Credence: Augmenting Datacenter Switch Buffer Sharing with ML Predictions *

Vamsi Addanki
TU Berlin

Maciej Pacut
TU Berlin

Stefan Schmid
TU Berlin

Abstract

Packet buffers in datacenter switches are shared across all the switch ports in order to improve the overall throughput. The trend of shrinking buffer sizes in datacenter switches makes buffer sharing extremely challenging and a critical performance issue. Literature suggests that push-out buffer sharing algorithms have significantly better performance guarantees compared to drop-tail algorithms. Unfortunately, switches are unable to benefit from these algorithms due to lack of support for push-out operations in hardware. Our key observation is that drop-tail buffers can emulate push-out buffers if the future packet arrivals are known ahead of time. This suggests that augmenting drop-tail algorithms with predictions about the future arrivals has the potential to significantly improve performance.

This paper is the first research attempt in this direction. We propose CREDENCE, a drop-tail buffer sharing algorithm augmented with machine-learned predictions. CREDENCE can unlock the performance only attainable by push-out algorithms so far. Its performance hinges on the accuracy of predictions. Specifically, CREDENCE achieves near-optimal performance of the best known push-out algorithm LQD (Longest Queue Drop) with perfect predictions, but *gracefully* degrades to the performance of the simplest drop-tail algorithm Complete Sharing when the prediction error gets arbitrarily worse. Our evaluations show that CREDENCE improves throughput by 1.5x compared to traditional approaches. In terms of flow completion times, we show that CREDENCE improves upon the state-of-the-art approaches by up to 95% using off-the-shelf machine learning techniques that are also practical in today's hardware. We believe this work opens several interesting future work opportunities both in systems and theory that we discuss at the end of this paper.

1 Introduction

Datacenter switches come equipped with an on-chip packet buffer that is shared across all the device ports in order to improve the overall throughput and to reduce packet drops.

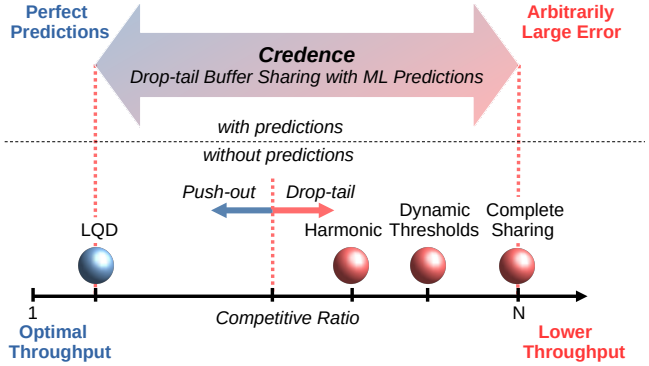


Figure 1: Augmenting drop-tail buffer sharing with ML predictions has the potential to significantly improve throughput compared to the best possible drop-tail algorithm (without predictions), and unlock the performance that was only attainable by push-out so far.

Unfortunately, buffers have become increasingly expensive and chip-manufacturers are unable to scale up buffer sizes proportional to capacity increase [11]. As a result, the buffer available per port per unit capacity of datacenter switches has been gradually reducing over time. Worse yet, datacenter traffic is bursty even at microsecond timescales [20, 51]. This makes it challenging for a buffer sharing algorithm to maximize throughput. Recent measurement studies in large scale datacenters point-out the need for improved buffer sharing algorithms in order to reduce packet drops during congestion events [20]. To this end, buffer sharing under shallow buffers is an emerging critical problem in datacenters [1, 10].

The buffer sharing problem has been widely studied in the literature from an online perspective [13] with the objective to maximize throughput [8, 22, 24, 30, 31]. Traditional online algorithms for buffer sharing can be classified into two types: **drop-tail** e.g., Dynamic Thresholds (DT) [18], Harmonic [30], ABM [1] and **push-out** e.g., Longest Queue Drop (LQD). The performance gap of these algorithms compared to an offline optimal (clairvoyant) algorithm can be expressed in terms of the competitive ratio [13]. For instance, we say that an online algorithm is 2-competitive if it performs at most 2x worse

*Author's version. Final version to appear in Usenix NSDI 2024.

Algorithm	Competitive Ratio
Complete Sharing [24]	$N+1$
Dynamic Thresholds [18, 24]	$O(N)$
Harmonic [30]	$\ln(N)+2$
LQD (push-out) [8, 24]	1.707
LateQD (clairvoyant) [12]	1
CREDENCE	$\min(1.707\,\eta,\ N)$

Table 1: CREDENCE's performance smoothly depends on the prediction error (η). CREDENCE outperforms traditional drop-tail buffer sharing algorithms and performs as good as push-out when the predictions are perfect ($\eta = 1$) but is also never worse than Complete Sharing even when the predictions are bad ($\eta \to \infty$). N denotes the number of ports.

compared to an offline optimal algorithm. Figure 1 illustrates the performance spectrum of drop-tail and push out buffer sharing algorithms. In terms of throughput-competitiveness, it is well-known that push-out algorithms perform significantly better than drop-tail algorithms. In fact, no deterministic drop-tail algorithm can perform better than a certain throughput (a lower bound for competitive ratio), beyond which only push-out algorithms exist (Figure 1). Table 1 presents the competitive ratios of known algorithms. Interestingly, LQD pushes out packets when the buffer is full, and it is ≈ 2-competitive whereas Complete Sharing drops packets when the buffer is full, but it is $N+1$-competitive.

Intuitively, the poor throughput-competitiveness of drop-tail buffers owes it to the fundamental challenge that utilizing the buffer for some queues comes at the cost of deprivation of buffer for others [1]. To this end, drop-tail algorithms proactively drop packets i.e., packets are dropped even when the buffer has remaining space [9, 18, 24, 26, 30]. On one hand, maintaining remaining buffer space is necessary to serve future packet arrivals. On the other hand, maintaining remaining buffer space could lead to under-utilization, throughput loss and excessive packet drops. In contrast, the superior throughput-competitiveness of push-out algorithms owes it to their fundamental advantage to push out packets instead of dropping them. Hence, push-out algorithms can utilize the entire buffer as needed and only push out packets when multiple ports contend for buffer space. Although push-out algorithms offer far superior performance guarantees compared to drop-tail, hardly any datacenter switch supports push-out operations for the on-chip shared buffer. This begs the question: Are drop-tail buffer sharing algorithms ready for the trend of shrinking buffer sizes?

Our key observation is that every push-out algorithm can be converted to a drop-tail algorithm. However, such a conversion requires certain (limited) visibility into the future packet arrivals. Specifically, pushing out a packet is equivalent to dropping the packet when it arrives. Recent advancements in traffic predictions play a pivotal role in providing such visibility into the future packet arrivals: paving a way for better drop-tail buffer sharing algorithms.

In this paper, we take the first step in this direction. Figure 1 illustrates our perspective. We propose CREDENCE, a drop-tail buffer sharing algorithm augmented with machine-learned predictions. CREDENCE's performance is tied to the accuracy of these predictions. As the prediction error decreases, CREDENCE unlocks the performance of push-out algorithms and reaches the performance of the best-known algorithm. Even when the prediction error grows arbitrarily large, CREDENCE offers at least the performance of the simplest drop-tail algorithm Complete Sharing. Table 1 gives the competitive ratio of CREDENCE as a function of the prediction error η. Importantly, CREDENCE's performance *smoothly* varies with the prediction error, generalizing the performance space between the known push-out and drop-tail algorithms. Hence, CREDENCE achieves the three goals of prediction-augmented algorithms, in the literature referred to as consistency, robustness and smoothness [38, 42].

In addition to the theoretical guarantees for CREDENCE's performance, our goal is also its practicality. Specifically, without predictions, CREDENCE's core logic only uses additions, subtractions, and does not add additional complexity compared to existing approaches. For predictions, we currently use random forests, which have been recently shown to be feasible on programmable switches at line rate [4, 15]. A full implementation of CREDENCE in hardware unfortunately requires switch vendor involvement since buffer sharing is merely a blackbox even in programmable switches. With this paper, we wish to gain attention from switch vendors on the fundamental blocks required for such algorithms to be deployed in the dataplane. We currently implement CREDENCE in NS3 to evaluate its performance using realistic datacenter workloads. We present a detailed discussion on the practicality of CREDENCE later in this paper.

Our evaluations show that CREDENCE performs 1.5x better in terms of throughput and up to 95% better in terms of flow completion times, compared to alternative approaches.

We believe CREDENCE is a stepping stone towards further improving buffer sharing algorithms. Especially, achieving better performance than CREDENCE under large prediction error remains an interesting open question. Our approach of augmenting buffer sharing with predictions is not limited to drop-tail algorithms, but push-out algorithms can also benefit from predictions. We discuss exciting future research directions both in systems and theory at the end of this paper. In summary, our key contributions in this paper are:

- CREDENCE, the first buffer sharing algorithm augmented with predictions, achieving near-optimal performance with perfect predictions while also guaranteeing performance under arbitrarily large prediction error, and gradually degrading the performance as the prediction error increases.
- Extensive evaluations using realistic datacenter workloads, showing that CREDENCE outperforms existing approaches in terms of flow completion times.
- All our artifacts have been made publicly available at https://github.com/inet-tub/ns3-datacenter.

2 Motivation

In this section, we provide a brief background and motivate our approach by highlighting the drawbacks of traditional approaches. We show the potential for reaching close-to-optimal performance when buffer sharing algorithms are augmented with machine-learned predictions. To this end, we first describe our model and throughput competitiveness (§2.1). We then discuss the drawbacks of existing approaches (§2.2). We show that a renewed hope for improved buffer sharing is enabled by the recent rise in algorithms with predictions (§2.3).

2.1 Buffer Sharing from Online Perspective

A network switch receives packets one after the other at each of its ports. The switch does not know the packet arrivals ahead of time. This makes buffer sharing inherently an *online* problem i.e., algorithms must take instantaneous decisions upon packet arrivals without the knowledge of the future. In order to systematically understand the performance of such algorithms, we take an online approach following the classical model in the literature [8, 22, 24, 30, 31]. In this section, we describe our model intuitively, and we refer to Appendix A for formal definitions. Figure 2 illustrates the model.

Buffer model: We consider an output-queued switch with N ports and a buffer size of B. Buffer is shared across all the ports. A buffer sharing algorithm takes buffering decisions that we describe next. We assume that time is discrete. At most N packets can arrive in a single timeslot (since there are N ports), and each port removes at most one packet in a timeslot.

Online algorithm: When a packet arrives, a buffer sharing algorithm determines whether it should be accepted into the available buffer space. Drop-tail algorithms can only accept or discard incoming packets, while push-out algorithms can also remove packets from the buffer.

Objective: The network throughput is of utmost importance for datacenter operators since throughput often relates to the cost in typical business models (e.g., $ per bandwidth usage). We hence consider throughput as an objective function, following the literature. Specifically, for any packet arrival sequence, our objective is to maximize the total number of transmitted packets. The throughput maximization objective is closely related to packet drops minimization objective. In this sense, our objective captures two important performance metrics i.e., throughput and packet drops.

Competitive Ratio: We use competitive ratio as a measure to compare the performance of an online algorithm to the optimal offline algorithm. Specifically, let ALG and OPT be an online and optimal offline algorithm correspondingly. Let $ALG(\sigma)$ be the throughput of ALG for the packet arrival sequence σ. We say an algorithm ALG is c-competitive if the following relation holds for any packet arrival sequence.

$$OPT(\sigma) \leq c \cdot ALG(\sigma)$$

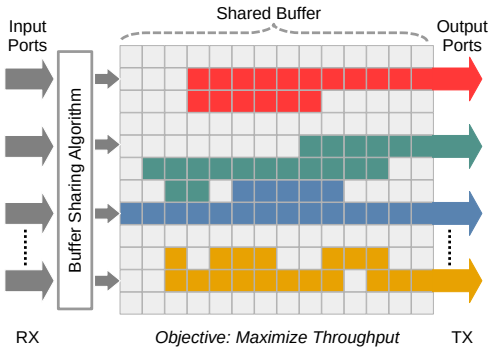


Figure 2: The switch has a buffer size of B shared across N output ports. Each color indicates the packets residing in the shared buffer corresponding to each port. A buffer sharing algorithm takes decisions (accept or drop) for each input packet.

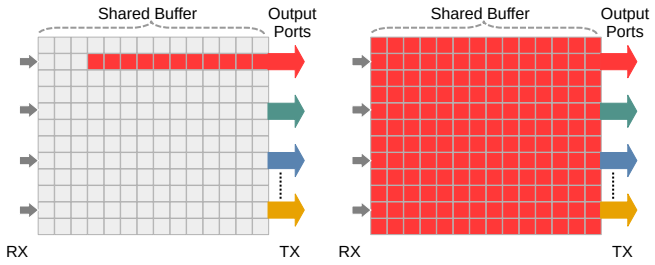
Competitive ratio is a particularly interesting metric for buffer sharing since it offers performance guarantees without any assumptions on specific traffic patterns. For example, the buffer may face excessive packet drops or may temporarily experience throughput loss due to bursty traffic. One could argue that the buffer sharing algorithm is the culprit and should have allocated more buffer to the bursty traffic. While this may have solved the problem for a particular bursty arrival, the same solution could result in unexpected drops and throughput loss if there were excessive bursty arrivals i.e., large bursts could monopolize the buffer. Instead, from an online perspective, better competitive ratio indicates that the buffer sharing algorithm performs close to optimal under any traffic conditions.

■ ***Takeaway.*** *A buffer sharing algorithm with lower competitive ratio improves the throughput of the switch and reduces packet drops under worst-case packet arrival patterns.*

2.2 Drawbacks of Traditional Approaches

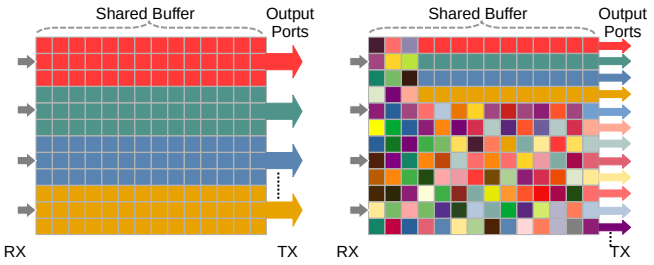
We observe two main drawbacks of traditional buffer sharing algorithms, both affecting the competitive ratio. First, algorithms proactively and unnecessarily drop packets in view of accommodating future packet arrivals. Second, algorithms reactively drop packets when the buffer is full and incur throughput loss, which could have been avoided. We argue that these drawbacks are rather fundamental to drop-tail algorithms and cannot be addressed by traditional online approaches.

Proactive unnecessary packet drops → throughput loss: A drop-tail buffer sharing algorithm typically drops packets even if there is remaining buffer space available [1, 9, 18]. We refer to such drops as proactive drops. Being proactive is indeed necessary in order to accommodate transient bursts. However, proactive packet drops and the corresponding remaining buffer space ends up being wasteful if the future packet arrivals do not need additional buffer space (if the anticipated burst does not arrive). Figure 3a and Figure 3b



(a) ALG (b) OPT

Figure 3: Upon a large burst arrival, a typical drop-tail algorithm (ALG) *proactively* drops the incoming packets in anticipation of future bursts and significantly under-utilizes the buffer. In this case, an optimal offline algorithm accepts the entire burst without any packet drops.



(a) ALG (b) OPT

Figure 4: In pursuit of high burst absorption, a drop-tail algorithm ALG may absorb bursts but this results in excessive *reactive* drops for the future packet arrivals. In this case, an optimal offline algorithm OPT drops few packets such that the overall throughput is maximized.

illustrate an example. Consider a traffic pattern where there is little to no congestion on all the ports but once in a while, a large burst appears. Specifically, the buffer is empty initially and a large burst of size B appears. A deterministic drop-tail algorithm has two choices: *(i)* accept a portion of the burst and proactively drop the rest of the burst or *(ii)* accept the entire burst. Typical algorithms in the literature choose the former in view of accommodating future packet arrivals. An optimal offline algorithm that knows the arrivals ahead of time would accept the entire burst of size B in this case. This makes an online algorithm at least c-competitive for this particular arrival pattern, where $\frac{1}{c}$ is the fraction of the burst accepted: since the optimal solution accepts and transmits B packets over time, whereas an online algorithm only accepts and transmits only $\frac{B}{c}$ packets over time. We observe that recent works focus on minimizing proactive unnecessary packet drops by prioritizing bursty traffic to the extent that they allow burst on a single port to monopolize the buffer [1, 9, 26]. However, note that competitive ratio is not defined for a particular arrival sequence, but over all scenarios. To this end, accepting a larger burst size may be helpful in the above example but if there were indeed future packet arrivals on other ports that need buffer, the algorithm incurs excessive reactive drops (described next) and throughput loss.

Reactive avoidable packet drops → throughput loss: Any drop-tail algorithm is forced to drop the incoming packets once the shared buffer is full. We call such drops reactive drops. Reactive drops result in throughput loss if the algorithm fills up significant portion of the buffer on a small set of ports but reactively drops incoming packets to other ports. Figure 4a and Figure 4b illustrate an example. Consider that the buffer is initially empty and four simultaneous bursts each of size B arrive to four ports. If an algorithm proactively drops a significant portion of the bursts, it would suffer under arrival sequences such as in the previous example (Figure 3a). Alternatively, the algorithm may choose to accept a larger portion of the bursts and ends up filling up the entire buffer in aggregate. At this point, several short bursts arrive to multiple other ports. An optimal offline algorithm accepts only a fraction of the large bursts such that it is able to accommodate upcoming short bursts. In doing so, the optimal algorithm benefits in throughput since the switch transmits packets from more number of ports. However, since the online algorithm fills up the entire buffer due to the initial large bursts, it is forced to reactively drop the upcoming short bursts, losing throughput. In fact, a similar arrival pattern for Dynamic Thresholds yields at least $\Omega\left(\sqrt{\frac{N}{\log(N)}}\right)$-competitiveness [24]. The known upper bound for Dynamic Thresholds is $O(N)$ [24]. Further, it has been shown in the literature that no deterministic drop-tail algorithm can be better than $\Omega\left(\frac{\log(N)}{\log(\log(N))}\right)$-competitive [30].

Interestingly, push-out algorithms are not prone to the problems discussed above, since they can take revocable decisions i.e., to accept a packet and drop it later. Hence, push-out algorithms do not have to maintain free space in the buffer in order to accommodate transient bursts. Instead, such algorithms can defer the dropping decision until the moment the drop turns out to be necessary.

■ ***Takeaway.*** *Traditional drop-tail algorithms are fundamentally limited in throughput-competitiveness as they are unable to effectively navigate proactive and reactive drops due to the online nature of the problem i.e., future packet arrivals are unknown to the algorithm.*

2.3 Predictions: A Hope for Competitiveness

Given that the fundamental barrier in improving drop-tail buffer sharing algorithms is the lack of visibility into the future arrivals, we turn towards predictions. The recent rise of algorithms with predictions offers a renewed hope for competitive buffer sharing. Algorithms with predictions successfully enabled close to optimal performance for various classic problems [42]. The core idea is to guide the underlying online algorithm with certain knowledge about the future obtained via predictions. The machine-learned oracle that produces predictions is considered a blackbox with a certain error. The main challenge is to offer performance guarantees at the extremes i.e., close to optimal performance under perfect predictions and a minimum performance guarantee when the

prediction error gets arbitrarily large. Further, it is desirable that the competitiveness of the algorithm *smoothly* degrades as the prediction error grows.

2.3.1 Prediction Model

In the context of the buffer sharing problem, there are several prediction models that can be considered e.g., drops or packet arrivals. In this paper, we assume that a blackbox machine-learned oracle predicts packet drops. Our choice is due to the fact that packet drops are the basic decisions made by an algorithm. Concretely, we consider an oracle that predicts whether an incoming packet would eventually be dropped (or pushed out) by the Longest Queue Drop (LQD) algorithm serving the same packet arrival sequence. We classify the predictions into four types: *(i)* true positive i.e., a correct prediction that a packet is eventually dropped by LQD, *(ii)* false negative i.e., an incorrect prediction that a packet is eventually transmitted by LQD, *(iii)* false positive i.e., an incorrect prediction that a packet is eventually dropped by LQD and *(iv)* true negative i.e., a correct prediction that a packet is eventually transmitted by LQD. Figure 5 summarizes this classification. Following the literature [38, 42], our goals for prediction-augmented buffer sharing are consistency, robustness and smoothness.

α-Consistent buffer sharing algorithm has a competitive ratio α when the predictions are all true i.e., perfect predictions.

β-Robust buffer sharing algorithm has a competitive ratio β when the predictions are all false i.e., large prediction error.

Smoothness is a desirable property such that the competitive ratio degrades smoothly as the prediction error grows i.e., a small change in error does not drastically influence the competitive ratio.

Our goal is to design a prediction-augmented buffer sharing algorithm that is close to 1-consistent (with perfect predictions) i.e., near-optimal, at most N-robust (with arbitrarily large error) i.e., not worse than Complete Sharing algorithm, and has the desirable property of smoothness.

2.3.2 Common Pitfalls

It is intuitive that predictions can potentially improve the performance of a drop-tail algorithm. For instance, in the examples from Figure 3 and Figure 4, our prediction-augmented online algorithm could take nearly the same decisions as a push-out algorithm. However, the main challenge is to ensure robustness and smoothness. If an algorithm blindly trusts the predictions, we observe that false positive and false negative predictions have a significantly different impact on the performance.

Excessive false positives can lead to starvation: The worst case for a naive algorithm that blindly trusts predictions is when all the predictions are false positives. In this case, the algorithm ends up dropping every incoming packet. Blindly trusting false predictions could lead to a competitive ratio worse than the simplest drop-tail algorithm Complete Sharing

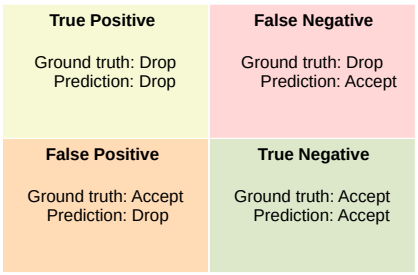


Figure 5: Confusion matrix for our prediction model.

i.e., the competitive ratio becomes unbounded (∞-robust) if the predictions are mostly false positives.

A single false negative can hurt throughput forever: A naive algorithm that blindly relies on false negative predictions is susceptible to adverse effects that propagate over time. Consider a packet arrival sequence that hits only one queue initially and consider that the predictions are all true negatives until the queue length reaches $B-1$, where B is the total buffer size. At this point, one more packet arrives and our prediction is a false negative. As a result, our naive algorithm has a queue of size B and the optimal algorithm has a queue of size $B-1$. Note that all non-empty queues drain one packet after each timeslot. From here on, in every timeslot, one packet (first) arrives to the large queue and one packet (second) arrives to any other queue. Also consider that all the predictions are true from now on. The optimal algorithm accepts both first and second packet in every timeslot. However, in every timeslot our naive approach can only accept the first packet to the large queue and cannot accept the second packet since the buffer is full. Notice that relying on just one false negative resulted in cumulative drops in this case even though all other predictions were true. In fact, a tiny error such as just N number of false negatives even with all other predictions being true could result in a competitive ratio for a naive approach as worse as Complete Sharing.

■ ***Takeaway.*** *Augmenting drop-tail algorithms with predictions has the potential to unlock the optimal performance. Ensuring performance guarantees with inaccurate predictions remains a challenge.*

3 Prediction-Augmented Buffer Sharing

Reflecting on our observations in §2, our goal is to design a drop-tail buffer sharing algorithm that performs close to optimal with perfect predictions but also provides a minimum performance guarantee when the prediction error is arbitrarily large. In essence, our aim is to enable performance improvement in terms of throughput and packet drops in datacenter switches. To this end, we first present an overview of our algorithm (§3.1). We then present the workings of CREDENCE (§3.2) and discuss its properties (§3.3). Finally, we discuss the practicality of CREDENCE (§3.4).

3.1 Overview

In a nutshell, CREDENCE relies on predictions and *follows* a push-out algorithm, reaching close to optimal performance under perfect predictions. CREDENCE cleverly takes certain decisions independent of the predictions in order to guarantee a minimum performance. Further, CREDENCE's competitiveness gradually degrades as prediction error grows (a property known as smoothness [38]), hence the algorithm still performs near-optimally when predictions are slightly inaccurate.

CREDENCE *follows* Longest Queue Drop algorithm: Our design of CREDENCE consists of two key ingredients. First, CREDENCE uses thresholds as a drop condition irrespective of the predictions. CREDENCE treats thresholds as queue lengths of LQD and updates the thresholds based on the LQD algorithm (simply arithmetic) upon every packet arrival. Second, CREDENCE relies on predictions as long as the queue lengths satisfy the corresponding thresholds. The combination of thresholds and predictions allows CREDENCE to closely *follow* the Longest Queue Drop algorithm (LQD) without requiring push-out operations[1].

CREDENCE guarantees performance under extremities: When all the predictions are perfectly accurate, CREDENCE achieves a competitive ratio of 1.707 (consistency) due to the straight-forward argument that the drops by CREDENCE and LQD are equivalent for true predictions. In order to guarantee a minimum performance even with arbitrarily large prediction error (robustness), CREDENCE bypasses the threshold and predictions as long as the longest queue is within $\frac{B}{N}$ size. Here, B is the buffer size and N is the number of ports. This allows CREDENCE to be most N-competitive even under large prediction error, similar to the Complete Sharing algorithm. We prove our claim formally in Appendix C.

CREDENCE smoothly degrades with prediction error: We design our error function in terms of the performance of LQD and the predicted drops. We analyze the types of drops incurred by CREDENCE due to false positive and false negative predictions. This allows us to show that CREDENCE satisfies the smoothness property i.e., the competitive ratio smoothly degrades from 1.707 to N as the prediction error grows.

3.2 CREDENCE

We now present CREDENCE and explain how it operates. Algorithm 1 presents the pseudocode of CREDENCE. Our pseudocode is simplified to discrete time for ease of presentation and for simplicity of analysis. It can be trivially extended to continuous time, and our implementation incorporates it[2].

Arrival: Upon a packet arrival, CREDENCE has three important steps that are highlighted in Algorithm 1. First, CREDENCE updates the threshold for the current queue (highlighted in blue). Second, CREDENCE takes a decision based on the thresholds and predictions whether or not to accept the incoming packet (highlighted in yellow). Finally, the packet is either accepted or dropped. We next describe each of these steps in detail. Third, depending on the state of the buffer, CREDENCE bypasses the thresholds and predictions with a safeguard condition in order to accept or drop the incoming packet (highlighted in green).

Thresholds: CREDENCE updates its thresholds based on the longest queue drop algorithm. Specifically, upon a packet arrival at time t to a queue i, CREDENCE increments the threshold $T_i(t)$ for queue i by the packet size. If upon arrival the sum of thresholds $\Gamma(t)$ is equal to the buffer size B, then CREDENCE first decrements the longest queue threshold by packet size and then increments the threshold for queue i by the packet size. Note that upon a packet arrival to a queue, the corresponding threshold is updated before accepting or dropping the packet.

Drop criterion: Similar to existing threshold-based algorithms, CREDENCE also uses thresholds as a drop criterion. CREDENCE compares the queue length $q_i(t)$ of a queue i against its threshold $T_i(t)$ and drops an incoming packet if the queue length is larger than or equal to the corresponding threshold. If and only if an incoming packet satisfies the thresholds, then CREDENCE takes input from a machine-learned oracle that predicts whether to accept or drop according to our prediction model discussed in §2.3.1. Finally, based on the thresholds and predictions, CREDENCE either accepts or drop the incoming packet.

Safeguard: In order to bound CREDENCE's competitiveness under arbitrarily large prediction error, we bypass the above drop criterion under certain cases. Specifically, when the longest queue length is less than $\frac{B}{N}$, CREDENCE always accepts a packet irrespective of the thresholds and predictions. This ensures that CREDENCE is at least N-competitive even with large prediction error. Our safeguard is based on the observation that even the push-out longest queue drop algorithm cannot push out a packet from a queue less than $\frac{B}{N}$ size since the longest queue must be at least $\frac{B}{N}$ size when the buffer is full. In essence, CREDENCE circumvents the impact of large prediction error by accepting packets until a certain amount of buffer is filled up.

Predictions: CREDENCE can be used with any ML oracle that predicts whether to accept or drop a packet, according to our prediction model (see §2.3.1). We do not rely on the internal details of the oracle. However, certain choices of ML oracles are better suited to operate within the limited resources available in a switch hardware. We discuss further on our choice of oracle later in §3.4.

3.3 Properties of CREDENCE

CREDENCE offers attractive theoretical guarantees in terms of competitive ratio. In this section, for simplicity, we refer an offline optimal algorithm as *OPT*.

Although we have so far discussed the prediction error more intuitively, it requires a quantitative measure in order to analyze the performance of an algorithm relying on predic-

[1] Recall that LQD is close to optimal with a competitive ratio of 1.707.

[2] Our source code will be made publicly available online.

Algorithm 1: CREDENCE

Input : Packet arrivals σ,
Drop predictions $\phi'(\sigma)$

1 **procedure** ARRIVAL $(\sigma(t))$**:**
2 **for** *each packet* $p \in \sigma(t)$ **do**
3 Let i be the destination queue for the packet p
4 UPDATETHRESHOLD$(i, arrival)$
 ▷ *Guarantees N-competitiveness*
 Let j be the longest queue
5 **if** $q_j(t) < \frac{B}{N}$ **then**
 $q_i(t) \leftarrow q_i(t) + 1$ ▷ Accept
 Continue to next packet
 ▷ *Enables* 1.707 η*-competitiveness*
 if $q_i(t) < T_i(t)$ **then**
 if $Q(t) < B$ **then**
 $drop$ = GETPREDICTION()
 if $drop$ **then**
6 ▷ Drop
 else
 $q_i(t) \leftarrow q_i(t) + 1$ ▷ Accept
 else
 ▷ Drop
7 **procedure** DEPARTURE (i)**:**
8 **if** $q_i(t) > 0$ **then**
9 $q_i(t) \leftarrow q_i(t) - 1$ ▷ Drain one packet
10 UPDATETHRESHOLD$(i, departure)$
11 **procedure** UPDATETHRESHOLD $(i, event)$**:**
12 **if** *event* = *arrival* **then**
 ▷ *Thresholds are treated as LQD queue lengths*
 if $\Gamma(t) = B$ **then** ▷ Sum of thresholds
 Let $T_j(t)$ be the highest threshold
 $T_j(t) \leftarrow T_j(t) - 1$ ▷ Decrease
13 $T_i(t) \leftarrow T_i(t) + 1$ ▷ Increase
 else
 $T_i(t) \leftarrow T_i(t) + 1$ ▷ Increase
 $\Gamma(t) \leftarrow \Gamma(t) + 1$
14 **if** *event* = *departure* **then**
15 **if** $T_i(t) > 0$ **then**
16 $T_i(t) \leftarrow T_i(t) - 1$ ▷ Decrease
17 $\Gamma(t) \leftarrow \Gamma(t) - 1$

tions. There are two important considerations in defining a suitable error function. First, following the literature, an error function must be independent of the state and actions of our algorithm, so that we can train a predictor without considering all possible states of the algorithm [27]. Second, it is desirable that the performance of our algorithm can be related to the error function in an uncomplicated manner. Taking these into consideration, we define our error function in Definition 1. Our definition captures the prediction error in terms of the performance of LQD (push-out) and the performance of an algorithm *FollowLQD*. Here, *FollowLQD* (Algorithm 2 in Appendix B) is a deterministic drop-tail algorithm (without predictions) with thresholds similar to CREDENCE.

Definition 1 (Error function)**.** *Let LQD*(σ) *and FollowLQD*(σ) *denote the total number of packets transmitted by the online push-out algorithm LQD and the online drop-tail algorithm FollowLQD over the arrival sequence* σ*. Let* ϕ *denote the sequence indicating drop by LQD for each packet in the arrival sequence* σ*. Let* ϕ' *denote the sequence of drops predicted by the machine-learned oracle. Let* ϕ'_{TP}*,* ϕ'_{FP}*,* ϕ'_{TN}*, and* ϕ'_{FN} *denote the sequence of true positive, false positive, true negative and false negative predictions for the arrival sequence* σ*. We define the error function* $\eta(\phi, \phi')$ *as follows:*

$$\eta(\phi, \phi') = \frac{LQD(\sigma)}{FollowLQD\left(\sigma - \phi'_{TP} - \phi'_{FP}\right)} \tag{1}$$

Using Definition 1, we analyze the throughput of CREDENCE over an entire packet arrival sequence σ based on the predictions ϕ'. In fact, our error function is upper bounded by an intuitive closed form expression, in terms of the number of true and false predictions, as follows, that can be easily computed:[3]

$$\eta(\phi, \phi') \leq \frac{\phi'_{TN} + \phi'_{FP}}{\phi'_{TN} - \min\left((N-1) \cdot \phi'_{FN}, \phi'_{TN}\right)}$$

The upper bound of our error function indicates intuitively that *(i)* the error decreases as the total number of true negative predictions dominate the total false predictions, *(ii)* the error increases with each false positive prediction and *(iii)* the error increases with each false negative with a larger weight. Lemma 1 states the relation between the throughput of CREDENCE, throughput of *LQD* and the prediction error.

Lemma 1. *The total number of packets transmitted by* CREDENCE *for an arrival sequence* σ*, a drop sequence* ϕ *by LQD and the predicted drop sequence* ϕ' *is given by*

$$\text{CREDENCE}(\sigma) \geq \frac{LQD(\sigma)}{\underbrace{\eta(\phi, \phi')}_{error}} \tag{2}$$

Equation 2 shows that the throughput of CREDENCE reaches closer to (moves away from) *LQD* as the prediction error becomes smaller (larger). We present a sketch of our proof here. Our full proof appears in Appendix C. We begin by analyzing the drops incurred by CREDENCE based on the drop criterion described in §3.2. We argue that for every true positive and false positive predictions, there is at most one drop by CREDENCE. All other drops incurred by CREDENCE are due to the thresholds. Using these observations, we show that CREDENCE transmits at least the number of

[3] We prove our upper bound in Theorem 2 (Appendix C).

packets transmitted by *FollowLQD* over the arrival sequence $\sigma - \phi'_{TP} - \phi'_{FP}$. This leads us to Equation 2.

Recall that CREDENCE bypasses the drop criterion and accepts packets through a safeguard condition under certain cases (see §3.2). Based on this, we obtain another bound for the throughput of CREDENCE in Lemma 2, which is independent of the prediction error.

Lemma 2. CREDENCE *transmits at least* $\frac{1}{N}$ *times the number of packets transmitted by an offline optimal algorithm OPT i.e.,* $\text{CREDENCE}(\sigma) \geq \frac{1}{N} \cdot OPT(\sigma)$.

Lemma 2 shows that irrespective of the prediction error (even under large error), CREDENCE can always transmit at least $\frac{1}{N}$ fraction of the packets transmitted by an optimal solution. Our proof of Lemma 2 is based on the fact that upon a drop by CREDENCE, there is at least one queue with length $\frac{B}{N}$ (the safeguard condition). As a result, for every B packets transmitted by OPT, there are at least $\frac{B}{N}$ number of packets transmitted by CREDENCE over the arrival sequence σ. This leads us to the bound expressed in Lemma 2.

Finally, using the above two results, we prove the competitive ratio of CREDENCE as a function of the prediction error. CREDENCE's competitive ratio satisfies the three desirable properties: 2-consistent, N-robust and exhibits smoothness.

Theorem 1. *The competitive ratio of* CREDENCE *grows linearly from* 1.707 *to N based on the prediction error* $\eta(\phi, \phi')$*, where N is the number of ports,* ϕ *is the drop sequence of LQD and* ϕ' *is the predicted sequence of drops i.e., the competitive ratio is at most* $\min(1.707\ \eta(\phi, \phi'), N)$.

Our proof follows from Lemmas 1 and 2 (see Appendix C). Theorem 1 essentially shows how CREDENCE's competitive ratio in terms of throughput improves from N to 1.707 as the prediction error (Definition 1) decreases. We note that our analysis compares an algorithm against an optimal offline algorithm over a fixed packet arrival sequence. This allows us to analyze the competitive ratio via an error function defined over the corresponding arrival sequence. However, real-world traffic is responsive in nature due to congestion control and packet retransmissions. Although we have used η as our error function to express the competitiveness of CREDENCE, in our evaluation (§4), we compare CREDENCE with state-of-the-art approaches under realistic datacenter workloads and we also present the quality of our predictions using more natural error functions that are widely used for machine learning models.

3.4 Practicality of CREDENCE

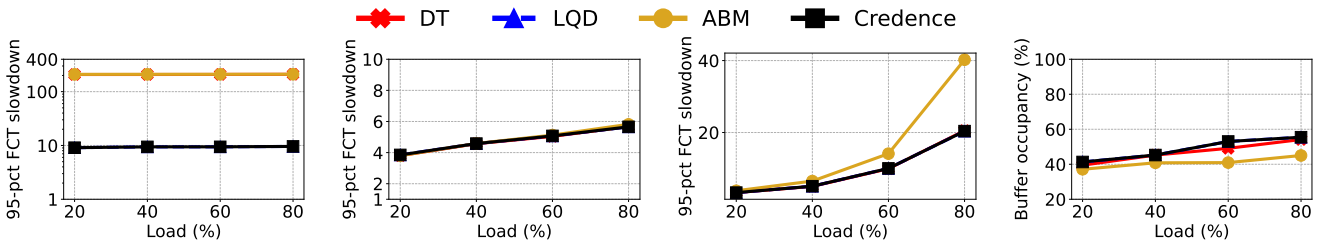
CREDENCE's algorithm itself is simple and close to complexity of the longest queue drop (push-out). However, the machine-learned oracle producing the predictions adds additional complexity in order to deploy CREDENCE on switches. Overall, there are three main parts of CREDENCE that contribute to additional complexity in terms of memory and computation: *(i)* finding the longest queue (and its threshold), *(ii)* remembering thresholds and *(iii)* obtaining predictions.

Finding the longest queue (and its threshold): For every packet arrival, CREDENCE requires finding the longest queue for the safeguard condition described in §3.2. Additionally, CREDENCE requires finding the largest threshold during the threshold updates upon every packet arrival. The maximum value search operation has a run-time complexity of $O(N)$, where N is the number of ports. Note that typical datacenter switches have a relatively small number of ports e.g., 64 ports in Broadcom Tomahawk4 [14]. Prior work in the context of LQD proposes an approximation to further reduce the complexity of finding the longest queue [47] to $O(1)$. The average case complexity can further be reduced by only maintaining the list of queue lengths (and their thresholds) that are larger than $\frac{B}{N}$. This is sufficient since the safeguard condition checks whether the longest queue is less than $\frac{B}{N}$, which is the same as checking that no queue is longer than $\frac{B}{N}$. Similarly, the largest threshold search during the threshold updates is only triggered when the buffer is full. In this case, the longest queue must be at least $\frac{B}{N}$. Given that switches are becoming more and more computationally capable, we believe that a basic function such as finding the maximum value in a small list is feasible to implement within the available resources.

Thresholds memory: In contrast to existing threshold-based algorithms, CREDENCE's thresholds depend on their previous value i.e., thresholds must be remembered. As a result, CREDENCE adds a small memory overhead of $O(N)$ for the thresholds. The threshold calculations are in fact much simpler than existing schemes and do not add any further computational complexity since CREDENCE only requires adding and subtracting the threshold values by the packet size.

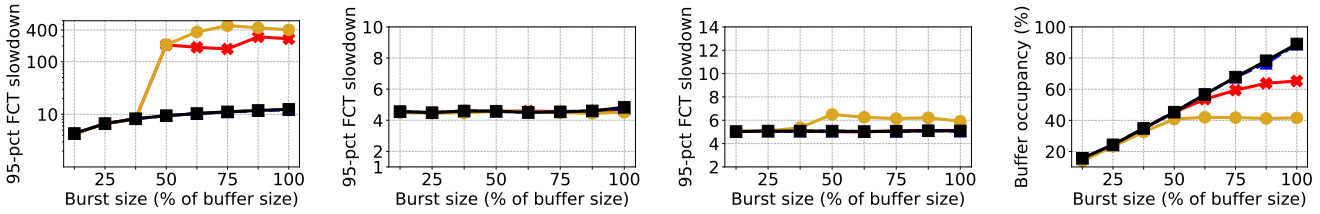
Predictions: Our prediction model (drop or accept) essentially boils down to binary classification problem. To this end, numerous ML techniques exist ranging from linear classifiers to more advanced neural networks. In view of practicality, we consider random forests as they are implementable in programmable hardware [4, 15]. In order to reduce the prediction latency, we also limit the number of trees and the maximum depth of our trained random forest model. We find that, even a model trained with a maximum depth of four, and as low as four to eight trees achieves reasonable prediction error (precision ≈ 0.65). Further, to reduce the complexity of the model, we also limit the number of features to four: queue length, total shared buffer occupancy and their corresponding moving averages (exponentially weighted) over one round-trip time.

The fundamental blocks required for CREDENCE are all individually practical in today's hardware. Unfortunately, modifying the buffer sharing algorithm and integrating it with predictions requires switch vendor support. Even in programmable switches, the traffic manager is merely a blackbox that implements Dynamic Thresholds with a single parameter exposed to the user. Given the superior performance of CREDENCE (§4), we wish to gain attention from switch vendors to discuss further on the implementation of CREDENCE.



(a) Incast flows (b) Short flows (c) Long flows (d) Shared buffer occupancy

Figure 6: Performance of CREDENCE across various loads of websearch workload and incast workload at a burst size 50% of the buffer size, with DCTCP as the transport protocol. As the load increases, ABM penalizes long flows. DT and ABM are unable to absorb bursts of size 50% of the buffer size. CREDENCE achieves superior burst absorption and does not penalize long flows.



(a) Incast flows (b) Short flows (c) Long flows (d) Shared buffer occupancy

Figure 7: Performance of CREDENCE across various burst sizes of incast workload and websearch workload at 40% load, with DCTCP as the transport protocol. At small burst sizes, DT and ABM achieve similar performance compared to CREDENCE but as the burst size increases, CREDENCE outperforms DT and ABM in terms of FCTs for incast flows (burst absorption).

4 Evaluation

We evaluate the performance of CREDENCE and compare it against state-of-the-art buffer sharing algorithms in the context of datacenter networks. Our evaluation aims at answering three main questions:

(Q1) *Does* CREDENCE *improve the burst absorption?*
Our evaluation shows that CREDENCE significantly improves the burst absorption capabilities of switches. We find that CREDENCE improves the 95-percentile flow completion times for incast flows by up to 95.4% compared to Dynamic Thresholds (DT) and by up to 96.9% compared to ABM.

(Q2) *Can* CREDENCE *improve the flow completion times for short flows as well as long flows?*
We find that CREDENCE performs similar to existing approaches in terms of 95-percentile flow completion times for short flows and improves upon ABM by up to 22% correspondingly for long flows.

Q3 *How does prediction error impact the performance of* CREDENCE *in terms of flow completion times?*
We increase the error of our prediction by artificially flipping the predictions with a probability. As the probability increases (error increases), we find that CREDENCE sustains performance up to 0.01 probability and smoothly degrades in performance beyond 0.01.

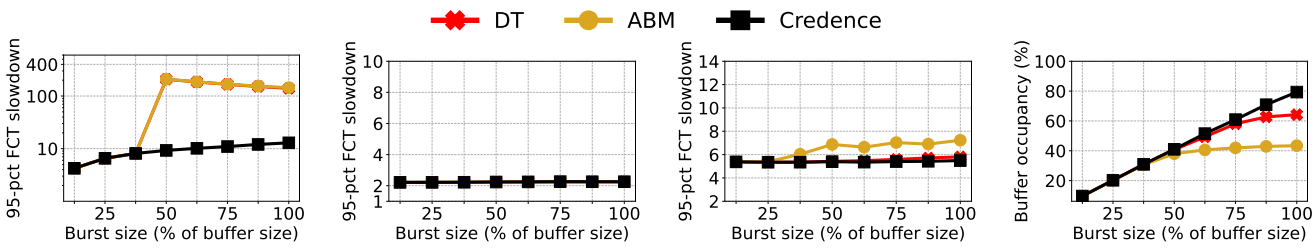
4.1 Setup

Our evaluation is based on packet-level simulations in NS3 [40]. We embed a Python interpreter within NS3 using pybind11 [43] in order to obtain predictions from a random forest model trained with scikit-learn [45].

Topology: We consider a leaf-spine topology with 256 servers organized into 4 spines and 16 leaves. Each link has a propagation delay of 3μs leading to a round-trip-time of 25.2μs. The capacity is set to 10Gbps for all the links leading to 4 : 1 oversubscription similar to prior works [1, 3, 44]. All the switches in our topology have 5.12KB buffer-per-port-per-Gbps similar to Broadcom Tomahawk [14].

Workloads: We generate traffic using websearch [6] flow size distribution that is based on measurements from real-world datacenter workloads. We vary the load on the network in the range 20-80%. We additionally generate traffic using a synthetic incast workload similar to prior work [1]. Our incast workload mimics the query-response behavior of a distributed file storage system where each query results in a bursty response from multiple servers. We set the query request rate to 2 per second from each server, and we vary the burst size in the range 10-100% of the switch buffer size. We use DCTCP [6] and PowerTCP [3] as transport protocols.

Comparisons & metrics: We compare CREDENCE with Dynamic Thresholds [18] (the default algorithm in datacenter switches), ABM [1] and LQD (push-out). Hereafter, we refer to Dynamic Thresholds as DT. We report four performance metrics: 95-percentile flow completion times for short flows ($\leq$ 100KB), incast flows (incast workload), long flows ($\geq$ 1MB), and the 99-percentile shared buffer occupancy. We present the CDFs of flow completion times in Appendix D.



(a) Incast flows (b) Short flows (c) Long flows (d) Shared buffer occupancy

Figure 8: Performance of CREDENCE across various burst sizes of incast workload and websearch workload at 40% load, with PowerTCP as the transport protocol. Even with advanced congestion control, DT and ABM only benefit in terms of FCTs for long flows, but CREDENCE outperforms in terms of FCTs for incast flows (burst absorption) as well as FCTs for long flows.

Predictions: We collect packet-level traces from each switch in our topology when using LQD (push-out) as the buffer sharing algorithm. Each trace consists of five values corresponding to each packet arrival: *(i)* queue length, *(ii)* average queue length, *(iii)* shared buffer occupancy, *(iv)* average shared buffer occupancy and *(v)* accept (or drop). We train a random forest classifier using queue length and shared buffer occupancy as features and the model predicts packet drops. We set the maximum depth for each tree to 4 in view of practicality. At a train-test split 0.6 of our LQD trace, based on our parameter sweep across the number of trees used for our classifier (Figure 15 in Appendix D), we set the number of trees to 4. We observe that the quality of our predictions does not improve significantly beyond four trees in our datasets. This gives us an accuracy of 0.99, error score $\frac{1}{\eta}$ 0.99[4] (inverse of our error function based on Definition 1), precision of 0.65, recall of 0.35 and F1 score of 0.45. We defer the definitions of these prediction scores to Appendix C as they are standard in the literature. We train our model with an LQD trace corresponding to websearch workload at 80% load, and a burst size of 75% buffer size for the incast workload, using DCTCP as the transport protocol. We use the same trained model in all our evaluations. We ensure that our test scenarios are different from the training dataset by using different random seeds in addition to different traffic conditions (different loads and different burst sizes) in each experiment in our evaluation.

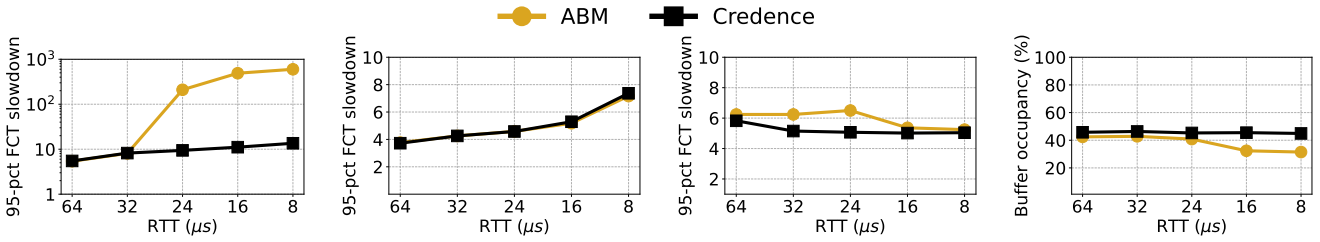
Configuration: CREDENCE is parameter-less, and it takes input from an oracle (described above) that predicts packet drops. We set $\alpha = 0.5$ for DT and ABM similar to prior work [1]. ABM uses $\alpha = 64$ for all the packets which arrive during the first round-trip-time [1]. We configure DCTCP according to [6] and PowerTCP according to [3].

4.2 Results

CREDENCE significantly improves burst absorption: In Figure 6a, using DCTCP as the transport protocol, we generate websearch traffic across various loads in the range 20-80% and generate incast traffic with a burst size of 50% buffer size. We observe that CREDENCE performs close to the optimal performance of LQD. CREDENCE improves the 95-percentile flow complete times for incast flows by 95.50% compared to DT, and by 95.53% compared to ABM. In Figure 7a, we set the load of websearch traffic at 40% and vary the burst size for incast workload in the range 10-100% buffer size. CREDENCE performs similar to DT and ABM for small burst sizes. As the burst size increases, CREDENCE improves the 95-percentile flow completion times for incast workload by 95% on average compared to DT, and by 96.92% on average compared to ABM. In Figure 8a, even when using PowerTCP as the transport protocol, we see that CREDENCE improves the 95-percentile flow completion times for incast flow by 93.27% on average compared to DT and by 93.36% on average compared to ABM. In terms of burst absorption, both DT and ABM are drop-tail algorithms, hence they face the drawbacks discussed in §2.2. CREDENCE relies on predictions and unlocks the performance of LQD (push-out) as shown by our results in Figure 6a and Figure 7a.

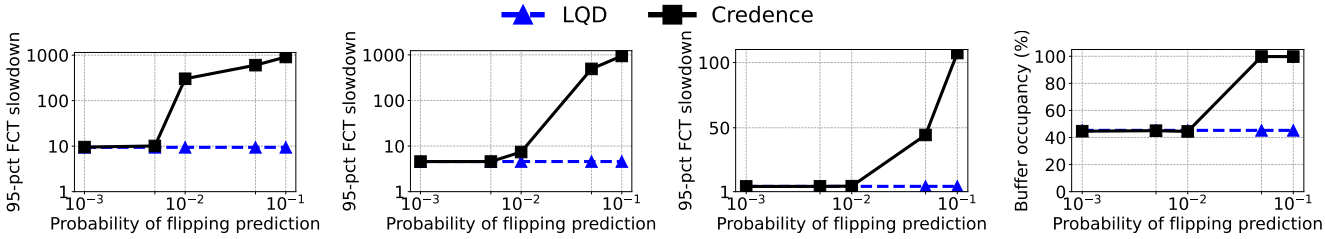
CREDENCE improves long flows FCTs: CREDENCE not only improves the burst absorption but also improves the flow completion times for long flows. In Figure 6c, at 50% burst size for incast workload and across various loads of websearch workload, we observe that CREDENCE performs similar to DT in terms of 95-percentile flow completion times for long flows and improves upon ABM on average by 28.49%. At 80% load, CREDENCE improves upon ABM by 49.34%. Across various burst sizes, and at 40% load of websearch workload, we observe from Figure 7c that CREDENCE improves upon ABM by up to 22.02% and by 12.02% on average. With PowerTCP as the transport protocol (Figure 8c), CREDENCE improves the 95-percentile flow completion times for long flows by 3.31% on average compared to DT and by 17.35% compared to ABM. At a burst size of 100% buffer size, CREDENCE improves the flow completion times by 5.49% compared to DT and by 24.09% compared to ABM. As described in §2.2, drop-tail algorithms such as DT and ABM cannot effectively navigate proactive and reactive drops, resulting in throughput loss i.e., high flow completion times for long flows. In contrast, predictions guide CREDENCE to effectively navi-

[4]The high values of accuracy and our error score $\frac{1}{\eta}$ are attributed to the dataset being skewed i.e., congestion is not persistent.



(a) Incast flows (b) Short flows (c) Long flows (d) Shared buffer occupancy

Figure 9: ABM is sensitive to RTT and performs significantly worse compared to CREDENCE at low RTTs. At high RTTs, ABM performs similar to CREDENCE.



(a) Incast flows (b) Short flows (c) Long flows (d) Shared buffer occupancy

Figure 10: Even though the predictions from our random forest classifier are intentionally flipped (to increase error), CREDENCE performs close to LQD up to 0.005 flipping probability but smoothly diverges from LQD at 0.01 flipping probability.

gate proactive and reactive drops. This allows CREDENCE to achieve better flow completion times even for long flows.

CREDENCE does not waste buffer resources: In anticipation of future burst arrivals, both DT and ABM buffer resources. We show the 99.99-percentile buffer occupancies[5] in Figure 6d for various loads of websearch workload and at a burst size of 50% buffer size for incast workload. We observe that, DT (ABM) utilizes 3.77% (18.68%) lower buffer space on average compared to CREDENCE, at the cost of increased flow completion times even for long flows. Even as the burst size increases (Figure 7d), DT and ABM are unable to efficiently utilize the buffer space. In contrast, CREDENCE efficiently utilizes the available buffer space as the burst size increases, improving burst absorption without sacrificing flow completion times for long flows.

ABM is sensitive to RTT: Although ABM is expected to outperform DT, our evaluation results especially in terms of flow completion times for incast flows contradict the results presented in [1]. We ran several simulations varying all the parameters in our setup in order to better understand the performance of ABM. We found that ABM is in fact sensitive to round-trip-time (RTT). We vary the base RTT of our topology in Figure 9 and compare CREDENCE with ABM. At high RTTs, we observe that ABM performs close to CREDENCE, but degrades in performance as RTT decreases. Specifically, at 8μs RTT, ABM performs 97.73% worse compared to CREDENCE in terms of flow completion times for incast flows.

[5]DT, ABM and CREDENCE have similar tail occupancies (100-percentile) that occurs at rare congestion events in our simulations.

Although ABM achieves on-par flow completion times for short flows, we observe that ABM degrades in flow completion times for long times as well as under-utilizes the buffer as RTT decreases. The poor performance of ABM at low RTTs is due to the fact that ABM prioritizes the first RTT packets and considers the rest of the traffic as steady-state traffic. However, it is not uncommon that datacenter switches experience bursts for several RTTs [20]. Further, congestion control algorithms require multiples RTTs to converge to steady-state. In contrast, CREDENCE is parameter-less and does not make such assumptions. CREDENCE performs significantly better than existing approaches even with an off-the-shelf machine-learned predictor with a simple model.

CREDENCE gradually degrades with prediction error: Our random forest classifier that we used in our evaluations so far, has a precision close to 0.65. In order to evaluate the performance of CREDENCE with even worse prediction error, we artificially introduce error by flipping every prediction obtained from our random forest classifier with a certain probability. We consider LQD (push-out) as a baseline since CREDENCE is expected to perform close to LQD and degrade as the prediction error grows large. Figure 10 presents our evaluation results, under websearch workload at 40% and burst size 50% of the buffer size for incast workload. At 0.001 flipping probability, CREDENCE performs close to LQD. However, at 0.01 flipping probability CREDENCE starts to diverge from LQD and gets significantly worse at 0.1 flipping probability. Figure 10 gives practical insights into smoothness of CREDENCE in addition to our analysis.

5 Related Work

The buffer sharing problem has been widely studied for many decades. Research works in the literature range from push-out as well as drop-tail algorithms tailored for ATM networks [17, 18, 32, 47, 48] to more recent drop-tail algorithms tailored for datacenter networks [1, 2, 9, 26, 46]. While we focus on the buffer sharing problem in this paper, several related but orthogonal approaches also tackle buffer problems in datacenter networks e.g., end-to-end congestion control [3, 6, 16, 23, 33, 34], AQM [19, 39, 41], scheduling [7, 25], packet deflection [50] and load-balancing [5, 21, 29]. These approaches aim at reducing congestion events and the overall buffer requirements, but they cannot fundamentally address buffer contention across multiple switch ports sharing the same buffer. Research on algorithms with predictions for various problems has recently been an active field of research [28, 35--38, 42] but ours is the first approach tackling the buffer sharing problem with predictions. Ongoing research efforts show the feasibility of deploying machine-learned predictions in the network data plane [4, 15, 49].

6 Future Research Directions

CREDENCE is the first approach showing the performance benefits and guarantees by augmenting buffer sharing algorithms with predictions. We believe that this paper opens several interesting avenues for future work both in systems and theory. In this section, we discuss some of the future work directions to push approaches such as CREDENCE to be deployed in the real-world (§6.1), as well as to improve the performance guarantees offered by such approaches (§6.2).

6.1 Systems for In-Network Predictions

In this paper, we show how predictions can improve the performance of drop-tail buffer sharing. Many interesting systems research questions remain in order to integrate buffer sharing and predictions in the network data plane.

Training the model: Achieving consistent performance requires lowering the prediction error. The training phase of the model is a critical step towards reaching optimal buffer sharing. Although, for simplicity, we have used only four features to train our random forest model, it would be interesting to study the tradeoff that may arise between prediction error and the complexity of the model both in space and time. Further, the trained model must be simple enough that fits within the resources available in the data plane. Developing such trained models is an important step forward.

Deploying the model: Recent works propose practical implementations for in-network machine-learning models e.g., in the context of traffic classification [15]. P4 implementation of a model that predicts drops would enhance not only the practicality but also stimulate further research to design algorithms with performance guarantees better than CREDENCE.

Alternative predictions: As mentioned in §2.3, there are several different prediction models that can be considered for the buffer sharing problem. For instance, instead of predicting the drops, an oracle could predict packet arrivals just for a tiny window of the near future. Systems research on studying the practicality and deployability of different prediction models is a valuable future direction that would better guide the design of algorithms with predictions for the buffer sharing problem.

Understanding push-out complexity: While push-out algorithms raised much interest initially, over the last years, research on this approach has been less active. We believe this is partly due to the lack of support from switch vendors. It is an open question how the complexity of obtaining drop predictions and the complexity of push-out fare against each other. Although we focused on augmenting drop-tail algorithms with predictions, we believe that our approach of using predictions has much potential also in other types of buffer algorithms. While switch vendors may be better informed about the complexity of push-out buffers, an understanding of this complexity in the scientific community is much needed in order to navigate the complexity vs performance spectrum.

6.2 Theory for Performance Guarantees

We believe the performance guarantees offered by CREDENCE can be improved in the future. Further, considering packet priorities and traffic classes in the competitive analysis is an open question.

Improving consistency and robustness: An open question is whether an algorithm could be designed to improve the competitive ratio under perfect predictions (consistency) better than 1.707, while also improving the ratio under large error (robustness) better than N. Further research in this direction would enable a better understanding whether a consistency-robustness tradeoff exists for the buffer sharing problem.

Competitive analysis with packet priorities: Literature in theory considers that all packets are of same priority in the context of competitive analysis. However, it is well-known that preferential treatment of packets has various performance benefits especially in terms of flow completion times when short flow packets are prioritized. To this end, developing analysis techniques for such a setup is an interesting future research direction.

7 Conclusion

We presented CREDENCE, the first buffer sharing algorithm augmented with predictions that not only reaches close to optimal performance given low prediction error but also guarantees performance with arbitrarily large prediction error, while maintaining smoothness. We analytically proved our claims and our evaluations show the superior performance of CREDENCE even with an off-the-shelf machine-learned predictor, compared to the state-of-the-art buffer sharing algorithms. The building blocks required for CREDENCE are all individually practical in today's hardware. In future, we plan to pursue switch vendors to further discuss the integration of predictions with buffer sharing algorithm in hardware.

Acknowledgements

This work is part of a project that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme, consolidator project Self-Adjusting Networks (AdjustNet), grant agreement No. 864228, Horizon 2020, 2020-2025.

References

[1] Vamsi Addanki, Maria Apostolaki, Manya Ghobadi, Stefan Schmid, and Laurent Vanbever. Abm: Active buffer management in datacenters. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, page 36–52, New York, NY, USA, 2022. Association for Computing Machinery.

[2] Vamsi Addanki, Wei Bai, Stefan Schmid, and Maria Apostolaki. Reverie: Low pass filter-based switch buffer sharing for datacenters with rdma and tcp traffic. In *21th USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, Santa Clara, CA, 2024. USENIX Association.

[3] 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)*, pages 51--70, Renton, WA, April 2022. USENIX Association.

[4] Aristide Tanyi-Jong Akem, Michele Gucciardo, and Marco Fiore. Flowrest: Practical flow-level inference in programmable switches with random forests. In *IEEE INFOCOM 2023 - IEEE Conference on Computer Communications*, pages 1--10, 2023.

[5] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, and George Varghese. Conga: Distributed congestion-aware load balancing for datacenters. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, page 503–514, New York, NY, USA, 2014. Association for Computing Machinery.

[6] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, page 63–74, New York, NY, USA, 2010. Association for Computing Machinery.

[7] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. Pfabric: Minimal near-optimal datacenter transport. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, page 435–446, New York, NY, USA, 2013. Association for Computing Machinery.

[8] Antonios Antoniadis, Matthias Englert, Nicolaos Matsakis, and Pavel Veselý. Breaking the Barrier Of 2 for the Competitiveness of Longest Queue Drop. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, volume 198 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 17:1--17:20, Dagstuhl, Germany, 2021. Schloss Dagstuhl -- Leibniz-Zentrum für Informatik.

[9] Maria Apostolaki, Laurent Vanbever, and Manya Ghobadi. Fab: Toward flow-aware buffer sharing on programmable switches. In *Proceedings of the 2019 Workshop on Buffer Sizing*, BS '19, New York, NY, USA, 2020. Association for Computing Machinery.

[10] Wei Bai, Shanim Sainul Abdeen, Ankit Agrawal, Krishan Kumar Attre, Paramvir Bahl, Ameya Bhagat, Gowri Bhaskara, Tanya Brokhman, Lei Cao, Ahmad Cheema, Rebecca Chow, Jeff Cohen, Mahmoud Elhaddad, Vivek Ette, Igal Figlin, Daniel Firestone, Mathew George, Ilya German, Lakhmeet Ghai, Eric Green, Albert Greenberg, Manish Gupta, Randy Haagens, Matthew Hendel, Ridwan Howlader, Neetha John, Julia Johnstone, Tom Jolly, Greg Kramer, David Kruse, Ankit Kumar, Erica Lan, Ivan Lee, Avi Levy, Marina Lipshteyn, Xin Liu, Chen Liu, Guohan Lu, Yuemin Lu, Xiakun Lu, Vadim Makhervaks, Ulad Malashanka, David A. Maltz, Ilias Marinos, Rohan Mehta, Sharda Murthi, Anup Namdhari, Aaron Ogus, Jitendra Padhye, Madhav Pandya, Douglas Phillips, Adrian Power, Suraj Puri, Shachar Raindel, Jordan Rhee, Anthony Russo, Maneesh Sah, Ali Sheriff, Chris Sparacino, Ashutosh Srivastava, Weixiang Sun, Nick Swanson, Fuhou Tian, Lukasz Tomczyk, Vamsi Vadlamuri, Alec Wolman, Ying Xie, Joyce Yom, Lihua Yuan, Yanzhao Zhang, and Brian Zill. Empowering azure storage with RDMA. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 49--67, Boston, MA, April 2023. USENIX Association.

[11] 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*, 29(2):489--502, 2021.

[12] Ivan A. Bochkov, Alex Davydow, Nikita Gaevoy, and Sergey I. Nikolenko. New competitiveness bounds

for the shared memory switch. *CoRR*, abs/1907.04399, 2019.

[13] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. 1998.

[14] Broadcom. StrataXGS® Switch Solutions. https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs.

[15] Coralie Busse-Grawitz, Roland Meier, Alexander Dietmüller, Tobias Bühler, and Laurent Vanbever. pforest: In-network inference with random forests. *arXiv preprint arXiv:1909.05680*, 2019.

[16] Qizhe Cai, Mina Tahmasbi Arashloo, and Rachit Agarwal. Dcpim: Near-optimal proactive datacenter transport. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, page 53–65, New York, NY, USA, 2022. Association for Computing Machinery.

[17] J.W. Causey and H.S. Kim. Comparison of buffer allocation schemes in atm switches: complete sharing, partial sharing, and dedicated allocation. In *Proceedings of ICC/SUPERCOMM'94 - 1994 International Conference on Communications*, pages 1164--1168 vol.2, 1994.

[18] A.K. Choudhury and E.L. Hahne. Dynamic queue length thresholds for shared-memory packet switches. *IEEE/ACM Transactions on Networking*, 6(2):130--140, 1998.

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

[20] Ehab Ghabashneh, Yimeng Zhao, Cristian Lumezanu, Neil Spring, Srikanth Sundaresan, and Sanjay Rao. A microscopic view of bursts, buffer contention, and loss in data centers. In *Proceedings of the 22nd ACM Internet Measurement Conference*, IMC '22, page 567–580, New York, NY, USA, 2022. Association for Computing Machinery.

[21] Soudeh Ghorbani, Zibin Yang, P. Brighten Godfrey, Yashar Ganjali, and Amin Firoozshahian. Drill: Micro load balancing for low-latency data center networks. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, page 225–238, New York, NY, USA, 2017. Association for Computing Machinery.

[22] Michael H. Goldwasser. A survey of buffer management policies for packet switches. *SIGACT News*, 41(1):100–128, mar 2010.

[23] 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.

[24] Ellen L. Hahne, Alexander Kesselman, and Yishay Mansour. Competitve buffer management for shared-memory switches. In *Proceedings of the Thirteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '01, page 53–58, New York, NY, USA, 2001. Association for Computing Machinery.

[25] Chi-Yao Hong, Matthew Caesar, and P. Brighten Godfrey. Finishing flows quickly with preemptive scheduling. *SIGCOMM Comput. Commun. Rev.*, 42(4):127–138, aug 2012.

[26] Sijiang Huang, Mowei Wang, and Yong Cui. Traffic-aware buffer management in shared memory switches. *IEEE/ACM Transactions on Networking*, 30(6):2559--2573, 2022.

[27] Sungjin Im, Ravi Kumar, Mahshid Montazer Qaem, and Manish Purohit. Non-clairvoyant scheduling with predictions. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '21, page 285–294, New York, NY, USA, 2021. Association for Computing Machinery.

[28] Sungjin Im, Benjamin Moseley, Chenyang Xu, and Ruilong Zhang. Online dynamic acknowledgement with learned predictions. *arXiv preprint arXiv:2305.18227*, 2023.

[29] Naga Katta, Mukesh Hira, Changhoon Kim, Anirudh Sivaraman, and Jennifer Rexford. Hula: Scalable load balancing using programmable data planes. In *Proceedings of the Symposium on SDN Research*, SOSR '16, New York, NY, USA, 2016. Association for Computing Machinery.

[30] Alexander Kesselman and Yishay Mansour. Harmonic buffer management policy for shared memory switches. *Theoretical Computer Science*, 324(2):161--182, 2004. Online Algorithms: In Memoriam, Steve Seiden.

[31] Koji Kobayashi, Shuichi Miyazaki, and Yasuo Okabe. A tight bound on online buffer management for two-port shared-memory switches. In *Proceedings of the Nineteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '07, page 358–364, New York, NY, USA, 2007. Association for Computing Machinery.

[32] S. Krishnan, A.K. Choudhury, and F.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 vol.1, 1999.

[33] Gautam Kumar, Nandita Dukkipati, Keon Jang, Hassan M. G. Wassel, Xian Wu, Behnam Montazeri, Yaogong Wang, Kevin Springborn, Christopher Alfeld, Michael Ryan, David Wetherall, and Amin Vahdat. 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*, SIGCOMM '20, page 514–528, New York, NY, USA, 2020. Association for Computing Machinery.

[34] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, and Minlan Yu. Hpcc: High precision congestion control. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, page 44–58, New York, NY, USA, 2019. Association for Computing Machinery.

[35] Michael Mitzenmacher. A model for learned bloom filters and optimizing by sandwiching. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[36] Michael Mitzenmacher. Scheduling with predictions and the price of misprediction. *arXiv preprint arXiv:1902.00732*, 2019.

[37] Michael Mitzenmacher. Queues with small advice. In *Proceedings of the 2021 SIAM Conference on Applied and Computational Discrete Algorithms (ACDA21)*, pages 1--12. SIAM, 2021.

[38] Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with predictions. *Commun. ACM*, 65(7):33–35, jun 2022.

[39] Kathleen Nichols and Van Jacobson. Controlling queue delay: A modern aqm is just one piece of the solution to bufferbloat. *Queue*, 10(5):20–34, may 2012.

[40] ns-3. Network Simulator. https://www.nsnam.org/.

[41] 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.

[42] Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ml predictions. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[43] Pybind11. Seamless operability between C++11 and Python. https://pybind11.readthedocs.io/en/stable/.

[44] Ahmed Saeed, Varun Gupta, Prateesh Goyal, Milad Sharif, Rong Pan, Mostafa Ammar, Ellen Zegura, Keon Jang, Mohammad Alizadeh, Abdul Kabbani, and Amin Vahdat. Annulus: A dual congestion control loop for datacenter and wan traffic aggregates. SIGCOMM '20, page 735–749, New York, NY, USA, 2020. Association for Computing Machinery.

[45] scikit-learn. Machine Learning in Python. https://scikit-learn.org/stable/.

[46] Danfeng Shan, Wanchun Jiang, and Fengyuan Ren. Analyzing and enhancing dynamic threshold policy of data center switches. *IEEE Transactions on Parallel and Distributed Systems*, 28(9):2454--2470, 2017.

[47] B. Suter, T.V. Lakshman, D. Stiliadis, and A.K. Choudhury. Buffer management schemes for supporting tcp in gigabit routers with per-flow queueing. *IEEE Journal on Selected Areas in Communications*, 17(6):1159--1169, 1999.

[48] Guo-Liang Wu and J.W. Mark. A buffer allocation scheme for atm networks: complete sharing based on virtual partition. *IEEE/ACM Transactions on Networking*, 3(6):660--670, 1995.

[49] Zhaoqi Xiong and Noa Zilberman. Do switches dream of machine learning? toward in-network classification. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*, HotNets '19, page 25–33, New York, NY, USA, 2019. Association for Computing Machinery.

[50] Kyriakos Zarifis, Rui Miao, Matt Calder, Ethan Katz-Bassett, Minlan Yu, and Jitendra Padhye. Dibs: Just-in-time congestion mitigation for data centers. In *Proceedings of the Ninth European Conference on Computer Systems*, EuroSys '14, New York, NY, USA, 2014. Association for Computing Machinery.

[51] Qiao Zhang, Vincent Liu, Hongyi Zeng, and Arvind Krishnamurthy. High-resolution measurement of data center microbursts. In *Proceedings of the 2017 Internet Measurement Conference*, IMC '17, page 78–85, New York, NY, USA, 2017. Association for Computing Machinery.

A Model and Definitions

We consider a network switch equipped with an on-chip buffer size of B units shared by N ports. We mainly follow the widely used model in the literature [8, 24, 30]. Time is discrete, and we refer to each step as timeslot. Packets (each of size unit 1) arrive in an online manner as time progresses. Each timeslot is divided into two phases, arrival phase and departure phase. During each arrival phase, at most N number of packets (in aggregate) arrive destined to N ports. During each departure phase, every queue drains out one packet unless the queue is empty. A buffer sharing algorithm manages the shared buffer allocation across the N ports. We next define preemptive (push-out) and non-preemptive (drop-tail) buffer sharing.

Definition 2 (Preemptive buffer sharing)**.** *During every arrival phase, the buffer sharing algorithm is allowed to preempt i.e., drop any number of existing packets in the buffer.*

Definition 3 (Non-preemptive buffer sharing)**.** *During every arrival phase, the buffer sharing algorithm is only allowed to accept or drop the incoming packet. Every accepted packet must eventually be drained out from the corresponding queue.*

We denote by $\sigma(t) = (\sigma_i(t), \sigma_i(t), ..., \sigma_N(t))$, an N-tuple, where $\sigma_i(t)$ denotes the number of packets arriving at time t to queue i. We study the performance of a buffer sharing algorithm in terms of throughput i.e., our objective is to maximize the total number of packets transmitted over the entire arrival sequence. We compare the performance of our online algorithms against an offline optimal algorithm, which has access to the entire arrival sequence at $t = 0$ and has infinite computational capacity.

Definition 4 (Competitive ratio)**.** *Let ALG be an online algorithm and OPT be an offline optimal algorithm for the buffer sharing problem. Let $ALG(\sigma)$ and $OPT(\sigma)$ be the total number of packets transmitted by ALG and OPT for the arrival sequence σ. We say ALG is c-competitive if it satisfies the following condition for any arrival sequence σ.*

$$OPT(\sigma) \leq c \cdot ALG(\sigma) \tag{3}$$

B FollowLQD: A Deterministic Algorithm

In this section we propose a new online deterministic algorithm FollowLQD in the non-preemptive case which is a non-predictive building block of CREDENCE. Intuitively, FollowLQD simply *follows* the Longest Queue Drop (LQD) queues in the preemptive model. In particular, FollowLQD maintains a threshold $T_i(t)$ for each queue at time t. The thresholds are updated for every packet arrival and departure according to LQD in the preemptive model. We present the pseudocode for FollowLQD in Algorithm 2. While FollowLQD tries to *follow* LQD queue lengths by accepting packets as long as the queue lengths are smaller than the thresholds, it may happen that FollowLQD queues are larger than their thresholds. This is since FollowLQD cannot preempt (remove) existing packets in the buffer whereas LQD can preempt and correspondingly the thresholds may drop below the queue lengths. FollowLQD simply drops an incoming packet if it finds that the corresponding queue exceeds its threshold.

Algorithm 2: FollowLQD

```
Input : σ(t)
procedure ARRIVAL(σ(t)):
    for each packet p ∈ σ(t) do
        Let i be the destination queue for the packet p
        UPDATETHRESHOLD(i, arrival)
        if q_i(t) < T_i(t) then
            if Q(t) < B then
                q_i(t) ← q_i(t) + 1                    ▷ accept
        else
                                                       ▷ Drop
procedure DEPARTURE(i):
    if q_i(t) > 0 then
        q_i(t) ← q_i(t) − 1                ▷ Drain one packet
    UPDATETHRESHOLD(i, departure)
function UPDATETHRESHOLD(i, event):
    if event = arrival then
        if Γ(t) = B then                   ▷ Sum of thresholds
            Let T_j(t) be the largest threshold
            T_j(t) ← T_j(t) − 1                        ▷ Decrease
            T_i(t) ← T_i(t) + 1                        ▷ Increase
        else
            T_i(t) ← T_i(t) + 1                        ▷ Increase
            Γ(t) ← Γ(t) + 1
    if event = departure then
        if T_i(t) > 0 then
            T_i(t) ← T_i(t) − 1                        ▷ Decrease
            Γ(t) ← Γ(t) − 1
```

Although LQD is known to be 1.707-competitive, we show that FollowLQD is still at least $\frac{N+1}{2}$-competitive. We present our lower bound based on a simple arrival sequence.

Observation 1. *FollowLQD is at least $\frac{N+1}{2}$-competitive.*

Proof. We construct an arrival sequence such that for every two packets transmitted by FollowLQD, the offline optimal algorithm OPT transmits $N+1$ packets. Consider that all the queues are empty at time $t = 0$. We then burst packets to a single queue say i until its queue length reaches B. Note that this is possible since the threshold for queue i that follows the corresponding LQD queue also grows up to B. At the end of the departure phase, FollowLQD transmits one packet and the queue length becomes $B-1$. At this point, we send N packets, one packet to each of the N queues. The thresholds are updated based on LQD, which has the following actions: *(i)* preempt $N-1$ packets from queue i and *(ii)* accept all N packets to N queues. Correspondingly, the threshold for

queue i of FollowLQD drops to $B-N+1$ but it still has $B-1$ packets in queue i. As a result, it can only accept one packet out of the N incoming packets. At the end of the departure phase during this timeslot, FollowLQD has $B-1$ packets in queue i and has transmitted 1 packet in total. In the next timeslot, we send N packets to the queue i so that LQD's queue i now gets back to size B again. As the threshold is larger than the queue length $(B-1)$, FollowLQD accepts 1 packet. At the end of the departure phase, FollowLQD transmits 1 packet from the queue i. Overall, FollowLQD transmitted 2 packets but OPT transmitted $N+1$ packets. We then repeat the sequence such that for every $N+1$ packets transmitted by OPT, FollowLQD transmits 2 packets. The competitive ratio is then at least $\frac{N+1}{2}$. □

C Buffer Sharing Algorithms with Predictions

In this section, we introduce our model for buffer sharing where there exists an oracle that predicts packet drop (or accept) for each packet in the arrival sequence σ, according to the prediction model introduced in §2.3.1. We denote the drop sequence of LQD for the arrival sequence σ by $\phi(\sigma)$, and the predicted drop sequence by $\phi'(\sigma)$. We classify prediction for each packet in to four types: true positive, false positive, true negative and false negative (see Figure 5. We denote the sequence of true positive predictions by $\phi'_{TP}(\sigma)$, false positive predictions by $\phi'_{FP}(\sigma)$, true negative predictions by $\phi'_{TN}(\sigma)$ and false negative predictions by $\phi'_{TP}(\sigma)$. We drop σ in our notations when the context is clear.

Hereafter, we mainly compare our online non-preemptive algorithm with predictions against online LQD (preemptive). We define the error made by the oracle by the following error function.

Definition 1 (Error function)**.** *Let LQD(σ) and FollowLQD(σ) denote the total number of packets transmitted by the online push-out algorithm LQD and the online drop-tail algorithm FollowLQD over the arrival sequence σ. Let ϕ denote the sequence indicating drop by LQD for each packet in the arrival sequence σ. Let ϕ' denote the sequence of drops predicted by the machine-learned oracle. Let ϕ'_{TP}, ϕ'_{FP}, ϕ'_{TN}, and ϕ'_{FN} denote the sequence of true positive, false positive, true negative and false negative predictions for the arrival sequence σ. We define the error function $\eta(\phi,\phi')$ as follows:*

$$\eta(\phi,\phi') = \frac{LQD(\sigma)}{FollowLQD\left(\sigma-\phi'_{TP}-\phi'_{FP}\right)} \tag{1}$$

We now analyze CREDENCE that relies on drop predictions ϕ' and takes decisions in pursuit of *following* LQD more accurately. Algorithm 1 presents the pseudocode for our algorithm.

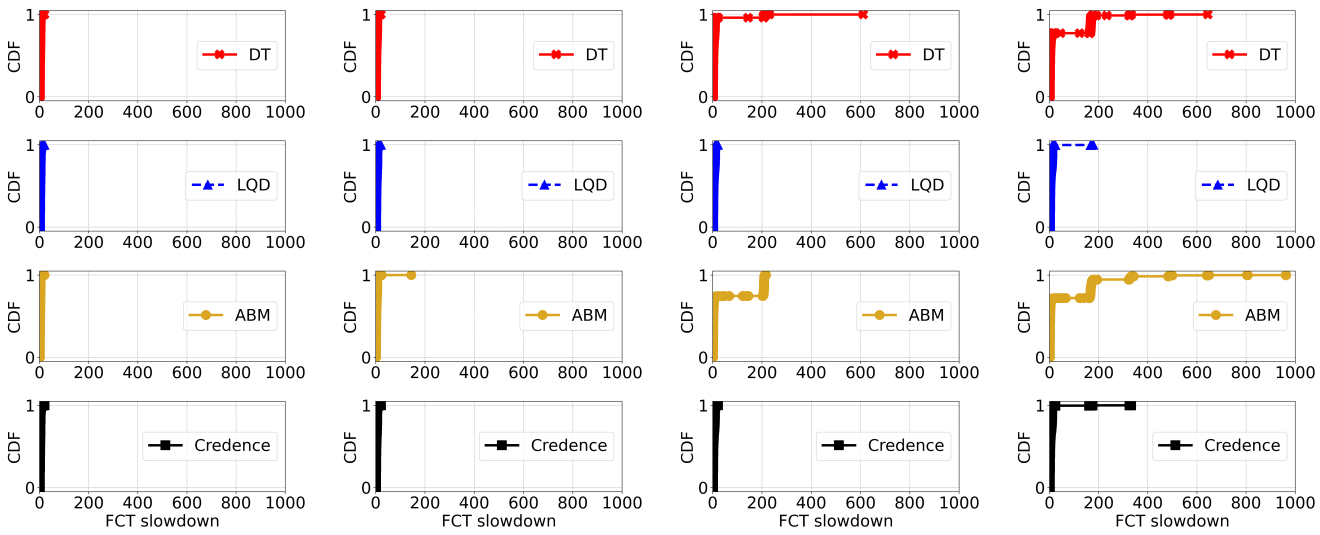
In essence, perfect predictions allows us to perfectly follow LQD queues, essentially transmitting as many packets as LQD. However, we are also concerned about the performance of the algorithm when the oracle makes mispredictions. In the following, we study the competitive ratio of CREDENCE as the error grows. We obtain the competitive ratio as a function of the error η: we show that CREDENCE is 1-competitive against LQD with perfect predictions but at most N-competitive when the error is arbitrarily large.

Theorem 1. *The competitive ratio of* CREDENCE *grows linearly from* 1.707 *to N based on the prediction error* $\eta(\phi,\phi')$*, where N is the number of ports,* ϕ *is the drop sequence of LQD and* ϕ' *is the predicted sequence of drops i.e., the competitive ratio is at most* $\min(1.707\,\eta(\phi,\phi'),N)$.

Lemma 1. *The total number of packets transmitted by* CREDENCE *for an arrival sequence* σ*, a drop sequence* ϕ *by LQD and the predicted drop sequence* ϕ' *is given by*

$$\text{CREDENCE}(\sigma) \geq \frac{LQD(\sigma)}{\underbrace{\eta(\phi,\phi')}_{error}} \tag{2}$$

Proof. For simplicity, we refer to CREDENCE as ALG in the following. We prove our claim by analyzing the drops of ALG and relating the transmitted packets by $ALG(\sigma)$ to $LQD(\sigma')$. Every drop of *ALG* arises from three types of situations. First, ALG can drop a packet due to the thresholds. Note that the thresholds used by *ALG* correspond to the queue lengths of preemptive LQD over the same arrival sequence σ. As a result, both ALG and FollowLQD algorithm have the same thresholds at any time instance. Second, ALG drops a packet if the prediction is either true positive or false positive if and only if the queue length satisfies the corresponding thresholds. This type of drops are at most the total number of true positive and false positive predictions. Third, ALG drops a packet when the buffer is full which is the same condition for FollowLQD. In essence, ALG drops at most all the positive predictions and drops at most the number of packets dropped by FollowLQD serving the arrival sequence $\sigma-\phi'_{TP}-\phi'_{FP}$ i.e., the arrival sequence in which all the packets predicted as positive are removed from σ. In order to prove our main claim, it remains to argue that the extra packets accepted by ALG due to the safeguard condition do not result in additional drops compared to FollowLQD with the arrival sequence $\sigma-\phi'_{TP}-\phi'_{FP}$. For every packet that fails to satisfy the threshold but gets accepted due to the safeguard condition by ALG, could cause at most one extra drop due to the thresholds before the buffer full again compared to FollowLQD. This is since, if FollowLQD accepts a packet, then its queue length is certainly less that the corresponding threshold (that is same for ALG). However, the queue length of ALG may have some extra packets that are accepted due to the safeguard condition. As a result, each such extra packet (dropped by FollowLQD) contributes to at most one drop compared compared to FollowLQD and the transmitted packets remains equivalent. Further, by the time the buffer is full in ALG, all the extra packets accepted due



(a) Burst size = 12.5% (b) Burst size = 25% (c) Burst size = 50% (d) Burst size = 75%

Figure 11: CDF of flow completion times (slowdown) for CREDENCE, DT, ABM and LQD across various burst sizes of incast workload and websearch workload at 40% load, with DCTCP as the transport protocol. Burst size is expressed as a percentage of the buffer size.

to the safeguard condition would have been drained out of the buffer. This is due to the fact that any such extra packet is at a queue length of at most $\frac{B}{N}$ (the safeguard condition) that drains out before the buffer fills up i.e., it takes at least $\frac{B}{N}$ timeslots to fill the buffer (only N packets can arrive in each timeslot). As a result, ALG transmits at least the total number of packets transmitted by FollowLQD over the arrival sequence $\sigma - \phi'_{TP} - \phi'_{FP}$ i.e.,

$$ALG(\sigma) \geq FollowLQD(\sigma - \phi'_{TP} - \phi'_{FP})$$

Using Definition 1, we express FollowLQD in terms of LQD and the error function $\eta(\phi, \phi')$, and obtain Equation 2. □

Lemma 2. CREDENCE *transmits at least* $\frac{1}{N}$ *times the number of packets transmitted by an offline optimal algorithm OPT i.e.,* $\text{CREDENCE}(\sigma) \geq \frac{1}{N} \cdot OPT(\sigma)$.

Proof. Irrespective of the predictions, CREDENCE always accepts an incoming packet if the longest queue is less than or equal to $\frac{B}{N}$. When CREDENCE drops a packet, there is at least one queue that has at least $\frac{B}{N}$ number of packets. Hence, every packet in OPT can be matched to at least $\frac{B}{N}$ number of packets. Consequently, the competitive ratio is at most N. □

We are now ready to prove our main claim (Theorem 1) using the above results.

Proof of Theorem 1. From Definition 4, in order to prove the competitive ratio of our CREDENCE, we are mainly concerned with the upper bound of $\frac{OPT(\sigma)}{\text{CREDENCE}(\sigma)}$ for any arrival sequence σ. Since $\frac{OPT(\sigma)}{LQD(\sigma)} \leq 1.707$ is known from literature [8, 24], we use this result to compare CREDENCE and LQD in order to argue about the competitive ratio i.e, $\frac{OPT(\sigma)}{\text{CREDENCE}(\sigma)} \leq 1.707 \cdot \frac{LQD(\sigma)}{\text{CREDENCE}(\sigma)}$ for any request sequence σ. From Lemma 1, we have the following:

$$\frac{LQD(\sigma)}{\text{CREDENCE}(\sigma)} \leq \eta(\phi, \phi')$$

From Lemma 2, irrespective of the predicted sequence, we have that $\frac{OPT(\sigma)}{\text{CREDENCE}(\sigma)} \leq N$. Finally, since $\frac{OPT(\sigma)}{\text{CREDENCE}(\sigma)} \leq 1.707 \cdot \frac{LQD(\sigma)}{\text{CREDENCE}(\sigma)}$, the competitive ratio of CREDENCE is given by:

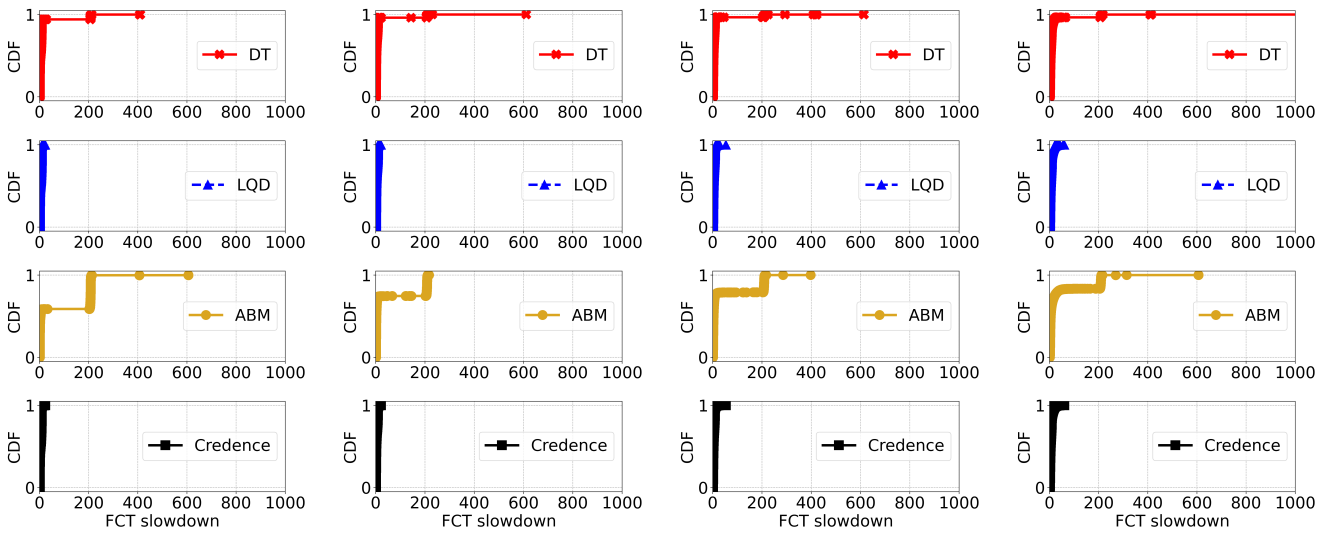
$$\frac{OPT(\sigma)}{\text{CREDENCE}(\sigma)} \leq \min\left(1.707\ \eta(\phi, \phi'),\ N\right)$$

The proof follows by Definition 4. □

Theorem 2. *Let* ϕ' *denote the sequence of drops predicted by the machine-learned oracle. Let* ϕ'_{TP}, ϕ'_{FP}, ϕ'_{TN}, *and* ϕ'_{FN} *denote the sequence of true positive, false positive, true negative and false negative predictions for the arrival sequence* σ. *The error function* $\eta(\phi, \phi')$ *(Definition 1) is upper upper bounded as follows:*

$$\eta(\phi, \phi') \leq \frac{\phi'_{TN} + \phi'_{FP}}{\phi'_{TN} - \min\left((N-1) \cdot \phi'_{FN}, \phi'_{TN}\right)}$$

Proof. Our proof is based on two arguments: *(i)* $LQD(\sigma) = \phi'_{TN} + \phi'_{FP}$ and *(ii)* $FollowLQD(\sigma - \phi'_{TP} - \phi'_{FP}) \geq \phi'_{TN} - \min\left((N-1) \cdot \phi'_{FN}, \phi'_{TN}\right)$.



(a) Load = 20% (b) Load = 40% (c) Load = 60% (d) Load = 80%

Figure 12: CDF of flow completion times (slowdown) for CREDENCE, DT, ABM and LQD across various loads of websearch workload and incast workload at a burst size 50% of the buffer size, with DCTCP as the transport protocol.

First, $LQD(\sigma)$ is the total number of transmitted packets by LQD. Recall that the ground-truth (transmitted by LQD) for a prediction is an accept if and only if the prediction is either true negative or a false positive. Hence, the total number of packets transmitted by LQD i.e., $LQD(\sigma)$ is the sum of true negative predictions and the false positive predictions.

Second, $FollowLQD(\sigma - \phi'_{TP} - \phi'_{FP})$ transmits at least $\phi'_{TN} + \phi'_{FN} - Y$, where Y is the total number of drops caused by false negative predictions. Note that $\sigma = \phi'_{TN} + \phi'_{FN} + \phi'_{FP} + \phi'_{TP}$. The proof follows by showing that each false negative results in at most N extra drops due to the buffer limit. Further, these extra drops *must* be true negative predictions since we have already removed positive predictions from our arrival sequences (i.e., we assume at most all the positive predictions have be dropped). Additionally, since the extra drops are true negative predictions, it implies that LQD transmits those packets but although our prediction is true, we incur additional drop due to the buffer limit. For each false negative, there can be at most one drop in a single timeslot for up to $N-1$ distinct timeslots such drops that LQD accepts and transmits those packets but FollowLQD drops them. Beyond $N-1$ drops, there can only be at most 1 other drop upon which the existence of an additional packet (false negative) in FollowLQD's buffer would be nullified. This is since, during the initial $N-1$ drops, FollowLQD could not accept the incoming packet but after the transmission phase, the queues having false negative predictions decrement their size by 1. This leaves at least $N-1$ packets free space in FollowLQD after $N-1$ drops and LQD also has the same remaining space after those extra $N-1$ accepted by LQD are also transmitted. At this time, both LQD and FollowLQD have the same remaining space and they also transmit the same number of packets in each timeslot. One additional drop by FollowLQD corresponding to a false negative is still possible due the thresholds i.e., if there exists a packet arrival to the queue having false negative, the incoming packet is dropped since the existence of false negatives implies that the queue length is large than the threshold. As a result, there are at most N drops by FollowLQD for each false negative prediction.

The proof follows by the above two arguments. □

For completeness, although well-known in the literature, we define accuracy, precision, recall and f1 score below (used in Figure 15 in §4).

$$Accuracy = \frac{\phi'_{TP} + \phi'_{TN}}{\phi'_{TP} + \phi'_{TN} + \phi'_{FP} + \phi'_{FN}}$$

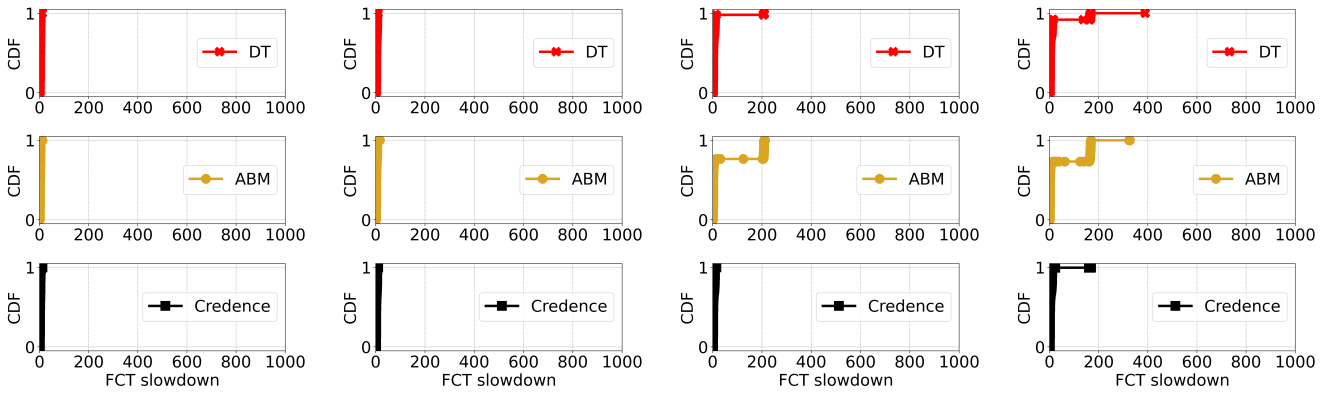
$$Precision = \frac{\phi'_{TP}}{\phi'_{TP} + \phi'_{FP}}$$

$$Recall = \frac{\phi'_{TP}}{\phi'_{TP} + \phi'_{FN}}$$

$$F1\ score = \frac{2 \cdot \phi'_{TP}}{2 \cdot \phi'_{TP} + \phi'_{FP} + \phi'_{FN}}$$

D Additional Results

In this section, we present additional results from our evaluations. Figures 11, 12, 13 present the CDF of flow completion times for each experiment in our evaluations (§4), showing the complete performance profile of each algorithm.



(a) Burst size = 12.5% (b) Burst size = 25% (c) Burst size = 50% (d) Burst size = 75%

Figure 13: CDF of flow completion times (slowdown) for CREDENCE, DT, ABM and LQD across various burst sizes of incast workload and websearch workload at 40% load, with PowerTCP as the transport protocol. Burst size is expressed as a percentage of the buffer size.

Figure 14 presents our numerical results based on a custom simulator in discrete time. Note that Figure 14 shows the throughput *ratio* of an algorithm vs LQD. We perform this experiment using custom simulator in order to fully control the prediction error (artificially).

We generate large bursts of the size of the total buffer, where each such burst arrives according to a poisson process (which is fixed in subsequent runs). We then collect a trace of per-packet drop (or accept) trace using LQD as the buffer sharing algorithm. This trace serves as the ground-truth as well as the case for perfect predictions for CREDENCE. We then run CREDENCE over the same packet arrival sequence from above, and use the drop trace of LQD as predictions. With full access to this trace i.e., perfect predictions case, CREDENCE performs exactly as LQD as expected. However, in order to study the performance of CREDENCE with increasing error, in a controlled manner, we flip each packet drop (or accept) from our LQD's drop trace i.e., each flip becomes a false prediction. We control the error via the flipping probability i.e., the false prediction rate. We observe from Figure 14 that CREDENCE degrade in throughput as the probability of false predictions increases i.e., as the prediction error increases. However, even at as high as 0.7 probability of false predictions, CREDENCE still out-performs DT.

In Figure 15, we present our results obtained from a parameter sweep across the number of trees used for random forest model vs prediction scores.

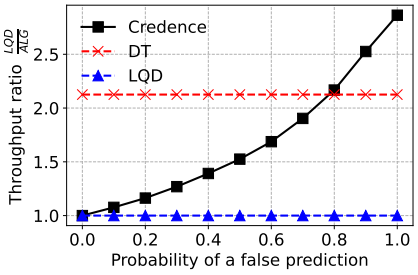


Figure 14: As the probability of false predictions increases, CREDENCE's throughput compared to LQD (push-out) i.e., the ratio $\frac{LQD}{ALG}$ increases from 1 to 2.9 (lower values are better). CREDENCE performs significantly better than DT even when the probability of false predictions is as high as 0.7.

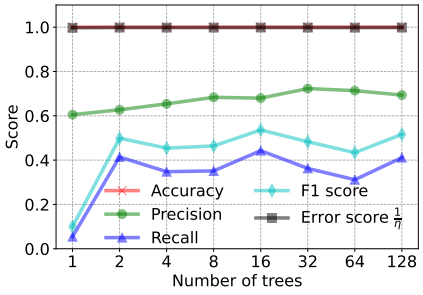


Figure 15: The quality of our predictions does not improve significantly beyond 4 trees in our random forest classifier.