Per-CCA Queueing

Yara Mulla [1,2], Isaac Keslassy[1]
[1] *Technion* [2] *Nvidia*

***Abstract*—Due to their increasing aggressiveness, recent congestion control algorithms (CCAs) can starve vanilla TCP flows in their shared router queues. Unfortunately, existing router-based solutions cannot prevent this starvation.**

In this paper, we introduce a *per-CCA queue isolation* where incoming flows first undergo CCA classification, and then are mapped to isolated queues based on their classified CCA. We provide a fundamental analysis for this per-CCA isolation, and present two models for its performance. Then, in evaluations, we show how this per-CCA isolation clearly outperforms buffer sharing, and how our advanced model can accurately represent its performance. We further show how we can increase fairness by optimizing the queue service rates.

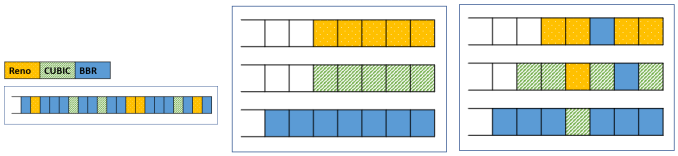
I. Introduction

CCA starvation. Public cloud providers face increasing unfairness between the congestion control algorithms (CCAs) in their network. The main reason is that cloud users can relatively easily change the CCA that runs in their virtual machines. Therefore, users increasingly implement aggressive CCAs that can be strongly unfair to competing vanilla TCP flows. For example, even a single BBR flow can quickly take over a large portion of a shared queue and nearly starve competing Reno and CUBIC flows [1]–[5]. In addition, users can leverage cloud-based TCP-split proxies to make these CCAs even more aggressive [6]–[8]. It is an arms race. The resulting CCA starvation is becoming a significant issue to cloud providers, because users pay for good network performance and can readily migrate to cloud competitors when their flows get starved [9].

Avoiding CCA starvation. To protect vanilla CCAs from aggressive ones, a naive approach for cloud providers would be to apply one of the many fair-queueing [10], [11] and/or admission-control related algorithms [12], [13]. However, as explained by Cebinae [14], fair-queueing algorithms cannot meet the hardware requirements of routers in public clouds with large numbers of flows, and admission-control algorithms typically drop overflow packets, thus arbitrarily hurting non-loss-based CCAs.

Instead, Cebinae [14] attempts to achieve reasonable fairness by slightly reducing the rates of a few heavy hitters. However, in practice, since Cebinae does not know the feedback mechanisms of the heavy-hitter CCAs, it always knocks them down with a triple combination of losses, latency, and ECN bits, potentially yielding large oscillations and starvation.

CCA-family queueing. Several recent papers introduce a promising approach that dedicates distinct queues to different CCA families. P4air [15] was apparently the first to map all flows that belong to the same CCA family into a distinct queue: e.g., a queue for loss-based CCAs, and another one for delay-based CCAs. This helps increase fairness, as P4air does not put together CCAs with different mechanisms. However, P4air can still map two loss-based CCAs with significantly different aggressiveness into the same loss-based queue, and therefore can still experience CCA-based starvation within a CCA family. The P4air approach seems to gain traction, as recently Confucius [16] and P4CCI [17] also rely on such a queue-based isolation for different CCA classes. For example, Confucius groups together flows based on their expected queue occupancy. However, it can group together CCAs with different feedback mechanisms, and therefore may still exhibit significant unfairness within a CCA class.



(a) Shared buffer (b) Per-CCA queues with ideal classification (c) Per-CCA queues with real-life classification

Fig. 1. Buffer shared by several CCAs: (a) Typical FIFO shared buffer. (b) Following an ideal CCA classification, each CCA is mapped into a different queue. (c) In real life, some flows may be misclassified.

Online CCA classification. Independently of the above methods to avoid CCA starvation, the popularization of strong machine learning (ML) tools has enabled the recent emergence of an online CCA classification field. While past CCA classification algorithms were mostly offline and often needed to monitor the ACKs as well [18]–[22], recent algorithms such as DeePCCI [23] and Dragonfly [24] enable online CCA classification without monitoring the reverse path.

Per-CCA queueing. P4air, Confucius and P4CCI create a queue isolation between CCA classes, but may exhibit unfairness *within* classes. In this paper, we naturally ask whether router vendors could provide full *per-CCA queueing*.

Fig. 1 illustrates our idea. First, Fig. 1(a) shows how today, a shared FIFO queue combines all flows. Instead, Fig. 1(b) illustrates *per-CCA queue isolation*: after going through a *CCA classifier*, packets of each flow are placed in an isolated queue that corresponds to their flow CCA. Assume at first for simplicity that buffers are split equally, and that each queue is served proportionally to its number of flows. Then flows with vanilla CCAs are guaranteed service and can avoid starvation from aggressive CCAs, which are restricted to their

own queues. For instance, by staying in their own (blue) swimming lane, BBR flows are prevented from interfering with Reno and CUBIC flows.

The benefits of this per-CCA queueing idea are clear. (1) The isolation would enable cloud users to use the CCA that best fits their application, without hurting other users. (2) It would enable routers to apply the best feedback mechanism (ECN, INT, etc.) to each CCA. (3) It would also enable CCA developers to design CCAs that better fit applications, and that only need to be fair to themselves, without a mandate to be *TCP-friendly* [5]. In short, it could develop into a new queueing discipline and unleash novel CCA developments.

On the other hand, the per-CCA isolation idea needs to solve countless implementation problems to be practical, e.g., (1) the CCA classifier should provide a classification after a few RTTs, even when starting to examine a flow in the middle; (2) it should automatically learn to classify CCAs, without a need to define CCA protocols manually; (3) it should do so while only locally examining the packets at the router buffer, without their ACKs and without communicating with the end-points; (4) it should be accurate enough to get a good queue isolation; (5) it should classify dozens of CCAs; (6) for scalability, the queueing algorithm should be able to separate elephants to classify urgently, and mice that can stay in some unclassified queue; (7) it should deal with re-classifications; (8) it should deal with unknown CCAs on which it has not trained; (9) it may need to further classify the same CCA into distinct effective CCAs based on its RTT or on its tuned parameters; (10) it should be able to approximate the number of flows per queue for a fair service rate; (11) it should remember classifier decisions; etc.

Some of these problems are already addressed in the literature. For instance, the DeePCCI and Dragonfly classifiers address the first 3 problems [23], [24]. The 10th problem of efficiently estimating the number of flows is largely solved [25]. The 11th problem can be solved with a simple hash table.

Building on the fast-growing machine-learning literature, classifiers get quickly better. We can soon expect them to reach a strong accuracy. Motivated by these recent advances, the goal of this paper is to consider the 4th problem: *how accurate should a classifier be to provide a good queue isolation?* Fig. 1(c) illustrates a significant problem of the per-CCA isolation idea: misclassifications. Some BBR flows may be classified as Reno or CUBIC, and will be mapped to the wrong queues. Like sharks in a pool of fish, even a few BBR flows in the wrong queues may cause unfairness again. Our goal is to fundamentally study how accurate the CCA classifier needs to be in order to counteract the imbalance in aggressiveness between the different CCAs. For instance, if a current classifier reaches an accuracy of 0.8, does it help, or should it be at least 0.999 to be helpful? We want to provide fundamental guidelines for cloud providers on the needed classifier accuracy, not to design another classifier.

Contributions. This paper makes several contributions. (1) It introduces *per-CCA queueing*, and discusses the many hurdles on the way to practicality. (2) It focuses on the fundamental classifier accuracy needed to obtain good isolation between CCAs. To do so, it discovers an intrinsic property of current and future classifiers: the misclassification probability for a CCA i is expected to fall as the inverse square root of its aggressiveness. For example, a CCA that is $4\times$ more aggressive than Reno is expected to have half of its misclassification rate. (3) Based on this property, the paper introduces two models for the performance of per-CCA queueing, and shows that the advanced model provides a tight fit. (4) The evaluations demonstrate how per-CCA queueing clearly outperforms buffer sharing. For instance, with equal numbers of Reno, CUBIC and BBR flows, buffer sharing only provides 11.9% of the fair bandwidth share to the most vulnerable CCA (Reno). Per-CCA queueing provides 50% for a classifier accuracy of $F_1 = 0.83$, and 80% for $F_1 = 0.95$. (5) This outperformance also holds with real classifier accuracy values from DeePCCI and Dragonfly [23], [24]. In fact, their current accuracy is already sufficient to strongly increase the bandwidth share provided to vulnerable CCAs. (6) The unfairness between CCAs can be further decreased by optimizing the per-CCA queue service rates. (7) Finally, evaluations show that our models extend to *per-CCA-class* queueing, as in P4air [15], Confucius [16] and P4CCI [17].

II. Performance model and analysis

A. Classifier assumptions

We study the fundamental performance limits of a classifier-based per-CCA queue isolation. Our goal is to determine the impact of this classifier's accuracy on the system fairness among the CCAs. We make two simplifying assumptions to focus on the intrinsic impact of the classifier accuracy, while setting aside for now the system's implementation details.

Classifier training in a shared buffer. The CCA classifier trains once offline with $K \geq 2$ CCAs in a dumbbell topology with a shared-buffer bottleneck router. Each CCA is used by the same number of flows (e.g., 50 flows per CCA).

No classifier delay. We assume that the classifier can immediately classify a flow when it enters the system, and do not study the impact of delaying the classification. All future packets of this flow are classified in the same way.

B. Classification error

We want to provide a general model for the classification error of current and future CCA classifiers.

Classification probability. Given that a flow has CCA $i \in \{1, \ldots, K\}$, let q_{ij} denote the conditional probability that it is classified as having CCA $j \in \{1, \ldots, K\}$. q_{ij} satisfies

$$\sum_{j=1}^{K} q_{ij} = 1, \quad \forall i \in \{1, \ldots, K\}. \tag{1}$$

In a perfect classifier, $q_{ii} = 1$ and $q_{ij} = 0$ for any $j \neq i$.

Classification error bound. We leverage the theory of multiclass learnability [26]–[29]. We assume that during training,

the CCA classifier has encountered n_i sample packets for each CCA i. We can then use the Natarajan dimension, which characterizes the complexity of learning a set of functions, and generalizes the well-known Vapnik-Chervonenkis dimension from two-class to multiclass functions [26]. It establishes the following upper bound on the misclassification error, which can also be seen as an application of Hoeffding's inequality [28]:

$$\exists \lambda > 0, \forall i, \forall j \neq i, \quad q_{ij} \leq \frac{\lambda}{\sqrt{n_i}}. \tag{2}$$

In other words, *the more training samples n_i we get for CCA i, the better we can classify it*. This bound on the misclassification error provides two interesting contributions. First, it falls like the square root of the number of samples. Second, it is independent of j, which can be counter-intuitive: e.g., the bound on the probability of misclassifying BBR as Reno only depends on the number of BBR samples we have seen, not on the number of Reno samples.

Classification error model. In the remainder of the paper, we want a simple exact model for the multiclass classification error. To do so, we will assume that this error falls with the number of samples in the same way as its upper bound, i.e., that there exists some β with $0 < \beta < \lambda$ that satisfies:

$$\forall i, \forall j \neq i, q_{ij} = \frac{\beta}{\sqrt{n_i}}. \tag{3}$$

Combining Eq. (1) and Eq. (3) yields the following formula for the correct classification probability of CCA i:

$$q_{ii} = 1 - \sum_{j \neq i} q_{ij} = 1 - \frac{(K-1) \cdot \beta}{\sqrt{n_i}}, \quad \forall i. \tag{4}$$

Training model. We now need to model how many packet samples the CCA classifier sees during its training. We assume that during training, we send the same number of flows for each CCA. However, a more aggressive flow will send many more packets per flow, and therefore will obtain more samples. Thus, the number of samples is related to the aggressiveness of the CCA. Let n denote the total number of packets seen during the training:

$$n = \sum_{i=1}^{K} n_i. \tag{5}$$

Then we define the aggressiveness factor g_i of CCA i as the ratio of the bandwidth of a flow with CCA i to its fair share.

$$g_i = \frac{n_i}{n/K}. \tag{6}$$

Combining with Eq. (3), we obtain the following proposition:

Proposition 1 (misclassification probability). *The probability of misclassifying a flow with CCA i as CCA $j \neq i$ is:*

$$q_{ij} = \frac{\beta\sqrt{K/n}}{\sqrt{g_i}} \quad \forall i, \forall j \neq i. \tag{7}$$

Proposition 1 states that *the misclassification error for CCA i is inversely related to the square root of its aggressiveness*. This formula can be seen as a key result that characterizes multi-class CCA classifiers, and is core to our understanding of per-CCA queueing with imprecise classification.

We saw in Fig. 1(c) that flows of aggressive CCAs could be misclassified and sent to the queues of less aggressive ones. An interesting observation about Eq. (7) is that we would expect a more aggressive CCA like BBR to experience less misclassification errors than a vanilla CCA like Reno. The more aggressive a CCA is, the less misclassified it can be, and therefore the less likely it is to be sent to the wrong queue. Therefore, this works in favor of per-CCA queueing: Sharks are less likely to be wrongly sent to the pool of fish. Of course, on the other hand, vanilla CCAs are more likely to be misclassified, and therefore fish are more likely to be wrongly sent to the pool of sharks.

C. Flow distribution with any CCA arrivals

Flow distribution A. Let a_{ij} denote the probability that an arbitrary flow arriving at the buffer uses CCA i and is classified as CCA j. It satisfies $\sum_{i,j} a_{ij} = 1$. Let A denote the associated matrix $[a_{ij}]_{ij}$ of all such probabilities.

Goal: model A given F_1. We want to study the impact of the accuracy of the classifier on the fairness of our system. To do so, we classically quantify the performance of the multiclass CCA classifier using its F_1 score, a measure of its accuracy that is defined as the harmonic mean of its precision and its recall. F_1 is in $[0, 1]$, and equals 1 for a perfect classifier. The following theorem states that A is a function of F_1.

Theorem 1 (flow distribution). *(i) The flow distribution matrix A can be expressed as a function of F_1.*
(ii) The percentage of flows with CCA i and misclassified as CCA $j \neq i$ is a strictly-decreasing linear function of F_1, and it equals 0 when $F_1 = 1$.

Proof: (i) Case 1: equal number of flows. We start with the homogeneous case where each CCA has an equal number of flows. In this case, we denote a_{ij} by a'_{ij} and A by A'.

Since $1/K^{\text{th}}$ of all flows belong to each CCA, for all i, $\sum_{j=1}^{K} a'_{ij} = \frac{1}{K}$. By Bayes' rule, the probability that an arbitrary flow uses CCA i and is classified as CCA j is the product of the probability $1/K$ that it uses CCA i, by the conditional probability q_{ij} that it is classified as CCA j given that it uses CCA i. Therefore,

$$a'_{ij} = \frac{1}{K} \cdot q_{ij} \quad \forall i, j. \tag{8}$$

Let $\gamma = \frac{\beta}{\sqrt{Kn}}$. Combining with the misclassification model of Eq. (7), we obtain $a'_{ij} = \frac{1}{K} \cdot \frac{\beta\sqrt{K/n}}{\sqrt{g_i}} = \frac{\gamma}{\sqrt{g_i}}$ for all i and all $j \neq i$. Likewise, for all i, since $\sum_{j=1}^{K} a'_{ij} = \frac{1}{K}$, we get $a'_{ii} = \frac{1}{K} - (K-1)\frac{\gamma}{\sqrt{g_i}}$. Combining both equations,

$$A' = \begin{pmatrix} \frac{1}{K} - \frac{(K-1)\gamma}{\sqrt{g_1}} & \frac{\gamma}{\sqrt{g_1}} & .. & \frac{\gamma}{\sqrt{g_1}} \\ \frac{\gamma}{\sqrt{g_2}} & \frac{1}{K} - \frac{(K-1)\gamma}{\sqrt{g_2}} & .. & \frac{\gamma}{\sqrt{g_2}} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\gamma}{\sqrt{g_K}} & \frac{\gamma}{\sqrt{g_K}} & .. & \frac{1}{K} - \frac{(K-1)\gamma}{\sqrt{g_K}} \end{pmatrix}. \tag{9}$$

F_1 is the sum of the diagonal values of A, thus

$$F_1 = 1 - (K-1)\gamma \sum_{i=1}^{K} \frac{1}{\sqrt{g_i}}. \quad (10)$$

We can verify that if the number n of training samples goes to infinity, then $\gamma = \frac{\beta}{K\sqrt{n}}$ goes to zero, and F_1 converges to one. Also, from Eq. (10), we can get γ as a function of F_1:

$$\gamma = \frac{1 - F_1}{(K-1)\sum_{i=1}^{K} \frac{1}{\sqrt{g_i}}}. \quad (11)$$

Combining this formula with Eq. (9) yields:

$$A' = \frac{1 - F_1}{(K-1) \cdot \sum_{i=1}^{K} \frac{1}{\sqrt{g_i}}} \cdot \begin{pmatrix} \alpha_{11}(F_1) & \frac{1}{\sqrt{g_1}} & .. & \frac{1}{\sqrt{g_1}} \\ \frac{1}{\sqrt{g_2}} & \alpha_{22}(F_1) & .. & \frac{1}{\sqrt{g_2}} \\ \vdots & \vdots & : & \vdots \\ \frac{1}{\sqrt{g_K}} & \frac{1}{\sqrt{g_K}} & .. & \alpha_{KK}(F_1) \end{pmatrix}, \quad (12)$$

where $\alpha_{ii}(F_1) = \frac{(K-1)\sum_{i=1}^{K} \frac{1}{\sqrt{g_i}}}{K(1-F_1)} - \frac{K-1}{\sqrt{g_i}}$.

Case 2: general case. We now extend the model to the general case. Let N_i denote the number of incoming flows that belong to CCA i, and $N = \sum_i N_i$ denote the total number of flows. By Bayes' rule, a_{ij} is the product of the probability $\frac{N_i}{N}$ that an arbitrary flow belongs to CCA i, by the conditional misclassification probability q_{ij}. Combining with Eq. (8),

$$a_{ij} = \frac{N_i}{N} \cdot K \cdot a'_{ij}. \quad (13)$$

Namely, the flow-distribution probability matrix A is the same as its homogeneous equivalent A' but with rescaled rows.

(ii) By Eq. (11), when F_1 increases, γ strictly decreases (linearly in F_1). Thus by Eq. (9) (homogeneous case) and Eq. (13) (general case), any misclassification element in row i also strictly decreases (because $K > 1$). For $F_1 = 1$, $\gamma = 0$ and A' is reduced to its diagonal. ■

D. Simple model

In per-CCA queueing, we now introduce a first simple model of the packet rate of CCA i arriving at queue j. We use the facts that: (1) This rate is the product of the number of flows using CCA i in queue j by their expected packet rate. (2) This number of flows using CCA i in queue j is proportional to a_{ij} by definition. (3) The packet rate of each flow is proportional to its aggressiveness, because the aggressiveness of each flow is the ratio of its packet rate by its fair share. Finally, (4) we approximate the aggressiveness of each flow in each queue as equal to its aggressiveness g_i during training. Thus, the packet rate of CCA i arriving at queue j is proportional to $g_i a_{ij}$. Let p_{ij} denote the probability of packets in queue j to use CCA i. By Bayes' rule, we get:

$$p_{ij} = \frac{g_i a_{ij}}{\sum_i g_i a_{ij}}. \quad (14)$$

Assume that each queue j is serviced at rate c_j (e.g., proportionally to its number of flows). Then the total throughput Thr_i obtained by CCA i across all queues is:

$$Thr_i = \sum_{j=1}^{K} p_{ij} \cdot c_j. \quad (15)$$

E. Advanced aggressiveness model

The simple model above only captures approximately the final fairness, because it uses a single measure of aggressiveness that is obtained during the training, when flows from all CCAs share the same buffer. We want a more precise model to (1) reliably use it as a sub-routine in the experimental optimizations (Sec. III-F), and (2) obtain more intuition. We build this model in a heuristic manner by running offline simulations that examine the CCA interactions. Thus, this advanced model trades off an increased precision against a more heuristic approach.

Relative aggressiveness. In this advanced model, for each CCA pair (i, j) out of the $K(K-1)/2$ possible ones, we model how CCAs i and j behave when they are alone in a shared buffer. To do so, we vary the number of flows from i and j to model a buffer dominated by either of the CCAs. Then, we denote the relative aggressiveness $g_{ij}(x_i)$ as the ratio of the average throughput of a flow of CCA i by that of CCA j, assuming that a proportion x_i of all flows belongs to CCA i. We model $g_{ij}(x_i)$ by first plotting the ratio of their throughputs in log scale as a function of their number of flows, and then using a minimum mean-squares estimator. We get $g_{ij}(x_i) = w \cdot e^{m x_i}$, for some constants w and m.

Advanced model. We now want to model the aggressiveness $g_i(k)$ of a flow with CCA i in queue k, i.e., the ratio of its rate to the average flow rate in queue k. Intuitively, this ratio is the relative aggressiveness against an average flow in the queue. We model it as a weighted average relative aggressiveness, where the weight of each CCA j is its proportion x_j^k of all flows in queue k. Putting it all together,

$$g_i(k) = \sum_{j=1}^{K} x_j^k g_{ij}(x_i^k). \quad (16)$$

Finally, we plug this aggressiveness in Eq. (14) to get the probability distribution of packets in each queue.

F. Model limitations

We introduce the simple and advanced models to evaluate how accurate the classifiers should be in order to reach a reasonable fairness among CCAs. The models are not needed to implement per-CCA queueing, only to get intuition. However, they also have several limitations:

Topology-dependent. If some evaluation settings change (e.g., link speeds, buffer sizes, RTTs, flow arrivals and departures, etc.), then we need to obtain new models for the new settings.

Classifier assumptions. We made simplifying assumptions to obtain the models, and they may need to change based on classifier implementation details.

Theory. We used multiclass learnability theory to characterize the bounds and trends of classifier accuracy as we scale the number of samples. The exact accuracy of current and future classifiers may vary.

III. Evaluations

A. Settings

Topology. To evaluate our aggressiveness models, we use Mininet emulation with iPerf3 traffic generation. We deploy $N = 150$ hosts (with one flow each) in a dumbbell-like topology. Each host sends a long-lasting flow to a single server through two consecutive switches. All hosts are connected to the first switch by one 1 Gbps link with a delay of 10 ms. The two switches are then connected by a bottleneck link of capacity $C = 1.5$ Gbps. The egress bottleneck buffer size at the output of the first switch is $\frac{C \cdot RTT}{\sqrt{N}} = 460$ packets, using $RTT = 40$ ms [30]. Then, the second switch is connected to the server by a final link of delay 10 ms. We run all simulations for 60 sec.

Buffer policy. We consider two modes for the buffer:

- *Buffer sharing*, with a shared buffer for all CCAs.
- *Per-CCA queueing*, with K isolated queues that statically share the buffer.

With per-CCA queueing, packets go through three stages at the switch: (1) *Classification*, assigning them to a given queue; (2) *Buffering*; (3) *Service*. For the *classification*, we first run a classifier offline on each flow to map it to a given CCA, and therefore to a given queue. If we want to simulate a given conditional probability q_{ij} (Eq. (1)), then we randomly classify each flow that uses CCA i as using CCA j by drawing a conditional probability q_{ij}. Next, online, we map the packets of each flow depending on their 5-tuple. In practice, due to the topology, we use the IP source address. Once classified, packets are buffered in one of the K queues, depending on their classified CCA. Finally, we service each queue by dividing the total link capacity proportionally to the number of mapped flows to this queue.

CCAs. We evaluate the same CCAs as those of the DeePCCI and Dragonfly classifiers: Reno, CUBIC and BBR [23], [24].

Starvation avoidance metric. We quantify the CCA starvation avoidance by measuring the percentage of bandwidth that the most vulnerable CCA gets out of its fair share. To do so, for each CCA, we measure the average throughput obtained by all of its flows, which reflects the expected throughput for a flow belonging to this CCA. We then normalize by the fair share, which is the average of all such average CCA throughputs. The result is between 0 and 1, with a 0 value for starvation and 1 for full fairness. Thus, the normalized minimum throughput $MinThr$ is

$$MinThr = \frac{\min_i avgThr_i}{fairShare}, \tag{17}$$

where $avgThr_i$ denotes the average throughput for flows of CCA i.

B. Equal number of flows per CCA

Starvation results. We start with a simple case, with 50 flows per CCA. Fig. 2 illustrates the achieved CCA starvation avoidance by plotting the normalized CCA minimum throughput as a function of the classifier accuracy F_1. First, the blue line for a shared buffer shows that there is little protection for vanilla CCAs from the aggressive BBR, as the least aggressive CCA (Reno) only gets 11.9% of its bandwidth fair share. On the other hand, the red line for per-CCA queueing shows that it improves the fairness and reduces the starvation. This holds even for a relatively low F_1, i.e., even when many BBR flows can enter the queues of Reno and CUBIC. To cross a throughput threshold of 0.5 for the vulnerable CCA, we need $F_1 = 0.83$, a high but reasonable classifier accuracy [23], [24].

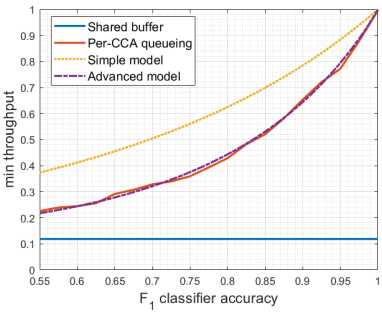


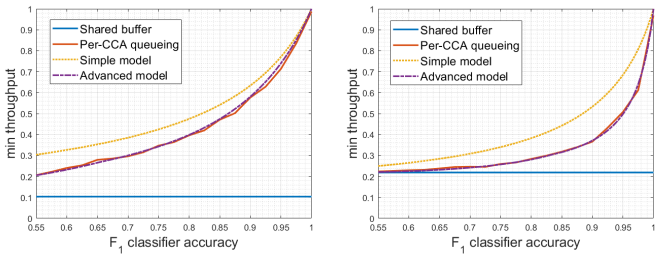
Fig. 2. Equal number of flows per CCA: Normalized minimum CCA throughput as a function of classifier accuracy F_1.

Model results. Fig. 2 also presents the results of our two models. The dotted yellow line with the first simple model shows that it indeed captures the general tendency of per-CCA queueing (red line), growing monotonously until it reaches a result of 1 for an ideal $F_1 = 1$. However, it can get far from the exact result. For instance, with $F_1 = 0.83$, it predicts a normalized minimum throughput of 0.66 while the exact result is 0.5. The dashed purple line shows the advanced model. It appears as extremely accurate.

C. Different number of flows per CCA

Few sharks. We consider the general case with a different number of flows for each CCA. We start by testing the case of many vulnerable flows and a few aggressive ones. We assume that 130 flows use Reno, 10 CUBIC, and 10 BBR. Fig. 3(a) shows how the most vulnerable CCA (Reno) only gets 10.4% of its fair share in a shared buffer. Per-CCA queueing doubles its share for $F_1 = 0.55$, and its share further increases with F_1. Moreover, the advanced model obtains very close results.

Many sharks. We now assume that 10 flows use Reno, 10 CUBIC, and 130 BBR. Fig. 3(b) illustrates how the most vulnerable CCA now gets 22% of its fair share with a shared buffer, a higher figure than in the previous examples. This is due to the fact that many BBR flows compete with each other, and therefore BBR flows cannot get too large and too aggressive. In per-CCA queueing, many BBR flows are misclassified and sent to the Reno and CUBIC queues, decreasing the isolation. Therefore, per-CCA queueing now needs a higher F_1 to reach an elbow and start increasing towards 1. To get a throughput of 0.5, it needs $F_1 = 0.95$. The advanced model obtains again very close results.



(a) Few aggressive flows.

(b) Many aggressive flows.

Fig. 3. Different number of flows per CCA.

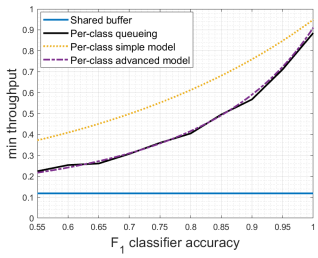


Fig. 5. Per-CCA-class queues.

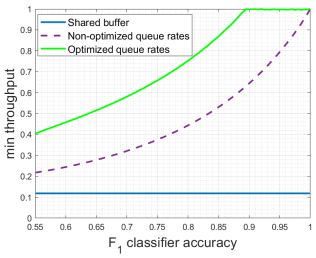
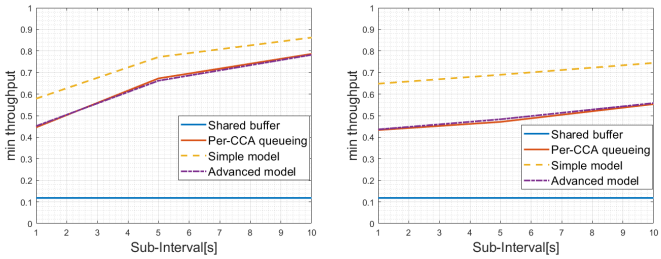


Fig. 6. Optimizing service rates.

TABLE I. Used F_1 scores from DeePCCI [23] and Dragonfly [24].

	DeePCCI			Dragonfly		
Duration[s]	Reno	CUBIC	BBR	Reno	CUBIC	BBR
1	0.666	0.5833	0.95	0.798	0.8733	0.91
5	0.84	0.8125	0.975	0.88	0.94	0.945
10	0.875	0.875	0.9875	0.93	0.935	0.95



(a) DeePCCI [23] CCA classifier.

(b) Dragonfly [24] CCA classifier.

Fig. 4. Fairness given F_1 accuracy of real CCA classifiers.

D. Evaluating classifiers from the literature

Until now, we assumed that the classifier satisfies our multiclass classification model, with the classification error following the inverse-square-root of the aggressiveness (Eq. (7)). In this part only, we want to evaluate the impact of using the real F_1 values of two different classifiers from the literature: DeePCCI [23] and Dragonfly [24].

Setup. We use 50 hosts per CCA. We further use the F_1 scores achieved by DeePCCI and Dragonfly (Table I) for a network with Reno, CUBIC, and BBR flows, while the classifier interval durations for classification are 1, 5, and 10 seconds. These values are taken from their respective papers.

Results. Fig. 4(a) and Fig. 4(b) show how the fair share of the most vulnerable flow varies as a function of the sampling interval of the classifiers. Per-CCA queueing clearly outperforms the shared buffer. The advanced model captures the performance quite accurately.

E. Modeling per-CCA-class queueing

We now examine per-CCA-class queueing, as in P4air [15].

Setup. We use 50 hosts per CCA. However, we merge the Reno and CUBIC queues into a single queue, using two queues instead of three. The merged queue includes all flows classified as Reno or CUBIC, including misclassified BBR flows.

Results. Fig. 5 shows that per-CCA-class queueing reduces starvation. For instance, with $F_1 = 0.6$, the most vulnerable CCA gets 21.7% of its fair share of the throughput, which nearly doubles the 11.9% in the shared buffer. However, per-CCA-class queueing presents a fundamental issue: With an ideal classification ($F_1 = 1$), it only reaches a min throughput of 0.88, and cannot obtain a full fairness among CCAs. This is because Reno and CUBIC share the same queue, and they have a slightly different aggressiveness. Thus, it is fundamentally weaker than per-CCA queueing (Fig. 2). This weakness is expected to significantly grow when more dissimilar CCAs are grouped together (e.g., grouping BBR and Reno in the same queue would lead to Reno getting crushed). Last, our advanced model closely approximates per-CCA-class queueing.

F. Leveraging the advanced model to optimize fairness

Since the advanced model closely captures the fairness of per-CCA queueing, we can leverage it to optimize the fairness. In particular, until now, we assumed that we service all queues with a rate proportional to their number of mapped flows. But the aggressiveness of BBR hurts the expected throughput of the Reno and CUBIC flows. If we provide a higher service rate to the Reno and CUBIC queues, we can counteract this aggressiveness and increase the expected throughput of the more vulnerable flows. We want to fundamentally analyze how much we can improve by varying the service rates.

Setup. We run a theoretical optimization using the advanced model (in Matlab, not Mininet). We assume the same number of flows for each CCA. We adjust the service rate of each queue to maximize the fair share of the most vulnerable CCA in the advanced model.

Results. Fig. 6 shows how the throughput of the most vulnerable CCA exceeds the previous results. For instance, for $F_1 = 0.6$, the most vulnerable CCA gets 46% of its fair share, exceeding the 26% that it obtains in unoptimized per-CCA queueing, and more than doubling the shared-buffer throughput. One of the most surprising results in the figure is that *it is even possible to reach the CCA fair share despite an imperfect classification*, starting from $F_1 = 0.88$.

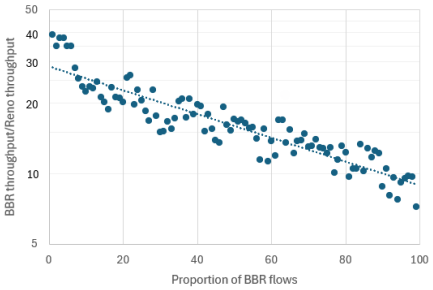


Fig. 7. (BBR, Reno) advanced aggressiveness.

G. Deep dive into the aggressiveness models

For completeness, we describe how we determined the parameters for the aggressiveness models.

Simple model. We evaluate the training-mode aggressiveness g_i from Eq. (6). This aggressiveness determines how many samples the classifier can see for each CCA, and therefore how efficiently it classifies this CCA. We assign 50 hosts to each CCA, and run the evaluation with a shared buffer (sharing mode). We repeat the evaluation 30 times.

The resulting average and median aggressiveness are almost the same. We obtain:

- $g_{Reno} = 0.1158$,
- $g_{CUBIC} = 0.1365$, and
- $g_{BBR} = 2.7477$.

In other words, BBR flows capture 91.6% of the bandwidth, while leaving the remainder to the other flows. BBR is 23.7 times more aggressive than Reno and 20 times more aggressive than CUBIC.

Advanced model. We measure the relative aggressiveness of each CCA i vs. CCA j. We define 100 hosts per CCA, and repeat each evaluation 5 times using a shared buffer. We obtain the following three relative-aggressiveness models:

- $g_{BBR,Reno}(x) = 28.7 \cdot e^{-1.17x}$,
- $g_{BBR,CUBIC}(x) = 33.2 \cdot e^{-1.15x}$, and
- $g_{CUBIC,Reno}(x) = 0.99 \cdot e^{0.38x}$.

For example, Fig. 7 plots the ratio of the average throughput of BBR flows to that of Reno flows, as a function of the percentage of BBR flows. As previously shown in the literature [1], [31], when the percentage of BBR flows increases in a shared buffer, BBR becomes less aggressive. We can see that BBR's relative aggressiveness vs. Reno decreases about linearly in logarithmic scale, from 28.7 to 8.9.

REFERENCES

[1] R. Ware, M. K. Mukerjee, S. Seshan, and J. Sherry, "Modeling BBR's interactions with loss-based congestion control," in *ACM IMC*, 2019.

[2] S. Utsumi and G. Hasegawa, "Improving inter-protocol fairness based on estimated behavior of competing flows," in *IFIP Networking*, 2022, pp. 1–9.

[3] A. Philip, R. Ware, R. Athapathu, J. Sherry, and V. Sekar, "Revisiting TCP congestion control throughput models & fairness properties at scale," in *ACM IMC*, 2021, pp. 96–103.

[4] V. Arun, M. Alizadeh, and H. Balakrishnan, "Starvation in end-to-end congestion control," in *ACM SIGCOMM*, 2022, p. 177–192.

[5] L. Brown, A. Gran Alcoz, F. Cangialosi, A. Narayan, M. Alizadeh, H. Balakrishnan, E. Friedman, E. Katz-Bassett, A. Krishnamurthy, M. Schapira, and S. Shenker, "Principles for internet congestion management," in *ACM SIGCOMM*, 2024.

[6] K. Toledo, D. Breitgand, D. Lorenz, and I. Keslassy, "CloudPilot: Flow acceleration in the cloud," *Computer Networks*, no. 224, p. 109610, 2023.

[7] C. X. Cai, F. Le, X. Sun, G. G. Xie, H. Jamjoom, and R. H. Campbell, "CRONets: Cloud-routed overlay networks," in *IEEE ICDCS*, 2016.

[8] A. Markuze, A. Bergman, C. Dar, I. Keslassy, and I. Cidon, "Kernels of splitting TCP in the clouds," *Netdev 0×14*, 2020.

[9] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, and I. Stoica, "FairCloud: Sharing the network in cloud computing," in *ACM SIGCOMM*, 2012, pp. 187–198.

[10] N. K. Sharma, M. Liu, K. Atreya, and A. Krishnamurthy, "Approximating fair queueing on reconfigurable switches," in *Usenix NSDI*, 2018, pp. 1–16.

[11] Z. Yu, J. Wu, V. Braverman, I. Stoica, and X. Jin, "Twenty years after: Hierarchical core-stateless fair queueing," in *Usenix NSDI*, 2021, pp. 29–45.

[12] E. Zahavi, A. Shpiner, O. Rottenstreich, A. Kolodny, and I. Keslassy, "Links as a Service (LaaS): guaranteed tenant isolation in the shared cloud," in *IEEE JSAC*, vol. 37, no. 5, 2019.

[13] L. Brown, G. Ananthanarayanan, E. Katz-Bassett, A. Krishnamurthy, S. Ratnasamy, M. Schapira, and S. Shenker, "On the future of congestion control for the public internet," in *ACM HotNets*, 2020, pp. 30–37.

[14] L. Yu, J. Sonchack, and V. Liu, "Cebinae: scalable in-network fairness augmentation," in *ACM SIGCOMM*, 2022, pp. 219–232.

[15] B. Turkovic and F. Kuipers, "P4air: Increasing fairness among competing congestion control algorithms," in *IEEE ICNP*, 2020, pp. 1–12.

[16] Z. Meng, N. Atre, M. Xu, J. Sherry, and M. Apostolaki, "Confucius queue management: Be fair but not too fast," *arXiv preprint arXiv:2310.18030*, 2023.

[17] E. Kfoury, J. Crichigno, and E. Bou-Harb, "P4cci: P4-based online TCP congestion control algorithm identification for traffic separation," in *IEEE ICC*, 2023, pp. 4007–4012.

[18] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker, "On the characteristics and origins of internet flow rates," in *ACM SIGCOMM*, 2002, pp. 309–322.

[19] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley, "Inferring TCP connection characteristics through passive measurements," in *IEEE Infocom*, vol. 3, 2004, pp. 1582–1592.

[20] J. Oshio, S. Ata, and I. Oka, "Identification of different TCP versions based on cluster analysis," in *IEEE ICCCN*, 2009, pp. 1–6.

[21] D. H. Hagos, P. E. Engelstad, A. Yazidi, and Ø. Kure, "Recurrent neural network-based prediction of TCP transmission states from passive measurements," in *IEEE NCA*, 2018, pp. 1–10.

[22] D. H. Hagos, P. E. Engelstad, and A. Yazidi, "Classification of delay-based TCP algorithms from passive traffic measurements," *IEEE NCA*, 2019.

[23] C. Sander, J. Rüth, O. Hohlfeld, and K. Wehrle, "DeePCCI: Deep learning-based passive congestion control identification," in *Workshop on Network Meets AI & ML*, 2019, pp. 37–43.

[24] D. Carmel and I. Keslassy, "Dragonfly: In-flight CCA identification," *IEEE Transactions on Network and Service Management*, vol. 21, no. 3, 2024.

[25] H. Wang, H. Lin, Z. Zhong, T. Yang, and M. Shahzad, "Enhanced machine learning sketches for network measurements," *IEEE Transactions on Computers*, vol. 72, no. 4, pp. 957–970, 2022.

[26] S. Boucheron, O. Bousquet, and G. Lugosi, "Theory of classification: A survey of some recent advances," *ESAIM: probability and statistics*, vol. 9, pp. 323–375, 2005.

[27] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of machine learning*. MIT Press, 2018.

[28] R. Nowak. (2007) Statistical learning theory: Classification error bounds. [Online]. Available: https://nowak.ece.wisc.edu/SLT07/lecture8.pdf

[29] A. Balsubramani, S. Dasgupta, Y. Freund, and S. Moran, "An adaptive nearest neighbor rule for classification," *NeurIPS*, vol. 32, 2019.

[30] N. McKeown, G. Appenzeller, and I. Keslassy, "Sizing router buffers (redux)," *ACM SIGCOMM Computer Communication Review*, vol. 49, no. 5, pp. 69–74, 2019.

[31] D. Scholz, B. Jaeger, L. Schwaighofer, D. Raumer, F. Geyer, and G. Carle, "Towards a deeper understanding of TCP BBR congestion control," in *IFIP Networking*, 2018, pp. 1–9.