LFQ: Online Learning of Per-flow Queuing Policies using Deep Reinforcement Learning

Maximilian Bachl, Joachim Fabini, Tanja Zseby
Technische Universität Wien; Vienna, Austria; firstname.lastname@tuwien.ac.at

***Abstract*—The increasing number of different, incompatible congestion control algorithms has led to an increased deployment of fair queuing. Fair queuing isolates each network flow and can thus guarantee fairness for each flow even if the flows' congestion controls are not inherently fair. So far, each queue in the fair queuing system either has a fixed, static maximum size or is managed by an Active Queue Management (AQM) algorithm like CoDel. In this paper we design an AQM mechanism (Learning Fair Qdisc (LFQ)) that dynamically learns the optimal buffer size for each flow according to a specified reward function online. We show that our Deep Learning based algorithm can dynamically assign the optimal queue size to each flow depending on its congestion control, delay and bandwidth. Comparing to competing fair AQM schedulers, it provides significantly smaller queues while achieving the same or higher throughput.**

I. Introduction

Given the high throughput and low queuing delay that is required by emerging cloud gaming and virtual reality applications, effective congestion control and the optimal management of each flow's queue become increasingly relevant questions. New Congestion Control Algorithms (CCAs) for TCP and QUIC are still being introduced, revisiting old ideas but also introducing new concepts [6], [5], [2]. Some of them, such as BBR, are already being used in real-world production systems. While new CCAs are commonly being designed with compatibility to other CCAs in mind, often they do not share the link completely fairly with older CCAs [8]. Besides that, network flows using the same CCA can also be unfair to each other: For example, BBR favors flows with a high Round Trip Time (RTT), while New Reno favors those with a low one [13]. This unfairness can be mitigated by using Fair Queuing (FQ) at the bottleneck link, isolating each flow from all other flows and assigning each flow an equal share of bandwidth [7].

While FQ solves the fairness problem, the optimal size of each of the flow's queues is still a problem. We thus propose an algorithm that fingerprints each flow dynamically and assigns the optimal buffer size to it according to a reward function. The mechanism that fingerprints the flows and outputs the correct buffer size is based on Reinforcement Learning (RL) and Deep Learning (DL).

To make our work reproducible and to encourage further research, we make the code, figures and trained neural network weights of this work publicly available[1]. For more details, we also provide a longer version of this paper[2].

[1]https://github.com/CN-TU/reinforcement-learning-for-per-flow-buffer-sizing
[2]https://arxiv.org/abs/2007.02735

II. Related Work

[11] use DL to recognize flows' CCA, showing that DL is feasible to fingerprint flows.

[1] use a hand-crafted algorithm to optimize the queuing behavior for different CCAs. It is, however, not guaranteed that this algorithm works for new CCAs and traffic patterns.

[4] develop a steering system for Active Queue Management (AQM), which takes as the input a target utilization (like 95%) and then tries to adjust the queue so that this target utilization is met. It operates on shared buffers and employs no Machine Learning (ML) and fingerprinting of flows.

[9] use Deep Reinforcement Learning to develop an AQM mechanism for IoT devices that works on the output queue of a switch. [3] use RL to build an AQM algorithm for shared buffers. [10] use a custom optimization approach that relies on an offline greedy algorithm which finds optimal rules for an AQM that fulfills a certain reward function.

Unlike our proposed approach, these works' aim is not fingerprinting each flow but instead they optimize the overall queue that is shared by flows. In our work we focus on fingerprinting each flow and providing the optimal queuing behavior for each flow individually. This direction has, to our knowledge, not been explored so far.

III. Concept

If the buffer for a flow is too small, a flow cannot achieve the full bandwidth. On the other side, if the buffer is too large, the flow can achieve the full bandwidth but a standing queue exists, which causes unnecessary delay. Because the optimal buffer size for many CCAs depends on the Bandwidth Delay Product (BDP) and thus on the bandwidth and the delay (RTT), for example, a flow only having half the RTT of another one only needs half its buffer size to achieve full bandwidth.

To learn an optimal AQM policy, we specify a reward function, which the ML based AQM should learn to optimize for each flow using RL. A simple and effective reward function is for example $\textit{Reward} = \textit{bandwidth} - \alpha \times \textit{queue size}$. In this reward function, the choosable parameter α specifies the tradeoff between the bandwidth and the buffer size. With α going to zero, the optimum policy approaches the one in which the buffer size is large enough that a flow never underutilizes the link but at the same time the buffer is never going to be larger than necessary if this doesn't provide a benefit in throughput.

As the input for the neural network we use the following features, which are updated continuously when receiv-

ing/sending a packet at the queue: • queue size • standard deviation of the queue size • maximimum buffer size • rate of incoming data • rate of outgoing data • time since the last packet loss

We do not use these features directly because this could result in large, spontaneous variation in the input features. For example, packets might not be received in regular intervals but in bursts. Thus, instead we use 10 exponentially weighted averages of each feature with weights of 2^{-4}, 2^{-5}, ... , 2^{-13}. We thus get a vector of $6 \times 10 = 60$ features at each point in time. The features are fed into a neural network which has one output: The deemed optimal queue size.

IV. Offline Learning

First, we implement an ML system that is capable of learning an optimal queuing policy in a simulator. The idea is that in an ideal setting training is faster. After training in a simulator, the finished neural network can be deployed in a real production setting. The training procedure involves the following steps:

1) Draw a random sample of bandwidths, delays, CCA and flow durations.
2) Simulate each flow concurrently, compute the feature vector continuously and let the neural network output its optimal buffer size each time a new packet arrives.
3) During each flow, at a random time between 0 and half the flow length in seconds, perform the experiment: Fork the simulation process for each flow and continue one simulation with the current buffer size $+1$ packet (option A) and one with the current buffer size -1 packet (option B) until the end of the flow. This procedure is commonly called *A/B testing*.
4) Wait until all flows are finished
5) Check for each flow which of the two versions performed better ($+1$ or -1) regarding the reward function.
6) Update the neural network so that it learns to output the better buffer size when being fed the inputs as they were at the time at which the experimentation started.
7) Start the next iteration.

During deployment, the trained neural network is used and the experiment step is skipped.

A. $\alpha = 0.01$

For offline learning with $\alpha = 0.01$, the neural network needs around 250000 training flows to converge.

Figure 1 shows a New Reno flow at a link with a bottleneck bandwidth of 25 Mbit/s and a delay of 15 ms controlled by LFQ after it is fully trained. In the beginning, the maximum queue length that LFQ allows is around 15 packets. After less than a second, when the DL learns to understand the flow's congestion control, the bottleneck delay and bandwidth, it understands that a larger maximum queue is needed and raises its prediction to around 35 packets. It stays at this value until the end. The queue never becomes empty and no standing queue is formed at any point, which means that LFQ achieved its goal.

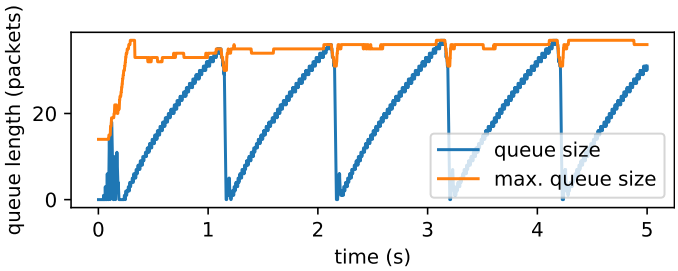

Figure 1: New Reno flow controlled by Learning Fair Qdisc (LFQ) (offline training, $\alpha = 0.01$); link speed: 25 Mbit/s; delay: 15 ms. The queue never becomes empty and also never forms a standing queue.

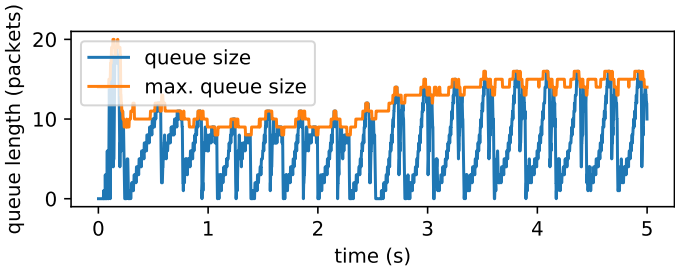

Figure 2: BIC flow controlled by LFQ (offline training, $\alpha = 0.01$); link speed: 25 Mbit/s; delay: 15 ms. The queue never becomes empty and also never forms a standing queue.

Figure 2 shows that LFQ learned that the flow uses BIC and thus needs a smaller buffer: For the same scenario with New Reno (Figure 1) LFQ output a queue length of around 35. For the same scenario with a BIC controlled flow, it chooses the optimal maximum queue length as around 12 packets. This demonstrates that LFQ learned to distinguish different congestion controls only by looking at the pattern of a flow.

Table I: Correlation between bandwidth/delay of the link and output buffer size; tradeoff: 0.1; offline learning. Link speed: 5-25 Mbit/s; delay: 5-25 ms. When bandwidth is varied, delay is kept at its mean (15 ms) and vice versa.

CCA	bandwidth	delay
New Reno	98.3%	99.1%
BIC	80.2%	90.2%

To have a quantifiable measure of the success of LFQ, we compute the correlation between a change of bandwidth/delay and the maximum queue size output by the neural network. If the neural network learned ideally, the correlation obtained would be 1 (100%) as, as mentioned in section III, with a larger delay/bandwidth the BDP is larger and a larger buffer is necessary. Table I shows that the for New Reno the correlation is very high while for BIC it is high but not as good as for New Reno. We attribute this to the fact that for BIC the buffer that is required is generally smaller (only a couple of packets) and thus fluctuations in the training process are more pronounced.

The correlations imply that, when increasing the bandwidth,

the neural network outputs a larger maximum queue size, because a larger buffer is needed for New Reno and BIC, when the bandwidth is larger. However, learning the relationship is not difficult: In fact, the neural network would simply need to learn to use the identity function to map the input feature that encodes the incoming or outgoing data rate to the output (the output maximum buffer size).

The results also show that increasing the delay increases the output (maximum buffer size) of the neural network. However, no feature directly encodes the delay. Thus, the neural network must observe the pattern of the CCA, estimate the RTT and output the suitable maximum queue size. The right column of Table I shows that there is clearly a linear relationship between the delay and the output maximum queue size as the correlation is close to 1 (or 100%).

The average output maximum queue size is more than double for New Reno (22 packets) than for BIC (8 packets). This shows that the neural network also learned to distinguish between these two CCAs. Since there are no features that directly indicate the CCA, the neural network must have learned these by combining the other features and observing distinct behavior for New Reno and BIC. This is expected, since BIC has a different multiplicative decrease factor than New Reno (0.7 vs 0.5) the neural network should learn to output different maximum queue sizes for these two. Specifically, we would expect BIC to need a smaller buffer as its decrease is smaller, thus requiring a smaller buffer.

B. $\alpha = 10$

As for the smaller alpha, the neural network converges after a couple of 100000 flows.

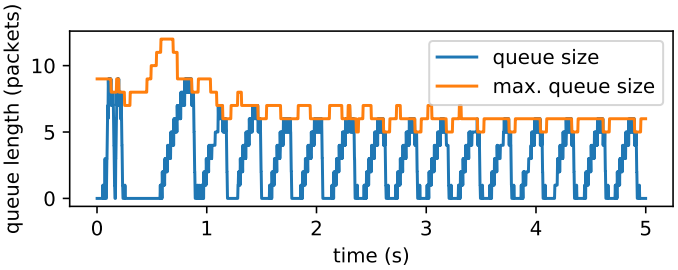

Figure 3: New Reno flow controlled by LFQ (offline training, $\alpha = 10$); link speed: 6 Mbit/s; delay: 15 ms. The queue occasionally becomes empty because with a higher tradeoff α, LFQ favors a small queue more than high throughput.

The flow in Figure 3 gets assigned a small buffer and sometimes the queue becomes empty, which never happens for the flow in Figure 2. This shows that the parameter α has a clear influence on the learning process and that with a larger α, throughput is sacrificed to keep the queue smaller.

Table II shows that correlations are still very high albeit a bit lower than for $\alpha = 10$ (Table I). The average buffer size is 13 packets for New Reno and BIC 5 for BIC. This is significantly smaller than the values for the smaller α, meaning that with a larger tradeoff parameter, the neural network learns that smaller buffers are better on average, which is exactly what is expected.

Table II: Correlation between bandwidth/delay of the link and output buffer size; tradeoff: 10; offline learning. Link speed: 5-25 Mbit/s; delay: 5-25 ms. When bandwidth is varied, delay is kept at its mean (15 ms) and vice versa.

CCA	bandwidth	delay
New Reno	99.3%	96.5%
BIC	75.4%	62%

V. Online Learning

In contrast to offline learning, online learning enables to train in deployment. This makes it possible to adapt the behavior slowly over time (over a time span of months or years), for example if new CCAs emerge. Furthermore, being able to training with real flows is more realistic than training only with simulated flows in a simulation.

The major change is that for online learning, A/B testing, like for the offline learning, is not possible: In the real world, each time the buffer size is updated for a flow, this is a definite decision. It is not possible to artificially "split" a flow like in the simulator and check if a larger or a smaller buffer would have been a better choice. Another analogy for this is that for online learning, we have to unveil what happened to Schrödinger's cat (if it is dead or not), when we launch the experiment, which buffer size is better. For the offline learning, on the other side, we can try out both versions in the simulator and continue one reality with the cat being dead (maximum buffer size -1) and the other one with the cat being alive (maximum buffer size $+1$).

To accommodate for this difference from offline learning, for online training we need two neural networks: • An *actor network*, which outputs the optimal buffer size at each time step. • A *critic network*, which outputs the reward that it predicts is going to be achieved when keeping the current buffer size until the end of the flow.

The training procedure is identical to the one used for offline learning section IV except that the critic network is used to determine if the choice of the buffer size was better or worse than expected.

A. $\alpha = 10$

Compared to offline learning, it takes significantly more network flows for training to converge when doing online learning. This is not surprising since for online training there are two neural networks involved and there is inherently more noise: For the offline training, we can perform A/B testing and thus the reward given to the neural network is always correct. On the other side, for online learning, the reward depends also on the output of the critic network and the critic network doesn't have to be 100% correct when it outputs the reward that it expected. Thus, the more noisy reward for online training makes it converge more slowly. Specifically, this means that first, the neural network learns that a buffer

size of around 10 is optimal on average for all flows and only very slowly (when compared to the offline learning) it learns to fingerprint and differentiate the different flows with different conditions such as different link speed, different delay and different CCA.

Table III: Correlation between bandwidth/delay of the link and output buffer size; tradeoff: 10; online learning. Link speed: 5-25 Mbit/s; delay: 5-25 ms. When bandwidth is varied, delay is kept at its mean (15 ms) and vice versa.

CCA	bandwidth	delay
New Reno	86.1%	87.7%
BIC	72.9%	73.6%

Table III shows that online learning also learns the expected mapping between CCA, bandwidth and delay to buffer size, albeit training takes longer and results are more noisy than for online learning as can be seen in the lower correlations when comparing Table II with Table III. The average buffer size is 14 for New Reno and 8 for BIC, which is very similar to the results of the offline learning. This demonstrates that online learning learns similar policies compared to offline learning.

VI. Comparing with other queue managers

In this section we compare the behavior of LFQ to the behavior of other popular fair queue discs: • *fq* [7], which is a common fair queue disc for Linux which has a drop tail buffer of fixed size for each flow (Fifo) with two different buffer sizes: 100 and 1000, which both are common default buffer sizes. • *FqCoDel* [12], which uses fq but manages each queue with the *CoDel* algorithm.

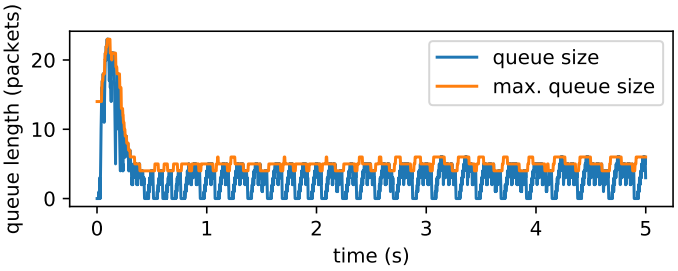

Figure 4: BIC flow controlled by LFQ (offline training, $\alpha = 0.01$); link speed:15 Mbit/s; delay: 5 ms. The queue never becomes empty and also never forms a standing queue.

The comparison of LFQ (Figure 4) with FqCoDel (Figure 5) shows that FqCoDel generally avoids a large standing queue but results in a very large spike in queued packets at the beginning of the flow. Thus, while CoDel was designed to avoid "bufferbloat" it actually induces a very large standing queue during the slow start phase of a flow.

The systematic comparison (Table IV) between the different fair AQM mechanisms shows that LFQ achieves about the same or better throughput than the competing algorithms while having a significantly smaller average queue (less than half)

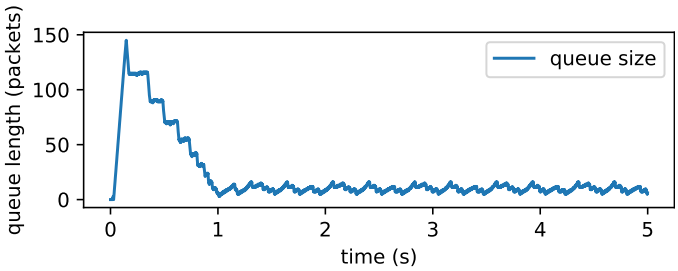

Figure 5: Bic flow controlled by FqCoDel; link speed: 15 Mbit/s; delay: 5 ms.

Table IV: Average throughput and maximum/average queue size for different AQMs. Link speed: 5-25 Mbit/s; delay: 5-25 ms for both New Reno and Bic. When bandwidth is varied, delay is kept at its mean (15 ms) and vice versa.

	avg. throughp.	queue size max.	queue size avg.
LFQ, offline $\alpha = 0.01$	13.4	23.9	7.7
LFQ, offline $\alpha = 10$	12.5	12.7	3.4
LFQ, online $\alpha = 10$	12.8	16.1	4.5
FqCoDel	13.7	155.4	15.4
fq 100	11.7	100	51.1
fq 1000	11.9	1000	630.4

and an even more pronounced reduction in maximum queue when compared to fq and FqCoDel.

References

[1] M. Bachl, J. Fabini, and T. Zseby, "Cocoa: Congestion Control Aware Queuing," in *Workshop on Buffer Sizing*. ACM, 2019.

[2] M. Bachl, T. Zseby, and J. Fabini, "Rax: Deep Reinforcement Learning for Congestion Control," in *ICC*. IEEE, 2019.

[3] S. K. Bisoy, P. K. Pandey, and B. Pati, "Design of an active queue management technique based on neural networks for congestion control," in *ANTS*, 2017.

[4] R. Bless, M. Hock, and M. Zitterbart, "Policy-oriented AQM Steering," in *IFIP Networking*, 2018.

[5] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-Based Congestion Control," *ACM Queue*, 2016.

[6] M. Dong, T. Meng, D. Zarchy, E. Arslan, Y. Gilad, P. B. Godfrey, and M. Schapira, "PCC vivace: online-learning congestion control," in *NSDI*, 2018.

[7] E. Dumazet, "pkt_sched: fq: Fair Queue packet scheduler [LWN.net]," 2013. [Online]. Available: https://lwn.net/Articles/565421/

[8] F. Fejes, G. Gombos, S. Laki, and S. Nádas, "Who will Save the Internet from the Congestion Control Revolution?" in *Workshop on Buffer Sizing*. ACM, 2019.

[9] M. Kim, "Deep Reinforcement Learning based Active Queue Management for IoT Networks," PhD Thesis, 2019.

[10] X. Lin and D. Zhang, "Kemy: An AQM generator based on machine learning," in *ChinaCom*. IEEE, 2015.

[11] C. Sander, J. Rüth, O. Hohlfeld, and K. Wehrle, "DeePCCI: Deep Learning-based Passive Congestion Control Identification," in *NetAI*. ACM, 2019.

[12] D. Taht, T. Hoeiland-Joergensen, J. Gettys, E. Dumazet, and P. McKenney, "The Flow Queue CoDel Packet Scheduler and Active Queue Management Algorithm," 2018. [Online]. Available: https://tools.ietf.org/html/rfc8290

[13] B. Turkovic, F. A. Kuipers, and S. Uhlig, "Interactions between Congestion Control Algorithms," in *TMA*, 2019.