it with state-of-the-art approaches in the datacenter setting. Our evaluation aims at answering the following key questions:

(Q1) *How does ABM perform compared to other BMs in burst absorption and isolation?*

We find that ABM improves the 99th percentile FCT slowdown of the flows contributing to bursts by up to 94% (12.6%) under high (low) load compared to existing schemes. Moreover, we show that (unlike DT) ABM offers isolation across priorities as performance of a given priority is unaffected by the load of other priorities.

(Q2) *Does ABM sacrifice short flows or throughput?*

ABM does not sacrifice throughput or short flows' FCTs in exchange for burst absorption. In fact, ABM reduces the 99th percentile FCT slowdown for short flows by 28.3% on average even at 20% load with Cubic compared to existing BM schemes, while achieving on-par throughput.

(Q3) *Can ABM further improve FCTs of advanced congestion control schemes?*

We show that ABM improves the tail FCT slowdown under bursty workloads in DCTCP by 88%; in TIMELY by 33.3% and in PowerTCP by 2.13% even at moderate burst sizes.

(Q4) *How well does ABM perform in extremely shallow buffers?*

We show that ABM maintains its performance and improves the 99th percentile FCTs of bursty workloads by up to 92% compared to other BMs in extremely shallow buffers.

(Q5) *Can we reap any of ABM's benefits by approximating its allocation with DT?*

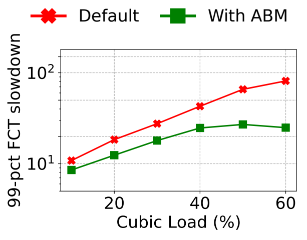
We perform a sensitivity analysis on the importance of the update interval in ABM's benefits. We find that one could benefit from ABM by re-configuring DT's α_p values at every ≈ 8 ms (100x RTT).

4.1 Setup

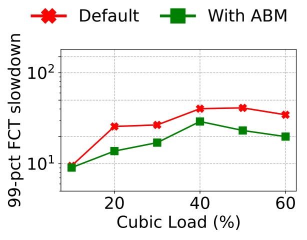
Our evaluation is based on network simulator NS3 [9].

Topology: We use a Leaf-Spine topology [13] with 8 spine switches and 256 servers organized into eight leaves. Each link has a capacity of 10Gbps (4:1 oversubscription) similar to prior work [12, 45]. Moreover, each link has 10μs propagation delay. Both leaf and spine switches have 9.6KB buffer-per-port-per-Gbps following the features of the Broadcom TridentII switch [10, 20].

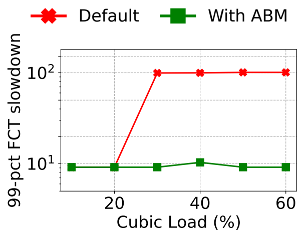
Workload: We generate traffic using two workloads *(i)* web-search; and *(ii)* incast traffic. First, we generate traffic following the web-search flow size distribution [14] which is based on real-world datacenter measurements across various loads in the range 20%-80%. Second, we generate incast traffic using a synthetic workload similar to prior work [12, 13]. Specifically, our incast traffic simulates a distributed file system query-response behavior in a datacenter. Each server in our topology requests a file from a set of servers



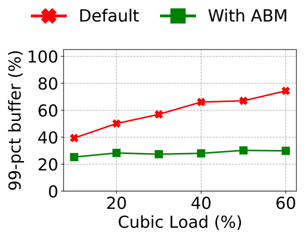
(a) Cubic flows



(b) DCTCP flows

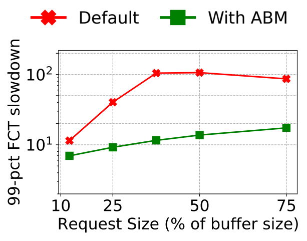


(c) θ-PowerTCP flows

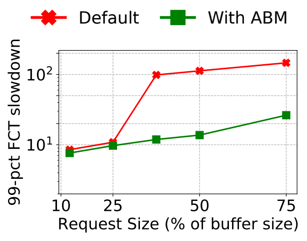


(d) Overall buffer occupancy

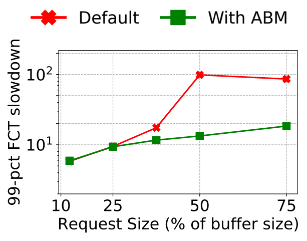
Figure 8: Cubic, DCTCP and θ-PowerTCP harm each other even though they use different queues with default BM (i.e., DT). ABM isolates each congestion control algorithm, effectively reducing their tail FCT.



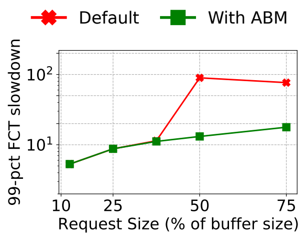
(a) Cubic flows



(b) DCTCP flows



(c) TIMELY flows



(d) PowerTCP flows

Figure 9: While advanced congestion control already offers low tail FCT, ABM allows for further improvements through better buffer allocations compared to the default buffer management (i.e., DT).

chosen uniformly at random from a different rack. All servers that receive the request respond by transmitting the requested part of the file. Each file request creates an incast. We evaluate across different request sizes from 10% -75% of the buffer.

Comparisons and metrics: We compare ABM with four alternative BM schemes: Dynamic Thresholds (DT) [22]; Flow Aware Buffer (FAB) [16], Complete Sharing (CS) and Cisco Intelligent Buffer (IB) [8]. DT allocates buffer proportionately to the remaining as we describe in § 2.3. FAB uses DT but prioritizes short flows. CS allows every queue in the switch to grow as long as there is remaining buffer in the shared space. IB uses Approximate Fair Dropping in combination with DT representing the typical hierarchical scheme.

We compare the performance of various congestion control algorithms with and without ABM. In particular, we evaluate ABM with Cubic [28] (loss-based), DCTCP [14] (ECN-based), TIMELY [38] (RTT-gradient-based), PowerTCP and θ-PowerTCP [12] (power-based). We report the switch total buffer occupancy, throughput and Flow Completion Time (FCT) slowdown i.e., the actual FCT divided by the ideal FCT with no other traffic in the network.

Parameter setting: We configure ABM, DT and FAB with $\alpha = 0.5$ for all the queues unless otherwise specified. We update n_p and μ_p^i for ABM once per RTT. We configure IB according to [4]. ABM uses $\alpha = 64$ for unscheduled packets (§ 3.3) and uses headroom similar to IB. We set $K = 65$ packets for DCTCP according to [14] and TIMELY parameters according to [38]. For PowerTCP and θ-PowerTCP, we set $baseRTT$ to the minimum RTT of the longest path; $NicBW$ is set to 10Gbps and the γ parameter is set according to [12]. We set $minRTO$ = 10ms.

4.2 ABM's Performance

ABM significantly improves incast-traffic FCTs: In Figure 6a, we show the 99th percentile FCT slowdown for the flows of incast workload with a fixed request size of 30% of the buffer size, as a function of the load created by the web-search workload. All flows use Cubic. We observe that at low load (20%) ABM reduces the 99th percentile FCT slowdown by 12.6% on average compared to DT, FAB, CS and IB. As the load increases, ABM outperforms alternatives in FCT slowdown by 90.12% at 40% load; and by 94% at 80% load.

In Figure 7a, we show the 99th percentile FCT slowdown for the flows of incast workload with fixed web-search load in 40% as a function of the request size of incast traffic. Even for a request size as small as 12.5% of the buffer size, ABM reduces the FCT slowdown by 39% on average compared to DT and IB; by 33% compared to FAB; and by 54% compared to CS. As the request size increases, ABM's benefits are more pronounced. At a request size of 50% of the buffer size, ABM reduces the FCT slowdown of incast workload by at least 75.1% on average compared to DT, FAB, CS and IB.

In essence, ABM's bounded buffer allocation (Theorem 2) effectively limits the used buffer by the web-search workload and provides enough headroom for the incast workload. Further, ABM's prioritization of unscheduled packets (incast traffic) fully exploits the available headroom.

ABM effectively isolates traffic priorities: To evaluate ABM's performance isolation across different priorities compared to the default, namely DT, we consider the following scenario. The network is shared across three priorities each using a different transport protocol among Cubic, DCTCP, and θ-PowerTCP. Each priority uses a distinct queue at each port. The most recent protocol, namely θ-PowerTCP is used for the incast workload, while Cubic and DCTCP serve web-search traffic. Figure 8 (a,b,c) shows the 99th percentile FCT slowdown of short flows belonging to each priority as a function of the load carried by Cubic.

At a high level, Figure 8 shows the inability of separated queues and DT to offer isolation. As the Cubic load increases, the default (i.e., DT) FCT slowdown performance of DCTCP, and θ-PowerTCP significantly increases. In contrast, ABM effectively protects θ-PowerTCP and DCTCP from the Cubic load and reduces the tail FCT slowdown by up to 90.92%. Concretely, we observe that when ABM manages the buffer the FCT slowdown of both Cubic and DCTCP stabilizes around 20 (i.e., stops degrading with increasing Cubic load) and θ-PowerTCP stabilizes at 10 despite the increasing Cubic load.

In essence, as ABM bounds the buffer drain time (Theorem 3) and the buffer occupancy of each traffic priority (Theorem 2), it reduces the impact of Cubic traffic in other priorities of the shared buffer.

ABM improves short flows FCTs (even if they are not part of a burst): ABM not only improves incast workload FCTs, but also improves FCTs of all short flows. In Figure 6b, we show the 99th percentile FCT slowdown of short flows belonging to the web-search traffic as a function of the load. We observe that even at 20% load, ABM reduces the FCT slowdown by 30.7% compared to DT and FAB; by 38.5% compared to CS; and by 28.9% compared to IB. At 80% load, ABM reduces the FCT slowdown for short flows by 76% on average. Similarly, Figure 7b we show the 99th percentile FCT slowdown of short flows belonging to the web-search traffic as a function of request size of the incast workload and at a fixed load of 40% from the web-search workload. We observe that ABM reduces the FCT slowdown for short flows of web-search workload by 41.8% on average.

ABM does not sacrifice throughput to serve short flows or bursts: We further evaluate the performance of ABM in terms of throughput. In Figure 6d, we observe that ABM achieves on-par throughput compared to alternative approaches. Even under large request sizes of the incast workload in Figure 7d, ABM does not sacrifice throughput for low FCTs of short flows.

ABM strategically increases buffer utilization to accommodate incast: ABM effectively limits the buffer occupied by medium and long flows. Even with a buffer-hungry transport algorithm such as Cubic, shown in Figure 6c, ABM reduces the buffer usage across various loads by 54.2% on average compared to DT, FAB and IB; and by 68.9% compared to CS while still achieving on-par throughput. At the same time, ABM strategically uses more buffer to accommodate bursts. Figure 7c shows that ABM uses more buffer as the request size (incast workload) increases. Indeed, ABM uses 5.3% more buffer compared to DT, FAB and IB for large request sizes.

ABM's benefits increase as the number of queues-per-port increases under stable load: To evaluate the impact of μ_p^i factor on ABM's thresholds, we generate *(i)* web-search workload at 40% load; and *(ii)* incast with request size at 25% of the buffer size and vary the number of queues per port across which the load is distributed. Figure 10a shows the tail FCT slowdown of Cubic and DCTCP when combined with ABM and DT. We observe that as the number of queues per port increases, ABM shows higher benefits even with stable load conditions. Concretely, ABM and DT perform similarly to DCTCP until 4 queues per port, where ABM only improves over DT by 10%. At 5 queues per port, ABM reduces the 99th percentile FCT slowdown of Cubic and DCTCP by 88.9% on average. Furthermore, at 8 queues per port, ABM reduces the FCT slowdown of Cubic and DCTCP by 92.6%. Figure 10b shows how even DCTCP uses more buffer as the number of queues per port increases, even when the load is fixed. Figure 10 validates the importance of bounded drain time in § 2.1.

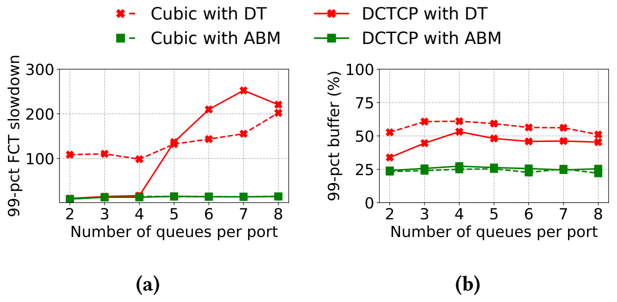


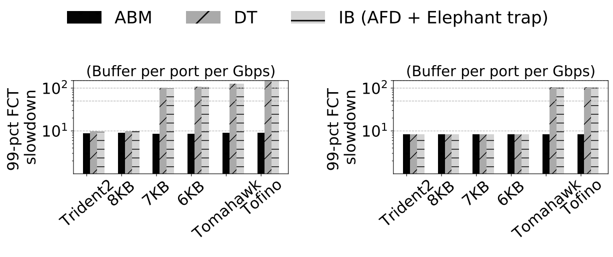
Figure 10: As the number of queues-per-port increases under stable load ABM's buffer occupancy remains low (b), similarly to its tail FCT (a).

ABM can improve FCTs of advanced congestion control algorithms under bursty workloads: Aiming at understanding the benefit of ABM in conjunction with advanced congestion control, we evaluated it under the following four scenarios. In each scenario one congestion control algorithm from Cubic, DCTCP, TIMELY or PowerTCP runs at the hosts generating web-search at 40% load and incast traffic at varying request sizes. We then compare the tail FCT slowdown achieved when ABM manages the buffer compared to when the default buffer management algorithm does. In Figure 9, we observe that for small-sized request at 12.5% of the buffer size, ABM reduces FCT slowdown by 39.1% compared to Cubic (with DT) and by 10.3% compared to DCTCP For this request size ABM achieves no improvements compared to TIMELY and PowerTCP. For medium-sized requests at 37.5% of the buffer size, ABM reduces the FCT slowdown by 88% compared to Cubic and DCTCP; reduces TIMELY's FCT slowdown by 33.3%. For this request size ABM achieves no improvements compared to PowerTCP. As the request size increases further, at 50% of the buffer size, ABM also reduces the 99th percentile FCT slowdown of PowerTCP by 76%.

Although advanced congestion control can improve the tail buffer occupancy and consequently the tail FCTs for short flows, we observe from Figure 9 that ABM can improve FCTs of advanced congestion control algorithms under bursty workloads.

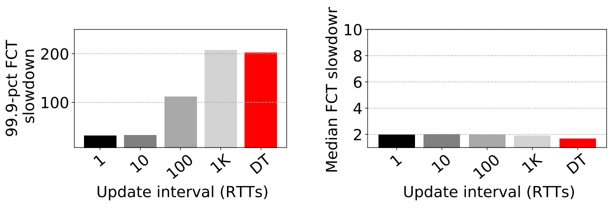
4.3 ABM's Performance in Shallow Buffers

Our evaluation setup so far considers a switch with 9.6KB buffer-per-port-per-Gbps that corresponds to a Trident2 switch. We further evaluate the benefits of ABM with smaller, yet realistic buffer sizes e.g., Tomahawk.



(a) DCTCP (b) PowerTCP

Figure 11: ABM maintains low tail FCT under various realistic buffer sizes that are smaller than Trident2 (which we use throughout § 4.2). On the contrary, DT and IB cause 10x tail FCT slowdown for multiple realistic buffer sizes.



(a) Short flows FCT (b) Long flows FCT

Figure 12: DT can approximate ABM's allocation if we reconfigure its α parameter every 100x RTT (i.e., every 8ms) according to ABM's formula. Such a coarse update interval will already improve DT's tail FCT by 44.78% compared to DT (red).

To that end, we vary the buffer-per-port-per-Gbps according to the specifications reported in [20]. We generate web-search workload at 40% load and incast workload at a request size at 25% of the buffer size corresponding to Trident2. We separately consider DCTCP and PowerTCP.

Figure 11, summarizes our results by showing the 99th percentile FCT slowdown of the incast flows for DCTCP (a) and PowerTCP (b). We observe that ABM's performance is robust to changes in buffer size even for extremely shallow buffers. Concretely, ABM achieves similar performance across various buffer sizes both with DCTCP and PowerTCP.

On the contrary, both DT and IB cannot effectively manage the buffer when its size is equal or smaller than 7KB per-port-per-Gbps. Indeed, at this buffer size the FCT slowdown of DT and IB with DCTCP increases by $\approx 10x$ compared to ABM with DCTCP as we observe in Figure 11a. PowerTCP hides DT's and IB's inability until 6KB per-port-per-Gbps but not further. Concretely, in Figure 11b, we observe that DT and IB cannot sustain low tail FCTs with 5.12KB per-port-per-Gbps (corresponding to Tomahawk). In summary, ABM effectively manages the buffer under bursty workloads even with a shallow buffer of 3.44KB (corresponding to Tofino). For instance, ABM reduces the 99th percentile FCT slowdown by 96% with DCTCP and by 92% with PowerTCP compared to alternatives i.e., DT and IB.

4.4 ABM's Performance with Periodic & Infrequent α Updates

In § 3.4, we introduced the possibility of approximating ABM on top of DT via periodically reconfiguring α at the control plane. We now aim at evaluating how sensitive ABM is to the update interval and what benefits such a practical approximation of ABM can have.

We generate traffic using web-search workload at 40% load and incast traffic with 75% request size. Flows are destined to one of the eight queues at each port chosen uniformly at random. We vary the update interval after which DT's α values are reconfigured to approximate ABM's allocation. We chose to have more queues per port compared to the previous experiment and a relatively large request size to make the scenario more challenging for ABM. Finally, in Figure 12a, we plot the 99.9th[4] percentile FCT slowdown for short flows and the median FCT slowdown for long flows for *(i)* ABM with varying update intervals; and *(ii)* DT (red bar).

At a high level, the 99.9th percentile FCT slowdown for short flows increases with larger update intervals, as the reconfiguration rate cannot keep up with the updates in n_p and μ_p^i used in ABM's thresholds. Thus, at around $1K$x RTT update interval, ABM's approximation is equivalent to DT for short flows FCT. Still, ABM's approximation significantly improves DT's performance for update intervals smaller than $1K$x RTT (i.e., 80ms) showing ABM's practical benefits. For instance, even with 10x RTT (100x RTT) update interval, the tail FCT slowdown of ABM decreases by 84.05% (44.78%) compared to DT. Importantly, the tail FCT of long flows is not affected by the intervals. Thus, the achieved throughput of ABM's approximation is on par with that of DT.

5 RELATED WORK

Optimally managing switch buffers has been an active area of research for more than two decades with a wide range of approaches, including BMs [16, 17, 19, 21–23, 33, 44] and AQMs [24, 32, 42, 49], scheduling [15, 30, 47] and end-host congestion control [12, 14, 35, 38–40, 48].

BM schemes such as FAB [16], Cisco's IB [5], TDT [31] and EDT [46] rely on DT [22] and attempt to give more of the remaining buffer to short flows. By relying on DT, these schemes inherit its pitfalls (§ 2.3).

AQM schemes such as RED [24], ARED [25], Codel [32] and PIE [42] control queue lengths or delays but under unrealistic assumptions. Indeed, AQM schemes assume per-queue buffer isolation i.e., that the maximum length per queue is static. Yet, in practice, this length dynamically changes, often depriving AQM from the required buffer to operate normally.

End-host congestion control algorithms have the potential to reduce the buffer requirements, but are orthogonal to our work. Control algorithms such as DCTCP [14], DCQCN [51] use marking schemes as feedback to adjust the sending rates. TIMELY [38] uses RTT-gradient approach to rapidly react to congestion onset. Even more advances algorithms use a variety of feedback signals

[4]We report 99.9th rather than 99th to stress ABM further.

e.g., HPCC [35] uses inflight bytes, Swift [34] uses delay and PowerTCP [12] uses the power. Yet, end-host congestion control algorithms cannot act on the first RTT packets, and are fundamentally unable to orchestrate the buffer-sharing at all times.

6 CONCLUSION

In this paper, we demonstrate the fundamental inability of Buffer Management and Active Queue Management schemes to address the challenges that occur from sharing the on-chip buffer across queues. Furthermore, we analytically show the limitations of the state-of-the-art buffer sharing scheme. We present ABM, a novel buffer-sharing scheme that offers isolation, bounded drain time, and high burst tolerance. ABM is practical in that it only uses statistics that are available to the MMU. We show that ABM outperforms all other buffer management schemes in tail FCT while achieving on-par throughput. Importantly, ABM improves the FCTs of advanced congestion control algorithms (i.e., TIMELY, DCTCP, and PowerTCP) under bursty workloads.

ACKNOWLEDGEMENTS

We would like to thank our shepherd, Michael Mitzenmacher, as well as the anonymous reviewers for their useful feedback. Research supported by the European Research Council (ERC), grant agreement No. 864228 (AdjustNet), Horizon 2020, 2020-2025, ARPA-E ENLITENED PINE, DARPA FastNICs, NSF grants CNS-2008624, SHF-2107244, ASCENT-2023468, CAREER-2144766, and Sloan fellowship.

REFERENCES

[1] Arista 7060CX-32 and 7260CX-64. https://people.ucsc.edu/~warner/Bufs/7060CX.html.

[2] Arista lanz overview. https://www.arista.com/assets/data/pdf/Whitepapers/Arista_LANZ_Overview_TechBulletin_0213.pdf.

[3] Braodcom Tomahawk. https://people.ucsc.edu/~warner/Bufs/tomahawk.

[4] Cisco nexus 9000 series switches. https://www.cisco.com/c/en/us/products/collateral/switches/nexus-9000-series-switches/white-paper-c11-738488.html.

[5] Cisco nexus 9000 series switches. https://www.cisco.com/c/en/us/products/collateral/switches/nexus-9000-series-switches/white-paper-c11-739134.html.

[6] Introduction to spectrum, the 100gbe switch silicon. https://www.cisco.com/c/en/us/support/docs/switches/nexus-3500-series-switches/118904-technote-nexus-00.html.

[7] Nexus 3500 output drops and buffer qos. https://community.mellanox.com/s/article/introduction-to-spectrum--the-100gbe-switch-silicon.

[8] Nexus 9000 architecture. https://www.ciscolive.com/c/dam/r/ciscolive/apjc/docs/2018/pdf/BRKDCT-3640.pdf.

[9] Ns3 network simulator. https://www.nsnam.org/.

[10] Trident2 / BCM56850 Series, High-Capacity StrataXGS® Trident II Ethernet Switch Series. https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm56850-series.

[11] Advanced Congestion and Flow Control with Programmable Switches, 2011. https://opennetworking.org/wp-content/uploads/2020/04/JK-Lee-Slide-Deck.pdf.

[12] Vamsi Addanki, Oliver Michel, and Stefan Schmid. PowerTCP: Pushing the performance limits of datacenter networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, April 2022.

[13] Mohammad Alizadeh and Tom Edsall. On the data path performance of leaf-spine datacenter fabrics. In *2013 IEEE 21st annual symposium on high-performance interconnects*, pages 71–74. IEEE, 2013.

[14] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). *ACM SIGCOMM computer communication review*, 41(4):63–74, 2011.

[15] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. pfabric: Minimal near-optimal datacenter transport. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 435–446. ACM, 2013.

[16] Maria Apostolaki, Laurent Vanbever, and Manya Ghobadi. Fab: Toward flow-aware buffer sharing on programmable switches. In *ACM Workshop on Buffer Sizing*, 2019.

[17] M. Arpaci and J. A. Copeland. Buffer management for shared-memory atm switches. *IEEE Communications Surveys Tutorials*, 3(1):2–10, First 2000.

[18] Chen Avin, Manya Ghobadi, Chen Griner, and Stefan Schmid. On the complexity of traffic traces and implications. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 4(1):1–29, 2020.

[19] James Aweya, Michel Ouellette, and Delfin Y Montuno. Buffer management scheme employing dynamic thresholds, September 7 2004. US Patent 6,788,697.

[20] Wei Bai, Shuihai Hu, Kai Chen, Kun Tan, and Yongqiang Xiong. One more config is enough: Saving (dc) tcp for high-speed extremely shallow-buffered datacenters. *IEEE/ACM Transactions on Networking*, 2020.

[21] Andreas V Bechtolsheim and David R Cheriton. Per-flow dynamic buffer management, February 4 2003. US Patent 6,515,963.

[22] Abhijit K Choudhury and Ellen L Hahne. Dynamic queue length thresholds for shared-memory packet switches. *IEEE/ACM Transactions On Networking*, 6(2):130–140, 1998.

[23] F. Ertemalp. Using dynamic buffer limiting to protect against belligerent flows in high-speed networks. In *Proceedings of the Ninth International Conference on Network Protocols*, ICNP '01, pages 230–, Washington, DC, USA, 2001. IEEE Computer Society.

[24] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, Aug 1993.

[25] Sally Floyd, Ramakrishna Gummadi, Scott Shenker, et al. Adaptive red: An algorithm for increasing the robustness of red's active queue management, 2001.

[26] Jim Gettys. Bufferbloat: Dark buffers in the internet. *IEEE Internet Computing*, 15(3):96–96, 2011.

[27] Prateesh Goyal, Preey Shah, Kevin Zhao, Georgios Nikolaidis, Mohammad Alizadeh, and Thomas E. Anderson. Backpressure flow control. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 779–805, Renton, WA, April 2022. USENIX Association.

[28] Sangtae Ha, Injong Rhee, and Lisong Xu. Cubic: a new tcp-friendly high-speed tcp variant. *ACM SIGOPS operating systems review*, 42(5):64–74, 2008.

[29] Yihua He, Nitin Batta, and Igor Gashinsky. Understanding switch buffer utilization in clos data center fabric.

[30] Chi-Yao Hong, Matthew Caesar, and P Brighten Godfrey. Finishing flows quickly with preemptive scheduling. *ACM SIGCOMM Computer Communication Review*, 42(4):127–138, 2012.

[31] Sijiang Huang, Mowei Wang, and Yong Cui. Traffic-aware buffer management in shared memory switches. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, pages 1–10. IEEE, 2021.

[32] V Jacobson and N Kathleen. Controlling queue delay-a modern aqm is just one piece of the solution to bufferbloat. *Asscociation for Computing Machinery (ACM Queue)*, 2012.

[33] Santosh Krishnan, Abhijit K Choudhury, and Fabio M Chiussi. Dynamic partitioning: A mechanism for shared memory management. In *IEEE INFOCOM'99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No. 99CH36320)*, volume 1, pages 144–152. IEEE, 1999.

[34] Gautam Kumar, Nandita Dukkipati, Keon Jang, Hassan MG Wassel, Xian Wu, Behnam Montazeri, Yaogong Wang, Kevin Springborn, Christopher Alfeld, Michael Ryan, et al. Swift: Delay is simple and effective for congestion control in the datacenter. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 514–528, 2020.

[35] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, et al. Hpcc: High precision congestion control. In *Proceedings of the ACM Special Interest Group on Data Communication*, pages 44–58. 2019.

[36] Shiyu Liu, Ahmad Ghalayini, Mohammad Alizadeh, Balaji Prabhakar, Mendel Rosenblum, and Anirudh Sivaraman. Breaking the transience-equilibrium nexus: A new approach to datacenter packet transport. In *NSDI*, pages 47–63, 2021.

[37] Rui Miao, Bo Li, Hongqiang Harry Liu, and Ming Zhang. Buffer sizing with hpcc. 2019.

[38] Radhika Mittal, Vinh The Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. Timely: Rtt-based congestion control for the datacenter. *ACM SIGCOMM Computer Communication Review*, 45(4):537–550, 2015.

[39] Behnam Montazeri, Yilong Li, Mohammad Alizadeh, and John Ousterhout. Homa: A receiver-driven low-latency transport protocol using network priorities. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 221–235, 2018.

[40] Aisha Mushtaq, Asad Khalid Ismail, Abdul Wasay, Bilal Mahmood, Ihsan Ayyub Qazi, and Zartash Afzal Uzmi. Rethinking buffer management in data center networks. *SIGCOMM Comput. Commun. Rev.*, 44(4):575–576, August 2014.

[41] Eugene Opsasnick. Buffer management and flow control mechanism including packet-based dynamic thresholding. *US patent US7953002B2*.

[42] R. Pan, P. Natarajan, C. Piglione, M. S. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg. Pie: A lightweight control scheme to address the bufferbloat problem. In *2013 IEEE 14th International Conference on High Performance Switching and Routing (HPSR)*, pages 148–155, July 2013.
[43] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C Snoeren. Inside the social network's (datacenter) network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 123–137, 2015.
[44] Ruixue Fan, A. Ishii, B. Mark, G. Ramamurthy, and Qiang Ren. An optimal buffer management scheme with dynamic thresholds. In *Seamless Interconnection for Universal Services. Global Telecommunications Conference. GLOBECOM'99. (Cat. No.99CH37042)*, volume 1B, pages 631–637 vol. 1b, Dec 1999.
[45] Ahmed Saeed, Varun Gupta, Prateesh Goyal, Milad Sharif, Rong Pan, Mostafa Ammar, Ellen Zegura, Keon Jang, Mohammad Alizadeh, Abdul Kabbani, et al. Annulus: A dual congestion control loop for datacenter and wan traffic aggregates. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 735–749, 2020.
[46] Danfeng Shan, Wanchun Jiang, and Fengyuan Ren. Absorbing micro-burst traffic by enhancing dynamic threshold policy of data center switches. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, pages 118–126. IEEE, 2015.
[47] Naveen Kr Sharma, Chenxingyu Zhao, Ming Liu, Pravein G Kannan, Changhoon Kim, Arvind Krishnamurthy, and Anirudh Sivaraman. Programmable calendar queues for high-speed packet scheduling. In *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*, pages 685–699, 2020.
[48] Balajee Vamanan, Jahangir Hasan, and TN Vijaykumar. Deadline-aware datacenter tcp (d2tcp). *ACM SIGCOMM Computer Communication Review*, 42(4):115–126, 2012.
[49] Cedric Westphal, Kiran Makhijani, and Richard Li. Packet trimming to reduce buffer sizes and improve round-trip times.
[50] Qiao Zhang, Vincent Liu, Hongyi Zeng, and Arvind Krishnamurthy. High-resolution measurement of data center microbursts. In *Proceedings of the 2017 Internet Measurement Conference*, pages 78–85, 2017.
[51] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. Congestion control for large-scale rdma deployments. *ACM SIGCOMM Computer Communication Review*, 45(4):523–536, 2015.

Appendices are supporting material that has not been peer-reviewed.

A ANALYSIS

In this section, we formally model and analyze a shared memory switch architecture with ABM's allocation scheme. The aim of our analysis is to show ABM's formal guarantees and its properties. We refer the reader to §2 for the terminology and definitions regarding ports, queues and priorities. Our analysis indeed generalizes both ABM and DT. Substituting static α_p values for ω_p^i (Definition 1) gives the analysis of DT.

A.1 Model

For generality, we model a switch with an arbitrary but fixed number of ports and queues per port. In particular, each port has only one queue per priority as defined in (§2). The switch in our model has a shared memory architecture with B buffer space. We denote the instantaneous occupied buffer at time t as $Q(t)$. Our analysis is based on a fluid model where packet (bits) arrivals and departures are assumed to be fluid and deterministic.

We denote by α_p, the parameter used by ABM for each priority in allocating the buffer. Each priority is associated with a separate queue at each port. We denote port indices by i and priority p. $\mathcal{P}$ is the set of priorities using the buffer. The number of congested queues of a priority p at time t is denoted by $n_p(t)$.

A.2 Formalizing ABM's Allocation

As described in §3, the threshold of a queue at port i and belonging to a priority p is calculated based on the alpha parameter α_p, the number of congested queues $n_p(t)$, the normalized drain rate of the queue indicated by $\gamma_p^i(t)$ and the remaining buffer $B - Q(t)$. Formally,

$$T_p^i(t) = \alpha_p \cdot \beta_p(t) \cdot \gamma_p^i(t) \cdot (B - Q(t)) \tag{12}$$

where, $\beta_p(t) = \frac{1}{n_p(t)}$ is the inverse of the total number of congested queues of priority p at time t.

Definition 1 (Omega - Adaptive α Parameter). *For a queue belonging to a priority p, the product $\alpha_p \cdot \beta_p(t) \cdot \gamma_p^i(t)$ in ABM's buffer allocation scheme (Eq. 12) is defined as Omega denoted by $\omega_p^i(t)$ and is viewed as an adaptive alpha parameter.*

$$\omega_p^i(t) = \alpha_p \cdot \beta_p(t) \cdot \gamma_p^i(t) \tag{13}$$

Based on the above definition of ω, in the following we derive an upper bound on the sum of ω values for all the classes of a priority p. Later in our analysis, we will see how the sum of ω values plays a key role in ABM's buffer allocation scheme. We will later use this upper bound to derive several properties and formal guarantees provided by ABM.

Lemma 1 (Property of Omega). *The instantaneous sum of $\omega_p^i(t)$ over all the queues belonging to a priority $p \in \mathcal{P}$ across all the ports is upper bounded by α_p.*

$$\sum_i \omega_p^i(t) \leq \alpha_p \tag{14}$$

Proof. Using Definition 1 and observing that $\beta_p(t)$ is the number of congested queues of a priority p is the same across all the queues of the same priority, we express the sum of $\omega_p^i(t)$ as follows:

$$\sum_i \omega_p^i(t) = \sum_i \alpha_p \cdot \beta_p(t) \cdot \gamma_p^i(t) = \alpha_p \cdot \beta_p(t) \cdot \sum_i \gamma_p^i(t)$$

Since $\gamma_p^i(t)$ is the normalized drain rate, $\gamma_p^i(t)$ is upper bounded by 1. Finally, we substitute $\beta_p(t) = \frac{1}{n_p(t)}$. We reduce the sum to an inequality as follows, where the last inequality holds since $\sum_i \gamma_p^i(t) \leq n_p(t)$ i.e., the sum of the normalized drain rates is upper bounded by the number of congested queues.

$$\alpha_c \cdot \beta_p(t) \cdot \sum_i \gamma_c^i(t) \leq \alpha_p \cdot \frac{1}{n_p(t)} \cdot n_p(t) \leq \alpha_p$$

□

A.3 Steady-State Analysis

We now analyze the steady-state behavior of ABM's buffer allocation scheme. Specifically, we say steady-state when the load-conditions remain stable and a steady buffer occupancy is achieved. Under steady-state, the queue lengths remain stable at less than or equal to their corresponding thresholds. To stress on the worst-case scenarios, we assume that any occupied queue is at the respective threshold. In our steady-state analysis, for simplicity of presentation, we drop the time variable in all the equations.

Under steady-state, we are interested in determining the overall buffer allocation and occupancy denoted by Q, the remaining buffer space $B - Q$ and ABM's threshold calculation per queue T_c^i.

Lemma 2 (Steady-state allocation)**.** *Under steady-state, given a set of congested queues, the overall buffer occupancy Q is given by Eq. 15, the remaining buffer $B - Q$ is given by Eq. 16 and the threshold per congested queue calculated by ABM is given by Eq. 17.*

$$Q = \frac{B \sum_i \sum_p \omega_p^i}{1 + \sum_i \sum_p \omega_p^i} \tag{15}$$

$$B - Q = \frac{B}{1 + \sum_i \sum_p \omega_p^i} \tag{16}$$

$$T_p^i = \frac{B \cdot \omega_p^i}{1 + \sum_i \sum_p \omega_p^i} \tag{17}$$

Proof. In the steady-state, from the assumption that the queue lengths are equal to their thresholds, we derive the overall buffer occupancy by summation of queue lengths of all the congested queues. Using Eq. 12 and Eq. 13 we express the total buffer occupancy Q as follows and solve for Q leading to the last equality.

$$Q = \sum_i \sum_p \omega_p^i \cdot (B - Q) = \frac{B \sum_i \sum_p \omega_p^i}{1 + \sum_i \sum_p \omega_p^i}$$

The remaining buffer space (Eq. 16) is then straight-forward by substituting Q. Finally, the threshold per queue T_p^i (Eq. 17) is obtained by definition from ABM's allocation scheme i.e., $T_p^i = \alpha_p \cdot \beta_p(t) \cdot \gamma_p^i(t) \cdot (B - Q) = \omega_p^i \cdot (B - Q)$.

□

In the following, we derive ABM's formal guarantee on isolation i.e., ABM offers minimum buffer space per priority based on the α parameters.

Theorem 1 (Isolation - Minimum guarantee)**.** *The total amount of buffer available for any priority p is lower bounded by B_p^{min} given by,*

$$B_p^{min} \geq \frac{B \cdot \alpha_p}{1 + \sum\limits_{p \in \mathcal{P}} \alpha_p}$$

Proof. The threshold T_p^i for each queue of priority p is given by Equation 17. We sum across all ports and obtain the total allocated buffer as follows,

$$\sum_i T_p^i = \frac{B \cdot \sum_i \omega_p^i}{1 + \sum_i \sum_{p \in \mathcal{P}} \omega_p^i} \geq \frac{B \cdot \alpha_p}{1 + \sum_{p \in \mathcal{P}} \alpha_p}$$

where the last inequality holds since $\sum_i \sum_p \omega_p^i \leq \sum_p \alpha_p$ from Lemma 1. □

Theorem 2 (Isolation - Preventing monopoly)**.** *The total amount of buffer available for any priority p is upper bounded by B_p^{max} given by,*

$$B_p^{max} \leq \frac{B \cdot \alpha_p}{1 + \alpha_p}$$

Proof. The proof is similar to Theorem 1. To obtain the upper bound, we use the property that $\sum_i \sum_{p \in \mathcal{P}} \omega_p^i \geq \alpha_p$, a case when only the priority p is using the buffer. □

Theorem 3 (Bounded drain time)**.** *The thresholds assigned by ABM upper bounds the drain time Γ for any queue of priority p given by,*

$$\Gamma \leq \frac{B \cdot \alpha_p}{(1 + \alpha_p) \cdot b}$$

Proof. Using Equation 17 from Lemma 2 and noting that the drain time is occupied buffer divided by its drain rate ($\gamma_p^i \cdot b$), we obtain the drain time Γ as follows for a queue at port i and of priority p,

$$\Gamma = \frac{B \cdot \alpha_p \cdot \frac{1}{n_p} \cdot \gamma_p^i}{\gamma_p^i \cdot b \cdot (1 + \sum_i \sum_p \omega_p^i)} \leq \frac{B \cdot \alpha_p}{b \cdot (1 + \alpha_p)}$$

The last inequality holds since $\sum_i \sum_p \omega_p^i \geq \alpha_p$ and $\frac{1}{n_p} \leq 1$. □

A.4 Transient-State Analysis

In this section we analyze ABM's transient-state properties. We define transient-state as a state when the buffer is initially in the steady-state and at time $t = 0$ load conditions change, creating a transient buffer state until the queue lengths stabilize. In particular, we consider that at time $t = 0$, a set of initially empty queues have incoming traffic. As a result, the thresholds and queue lengths undergo a transient state. Due to the appearance of new queues, ω_p^i of some of the existing queues get affected due to the changes in β_p (number of congested queues of a priority p) and γ_p^i (normalized drain rate). In the following, we introduce and describe certain notations specific to our transient-state analysis.

- The arrival rate of traffic at each *new* queue is denoted by r and the arrival process is fluid and deterministic. Note that we consider each port has a bandwidth of unit 1 and r is in the same unit.
- G_e denotes the set of queues whose ω_p^i gets affected.[5]
- G_{ne} denotes the set of queues whose ω_p^i does not get affected.
- For simplicity we denote the queue at port i and of priority p with ordered pairs as (i, p).
- The set of ordered pairs of existing queues is denoted as S_{old}. Observe that $S_{old} = G_{ne} \cup G_e$.
- The ordered pairs of new queues that trigger transient state are denoted as S_{new}.
- Dot over a variable denotes its rate of change i.e., derivative with respect to time. For example $\dot{x}$ denotes $\frac{dx}{dt}$.

A.5 Preliminaries

While the transient-state begins at $t = 0$, the initial buffer occupancy is based on the prior steady-state (Lemma 2) as expressed in Eq. 18 and Eq. 19.

$$T_p^i(0) = \frac{\omega_p^i \cdot B}{1 + \sum\limits_{S_{old}} \omega_p^i} \tag{18}$$

$$Q_p^i(0) = \begin{cases} \frac{\omega_p^i \cdot B}{1 + \sum\limits_{S_{old}} \omega_p^i} & , \text{for } \forall (i, p) \in S_{old} \\ 0 & , \text{for } \forall (i, p) \in S_{new} \end{cases} \tag{19}$$

[5]Note that the ω_p^i values of G_e only reduce. (It is not possible that ω_p^i increases due the appearance of a new queue)

At $t = 0^+$, ω_p^i of G_e change and remain same for the entire duration of transient state. At the same time, the ω_p^i of G_{ne} remains unchanged. Hence, such changes are assumed to occur at time $t = 0$ and the time variable is dropped for ω_p^i in the equations.

From Eq. 12, we express the rate of change of thresholds and queue lengths as follows,

$$\dot{T}_p^i(t) = -\omega_p^i \cdot \sum_{S_{old} \cup S_{new}} \dot{Q}_p^i(t) \tag{20}$$

$$\dot{Q}_p^i(t) = \begin{cases} max[-\gamma_p^i, min[\dot{T}_p(t), r - \gamma_p^i]] & \forall(i,p) \in S_{old} \\ r - \gamma_p^i & \forall(i,p) \in S_{new} \end{cases} \tag{21}$$

It can be proved by contradiction that $\frac{dT_p^i(t)}{dt} \leq 0 < r - \gamma_p^i$. Solving Eq. 20 and Eq. 21 for $t = 0+$,

$$\dot{T}_p^i(t) = -\omega_p^i \cdot \left(\sum_{S_{old}} max[-\gamma_p^i, \frac{dT_p(t)}{dt}_{(t=0+)}] \right) - \omega_c^i \cdot \sum_{S_{new}} (r - \gamma_p^i) \tag{22}$$

Recall that $S_{old} = G_e \cup G_{ne}$. All the queues belonging to G_e, will experience a change in their ω_p^i values at $t = 0^+$ resulting in their queue-lengths greater than threshold. As a result, the rate of change of their queue lengths is their corresponding drain rates. Eq. 22 can then be expanded as,

$$\dot{T}_p^i(t) = -\omega_p^i \cdot \left(\sum_{G_e} -\gamma_p^i + \sum_{G_{ne}} max[-\gamma_p^i, \dot{T}_p^i(t)] + \sum_{S_{new}} (r - \gamma_p^i) \right) \tag{23}$$

From Eq. 23, arrival rate of traffic in new queues i.e r can be expressed as,

$$r = \frac{\sum_{S_{new} \cup G_e} \gamma_p^i}{\sum_{S_{new}} 1} - \frac{\dot{T}_p^i(t) + \omega_p^i \cdot \left(\sum_{G_{ne}} max[-\gamma_p^i, \dot{T}_p(t)] \right)}{\omega_p^i \cdot \sum_{S_{new}} 1} \tag{24}$$

By applying summation over G_{ne} in Eq. 23 (will be seen later how this will be useful), r can be expressed as,

$$r = \frac{\sum_{S_{new} \cup G_e} \gamma_p^i}{\sum_{S_{new}} 1} - \frac{\sum_{G_{ne}} \dot{T}_p^i(t) + \left(\sum_{G_{ne}} max[-\gamma_p^i, \dot{T}_p(t)] \right) \cdot \sum_{G_{ne}} \omega_p^i}{(\sum_{G_{ne}} \omega_p^i) \cdot (\sum_{S_{new}} 1)} \tag{25}$$

Now it can be observed that the value of r influences all $\forall(i,p) \in G_{ne}$, $\dot{T}_p^i(t)$. In other words, the value of r influences the total i.e $\sum_{G_{ne}} \dot{T}_p^i(t)$ which is the aggregate rate at which thresholds drop for the non-affected set of queues i.e G_{ne}.

A.6 Case-1: Aggregate drain rate tracks the threshold changes

In this case, the arrival rate r is such that, the queues belonging to G_{ne} are able to reduce in length exactly tracking the changes in their thresholds. As a result, their queue-lengths remain equal to the threshold throughout the transient state i.e,

$$\left(\frac{dT_p^i(t)}{dt} \right)_{(t=0^+)} \geq -\gamma_p^i \tag{26}$$

leading to,

$$\sum_{\forall(i,p) \in G_{ne}} \left(\frac{dT_p^i(t)}{dt} \right)_{(t=0+)} \geq \sum_{\forall(i,p) \in G_{ne}} -\gamma_p^i \tag{27}$$

Using Eq. 26 and Eq. 27 in Eq. 25, the condition on r can be expressed as,

$$r \leq \frac{\sum_{S_{new} \cup G_e} \gamma_p^i}{\sum_{S_{new}} 1} + \left(\sum_{G_{ne}} \gamma_p^i \right) \cdot \frac{1 + \sum_{G_{ne}} \omega_p^i}{(\sum_{G_{ne}} \omega_p^i) \cdot (\sum_{S_{new}} 1)} \tag{28}$$

For such an arrival rate of traffic at new queues, in the following theorem we state the time up to which the new queues experience zero transient drops.

Theorem 4. *For an arrival rate r within Case-1 (Eq. 28) at a set of new queues S_{new}, given an initial state of the buffer at time $t = 0$, a new queue $(i,p) \in S_{new}$ experiences zero transient drops up to a time $t1_p^i$ given by Eq. 29*

$$t1_p^i = \frac{\omega_p^i \cdot B \cdot (1 + \sum_{G_{ne}} \omega_p^i)}{X_1 \cdot Y_1}$$

$$X_1 = (1 + \sum_{S_{old}} \omega_p^i)$$

$$Y_1 = ((r - \gamma_p^i) \cdot (1 + \sum_{G_{ne}} \omega_p^i) + \omega_p^i \cdot (\sum_{G_e} -\gamma_p^i + \sum_{S_{new}} (r - \gamma_p^i))) \tag{29}$$

Proof. Substituting Eq.26 and Eq.27 in Eq. 23 and using the result in Eq. 21 gives,

$$\dot{T}_p(t) = \frac{-\omega_p^i \cdot \left(\sum_{G_e} -\gamma_p^i + \sum_{S_{new}} (r - \gamma_p^i) \right)}{1 + \sum_{G_{ne}} \omega_p^i} \tag{30}$$

$$\dot{Q}_p^i(t) = \begin{cases} \dfrac{-\omega_p^i \cdot \left(\sum_{G_e} -\gamma_p^i + \sum_{S_{new}} (r - \gamma_p^i) \right)}{1 + \sum_{G_{ne}} \omega_p^i} & ,\forall(i,p) \in G_{ne} \\ -\gamma_p^i & ,\forall(i,p) \in G_e \\ r - \gamma_p^i & ,\forall(i,p) \in S_{new} \end{cases} \tag{31}$$

These differential equations will be valid as long as $Q_p^i(t) = T_p^i(t)$ for $\forall(i,p) \in G_{ne}$, $Q_p^i(t) \geq T_p^i(t)$ for $\forall(i,p) \in G_e$ and $Q_p^i(t) < T_p^i(t)$ for newly created queues i.e $\forall(i,p) \in S_{new}$. Solving these equations, using the initial conditions, Eq. 18 and Eq. 19 leads to,

$$T_p^i(t) = \frac{\omega_p^i \cdot B}{1 + \sum\limits_{S_{old}} \omega_p^i} - \frac{\omega_p^i \cdot \left(\sum\limits_{G_e} -\gamma_p^i + \sum\limits_{S_{new}} (r - \gamma_p^i)\right) \cdot t}{1 + \sum\limits_{G_{ne}} \omega_p^i} \tag{32}$$

$$Q_p^i(t) = \begin{cases} \frac{\omega_p^i \cdot B}{1+\sum_{S_{old}} \omega_p^i} - \frac{\omega_p^i \cdot t \cdot (\sum_{G_e} -\gamma_p^i + \sum_{S_{new}} (r - \gamma_p^i))}{1+\sum_{G_{ne}} \omega_p^i} & \forall(i,p) \in G_{ne} \\ \frac{\omega_p^i \cdot B}{1+ \sum\limits_{\forall(i,p)\in S_{old}} \omega_p^i} - \gamma_p^i \cdot t & \forall(i,p) \in G_e \\ (r - \gamma_p^i) \cdot t & \forall(i,p) \in S_{new} \end{cases} \tag{33}$$

As we can observe from Eq. 32 and Eq. 33, the new queues will grow in length without dropping packets up to a time $t1_p^i$ when the threshold equals the queue length. The transient state continues after $t1_p^i$ until all the queues achieve a steady state occupancy. By equating Eq. 32 and Eq. 33 for the case of $\forall(i,p) \in S_{new}$, we obtain $t1_p^i$ as in Eq. 29. □

In order to offer guarantees, it is absolutely required that either γ_p^i is constant. The reason being that there is a dependency between γ_p^i and the number of queues of the same port using buffer, a dependency that is fundamentally impossible to evade unless γ_p^i is constant. As a result of this assumption, $G_e = \phi$ and $S_{old} = G_{ne}$ and Eq. 29 reduces to,

$$t1_p^i = \frac{\alpha_p \cdot \frac{1}{n_p} \cdot \gamma_p^i \cdot B}{(r - \gamma_p^i) \cdot (1 + \sum\limits_{S_{old}} \omega_p^i + \omega_p^i \cdot \sum\limits_{S_{new}} 1)} \tag{34}$$

As an example, we can further simplify for a case with one high priority and one low priority using the buffer where load variations occur for *High Priority* whose α value is α_H and the existing *Low Priority* in the queues have α value of α_L. We can then guarantee that for an arrival rate r that satisfies *Case-1* will experience zero drops i.e., no transient drops if its duration t satisfies the following condition:

$$t1_p^i = \frac{\alpha_H \cdot \frac{1}{n_p} \cdot \gamma_p^i \cdot B}{(r - \gamma_p^i) \cdot \left(1 + \alpha_L + \alpha_H \cdot \frac{1}{n_p} \cdot \gamma_p^i \cdot \sum\limits_{\forall(i,p)\in S_{new}} 1\right)} \tag{35}$$

Observe that Eq. 35 is independent of the number of queues of *Low Priority* and hence it can be said that *High Priority* isolation can be guaranteed.

A.7 Case-2: Aggregate drain rate is slower than the changes in thresholds

In this case, the arrival rate r is such that, the queues belonging to G_{ne} are unable to reduce in length in accordance with the changes in their thresholds. As a result, their queue-lengths remain greater than the threshold throughout the transient state i.e,

$$\left(\frac{dT_p^i(t)}{dt}\right)_{(t=0^+)} < -\gamma_p^i \tag{36}$$

leading to,

$$\sum_{\forall(i,p)\in G_{ne}} \left(\frac{dT_p^i(t)}{dt}\right)_{(t=0+)} < \sum_{\forall(i,p)\in G_{ne}} -\gamma_p^i \tag{37}$$

Using Eq. 36 and Eq. 37 in Eq. 25, the condition on r can be expressed as,

$$r > \frac{\sum\limits_{S_{new}\cup G_e} \gamma_p^i}{\sum\limits_{S_{new}} 1} + \left(\sum_{G_{ne}} \gamma_p^i\right) \cdot \frac{1 + \sum\limits_{G_{ne}} \omega_p^i}{(\sum\limits_{G_{ne}} \omega_p^i) \cdot (\sum\limits_{S_{new}} 1)} \tag{38}$$

Theorem 5. *For an arrival rate r within Case-2 (Eq. 38) at a set of new queues S_{new}, given an initial state of the buffer at time $t = 0$, a new queue $(i,p) \in S_{new}$ experiences zero transient drops up to a time $t1_p^i$ given by Eq. 39.*

$$t1_p^i = \frac{\omega_p^i \cdot B}{X_2 \cdot Y_2}$$
$$X_2 = 1 + \sum_{\forall(i,p)\in S_{old}} \omega_p^i$$
$$Y_2 = (r - \gamma_p^i) + \omega_p^i \cdot \left(\sum_{\forall(i,p)\in S_{old}} -\gamma_p^i + \sum_{\forall(i,p)\in S_{new}} (r - \gamma_p^i)\right) \tag{39}$$

We omit the proof of Theorem 5. The proof is similar to Theorem 4. As an example, with one high priority (α_H) queue experiencing burst and low priority (α_L) queues occupying buffer, from Lemma 1 on the property of ω and observing that $\sum_{\forall(i,p)\in S_{old}} -\gamma_p^i$ is the *Total drain rate of the congested ports of* S_{old}, we derive the following relation where n is the number of congested ports belonging to S_{old} and BW is the bandwidth of each port assuming ports are of same bandwidth (the assumption is not critical to our analysis, it can easily generalized).

$$t1_p^i = \frac{\frac{\alpha_H}{(1+\alpha_L)} \cdot \frac{1}{n_p} \cdot \gamma_p^i \cdot B}{\left((r - \gamma_p^i) + \frac{\alpha_H}{n_p} \cdot \gamma_p^i \cdot \left(-n \cdot BW + \sum\limits_{S_{new}} (r - \gamma_p^i)\right)\right)} \tag{40}$$

where n is the number of congested ports belonging to S_{old} i.e.,

$$n \cdot BW = \sum_{\forall(i,p)\in S_{old}} -\gamma_p^i \tag{41}$$

Notice that the presence of n in Eq. 40, is a dependency on the number of congested ports of *Low Priority*. However, n only creates a positive effect on $t1_p^i$ i.e., greater the n greater is $t1_p^i$. On the other hand, Eq. 40 is independent of negative dependencies as was in the traditional algorithm DT where a higher number of congested queues lead to lower $t1_p^i$ leading to faster transient drops.

A.8 Burst Tolerance

Building on the analysis in the previous sections, in this section, we discuss how the steady-state and transient-state behavior of ABM relates to burst tolerance. We denote the burst tolerance for a queue of priority p at port i as $Burst_p^i$ and is defined as follows:

$$Burst_p^i = r \cdot t1_p^i \tag{42}$$

where r is the arrival rate of traffic and $t1_p^i$ is the amount of time starting from $t = 0$ until the queue experiences zero drops.

Based on our analysis of transient state in Appendix A.4, we generate analytical graphs as shown in Figure 5 (§2.3). For each arrival rate r, we first distinguish whether r falls under case-1 (Appendix A.6) or case-2 (Appendix A.7). We then calculate the corresponding time $t1_p^i$ until the burst experiences zero drops. We then multiply r and $t1_p^i$ to obtain the burst tolerance.