Age-Based CoDel and His Friends. Improving the Linux Scheduler with 2-Level AQM Disciplines

Giovanni Moschini
Ca' Foscari University of Venice
Venice, Italy
879808@stud.unive.it

Andrea Marin
Ca' Foscari University of Venice
Venice, Italy
marin@unive.it

Leonardo Maccari
Ca' Foscari University of Venice
Venice, Italy
leonardo.maccari@unive.it

***Abstract*—We study the combination of Active Queue Management (AQM) techniques and age-based scheduling disciplines to improve network performance in the Linux packet scheduler. While AQM algorithms like CoDel, FQ-CoDel, and PIE manage queue lengths and ensure fair bandwidth sharing, age-based scheduling (e.g., Least Attained Service, Two-Level Processor Sharing) reduces mean flow completion time by prioritizing short flows. While their individual benefits are known, the integration of these orthogonal features had not been analysed so far. We propose a hybrid queuing solution using standard Linux utilities (`tc`, `nftables`) tested with state-of-the art open source traffic generators to generate realistic Pareto-distributed traffic. Our experiments on 1 Gb/s and 10 Gb/s links demonstrate that implementing age-based scheduling policies on top of AQM strategies significantly reduces flow completion time, with gains ranging from 3% to 20%. These improvements are achieved without increasing CPU load and while preserving or improving fairness among flows. The solution is easily implementable on any recent Linux kernel, and we provide all the code we realized to replicate and improve our results.**

***Index Terms*—aqm, linux, performance**

I. Introduction

The bufferbloat problem is a very well known issue that affects the Internet speed, rooted in the original multi-layered design of TCP/IP [1], [2]. Several Active Queue Management (AQM) algorithms, such as CoDel [3] and FQ-CoDel, have been proposed and adopted to tackle bufferbloat, and some of them matured enough to be included in recent versions of the Linux kernel. These algorithms dynamically manage queue lengths and provide mechanisms for fair bandwidth sharing among competing flows, but typically do not perform advanced scheduling beyond fair queueing or diffserv.

In parallel, research has demonstrated that age-based scheduling disciplines can significantly reduce the mean flow completion time. With age-based scheduling, a higher priority is given to flows in their initial life, which favors short flows compared to long ones. Since the size distribution of the typical Internet traffic is heavy-tailed, this reduces the overall average completion time. Algorithms such as Least Attained Service [4] and Two-Level Processor Sharing [5] are notable examples.

Despite the individual benefits of AQM and age-based scheduling, their combination has not been analysed in the literature. This work addresses this gap by investigating how AQM techniques and age-based scheduling can be combined to leverage the strengths of both approaches. Since these are orthogonal features of queueing algorithms, their integration is both logical and potentially beneficial. To evaluate this combination, we developed custom queueing solutions using the standard Linux utilities and benchmarked them against each other. The analysis focused on Flow Completion Time and its fairness across different algorithms, comparing age-based versus non-age-based solutions, queues with and without AQM, and flow-queueing versus single-queue configurations.

Our work is deeply rooted in the currently available possibilities of the Linux kernel. Our first scientific goal is to show the benefit of age-based and AQM scheduling applied together, our second goal is to provide a solution that can be easily implemented on any recent Linux kernel, without having to modify the kernel itself. This not only guarantees scientific reproducibility, but also immediate applicability in real situations. To obtain this, we used tools like traffic control (`tc`) and nftables (`nft`) Linux commands to rapidly prototype hybrid age-based AQM disciplines on a stable Linux distribution. Nftables was used to direct packets into the appropriate priority band based on the number of bytes already sent by a flow, effectively implementing two-level processor sharing queues (2LPS). Multi-band queues were assembled using `tc-prio` in combination with other disciplines such as `tc-codel`. To test our results we extended the Antler network testing tool, to generate traffic with arbitrary distributions, including Pareto-distributed flows, enabling the evaluation of these queueing configurations under realistic workload conditions[1].

We tested 6 different AQM policies with or without age-based strategies, on dedicated links of 1 Gb/s and 10 Gb/s. Our results show that implementing age-based scheduling policy on top of the current state-of-the-art AQM strategies can strongly reduce flow completion time while preserving fairness among the competing flows. The gain ranges from 3% to 20% depending on the ACM policy and it is obtained without increasing the CPU load of the server. Our contributions can

[1]The Antler code is available at: https://github.com/UniVe-NeDS-Lab/antler, while the code necessary to set up the system is available at: https://github.com/UniVe-NeDS-Lab/linux_age_based_scheduler with a short how-to.

then be summarized as follows:

- We propose to use of AQM policies together with age-based scheduling. We show that with high workload this configuration introduces a significant performance improvement, without impacting fairness or the CPU load.
- We publish all the open source tools and code, so that our solution can be easily reproduced and implemented in any real world setting, or lab.

II. A Primer on AQM For the Linux Kernel

In this section, we first describe AQM techniques. Note that AQM has been a florid research area in the past decades, so we focus only on the proposals that reached a level of maturity to be included in the Linux kernel. All the techniques are applied to the egress interface of any Linux machine.

A. AQM

Traditional drop-tail algorithms only drop packets when the buffer is completely full. This behavior causes buffers at bottleneck nodes to remain persistently full, leading to increased congestion and delay. In contrast, AQM algorithms prevent this situation by pre-emptively dropping packets before the buffer overflows. The need for AQM has long been recognized, initially as a solution to flow synchronization and persistently full queues [6], and more recently as a response to the bufferbloat problem [1], [2].

Random Early Detection (RED) [7] was one of the first algorithms developed to combat bufferbloat and was strongly recommended by the first RFC on AQM [6]. It uses predictive models based on the current amount of buffered packets to decide when to drop. While RED can be theoretically effective, its adoption has been limited by the difficulty of configuring it correctly and the significant performance issues that can arise from misconfiguration. More recent techniques addressed these issues.

1) Controlled Delay (CoDel): The Controlled Delay (CoDel) [3], [8] AQM strategy is a modern algorithm designed to provide a parameterless solution that can be easily deployed on typical Internet nodes. CoDel recognizes "good" and "bad" queues and treats them differently. As buffers are meant to absorb and smooth out transient traffic spikes, it is acceptable for a flow to build up a queue temporarily to maintain high utilization (a "good" queue). However, a queue that persists for longer than the network round-trip time of the flow suggests that an unnecessary amount of packets is being buffered, and congestion should be signaled (a "bad" queue). CoDel measures congestion using packet sojourn time: the amount of time each packet spends in the queue from enqueue to dequeue. More specifically, CoDel tracks the minimum packet sojourn time over a configurable `interval`, chosen to be long enough for a "good" queue to drain. The authors of CoDel exploit the definition of "*power*", by Kleinrock [9], and try to maximize its value, based on the expected round-trip-time. For typical Internet links the round-trip-times fall between 20 and 200 ms, then, the recommended parameters are an `interval` of 100 ms and a `target` of 5 ms.

When congestion is detected, CoDel uses a control loop implemented as a state machine with a nominal state and a drop state. If the minimum delay stays above the `target` for more than an `interval`, CoDel enters the drop state, which is exited only when a packet with less than the `target` delay is dequeued. In the drop state, packets are dropped from the head of the queue to quickly signal congestion to the sender, the drop rate increases with time till the measured delay is reduced.

A major reason for CoDel's effectiveness is its simplicity, it is meant to work well with default parameters and is very efficient. As CoDel adapts well to multiple queue systems, the CoDel RFC [8] recommends using a multiple-queue approach like FQ-CoDel or CAKE, instead of CoDel alone.

2) Flow Queue CoDel: Flow Queue CoDel (FQ-CoDel) [10] is the default queueing discipline in systemd based Linux distributions and is included in many router operating systems [11].

FQ-CoDel implements byte-based fairness between flows by maintaining a separate CoDel instance for each flow, organized using a hash table. Each entry in the hash table contains an independent CoDel instance and the state required for deficit round-robin (DRR) scheduling. In addition to the hash table, FQ-CoDel groups active flows in two lists called `new_flows` and `old_flows`.

An FQ-CoDel instance takes as parameters the size of the hash-table `flows_cnt`, the DRR `quantum`, and the maximum queue limit `drop_overlimit`, as well as the base CoDel parameters to initialize each flow queue.

At dequeue time, the CoDel queues are polled in round-robin, with `new_flows` given strict priority. This DRR scheduling ensures that all flows can transmit a `quantum` of bytes before any flow sends more, approximating byte-based fairness efficiently. The use of two active flow lists helps new flows start quickly and improves fairness for sparse flows.

3) Common Applications Kept Enhanced: Common Applications Kept Enhanced (CAKE) was developed as a successor to FQ-CoDel, building on its design while introducing several enhancements and new features. One of the key improvements is the introduction of 8-way set associativity, which effectively eliminates hash collisions even with a very large number of flows. Another important feature is enhanced DiffServ support, allowing the configuration of multiple priority tiers, and scheduling flows based on their DiffServ markings. In this work, we leverage this feature to implement age-based scheduling.

4) Proportional Integral controller Enhanced: Proportional Integral controller Enhanced (PIE) is another AQM solution

designed to address the bufferbloat problem [12], [13]. Its goal is to achieve results and ease of deployment similar to CoDel, while retaining the ease of implementation and scalability found in RED.

Like RED, PIE separates its logic into two parts: one algorithm computes the statistics needed to determine the drop probability drop_prob, and another decides when to actually drop packets. However, PIE adopts packet sojourn time as its congestion metric, following CoDel, rather than the queue length used by RED. PIE performs its packet dropping decisions during the enqueue operation, which is the same approach RED uses. The drop probability drop_prob is computed regularly, using both the distance between queue delay and delay target, and the first derivative of the delay, linearly combined using two parameters. Pie tends to react more quickly than CoDel, but it has more configuration parameters.

As with CoDel and FQ-CoDel, PIE is not typically deployed on its own, but it can be combined with a flow isolation mechanism, which we refer to as FQ-PIE.

III. Age-based Scheduling

The behaviour of TCP flows over an interface configured to schedule packets according to a FIFO discipline is typically modelled as an M/G/1/PS queue in queuing theory. This stands for Markovian arrivals, General service time, 1 server, Processor Sharing. When information about job sizes is available, either exactly or as a probability distribution, this formulation allows deriving theoretical results that can be used to optimize scheduling decisions. Here, we briefly review key age-based scheduling disciplines, their theoretical properties, and practical considerations for their implementation.

1) Shortest Remaining Processing Time: Shortest Remaining Processing Time (SRPT) is a size-based optimal scheduling discipline for minimizing average job completion time [14]. SRPT is a preemptive, work-conserving discipline. It always serves the job with the least remaining work, based on its total size and the amount of service it has already received.

Despite its optimality, SRPT is rarely used in practice because it requires knowing the exact size of each job in advance, without this knowledge, we can assume to know the statistical distribution of job sizes, and use a discipline called Shortest Expected Remaining Processing Time (SERPT). SERPT prioritizes jobs based on the expected remaining work, given the service they have already received. This is equivalent to prioritizing jobs with the highest hazard rate.

2) Least Attained Service: For long-tailed job size distributions, which have a monotonically decreasing hazard rate, an age-based scheduling approach is equivalent to SERPT. The most straightforward implementation of this idea is the Least Attained Service (LAS) discipline [4]. LAS works by always prioritizing jobs that have received the least amount of service so far, using Processor Sharing among jobs with equal attained service.

The main drawback of LAS is its implementation complexity. It requires maintaining a separate FIFO queue for every flow, as well as a priority queue to schedule flows according to their age or attained service.

3) Multi-Level and 2-Level Processor Sharing: A common practical approximation of LAS is Multi-Level Processor Sharing (MLPS). MLPS uses a fixed number of processor sharing queues, each corresponding to a range of attained service (or "age"). These queues are served in strict priority order: jobs start in the highest-priority queue and move to lower-priority queues as they accumulate more service. This approach reduces complexity while still capturing most of the benefits of LAS.

Two Level Processor Sharing (2LPS or PS+PS) is a special case of MLPS that uses only two levels of priority. It is parametrized using a service threshold a. Flows that have sent less than or equal to a are served with high-priority, while flows that have sent more than a bytes are only served when there are no other flows left. Adding more priority levels can make the approximation closer to the ideal LAS discipline, but with diminishing returns and increased implementation complexity. With just two levels and a well-chosen threshold a, most of the performance benefits of LAS can be achieved at minimal cost [5].

A. Threshold selection

While 2LPS is easier to implement than LAS, it does require choosing a good threshold to be effective. The optimal threshold a primarily depends on the flow size distribution, which is usually somewhat stable over time, and to a lesser extent on the system load. Previous works have shown that it is possible to compute the optimal threshold by analysing traffic statistics, and that this approach is both feasible and computationally efficient [5] with available tools [15].

For deployment in a real system, it would be best to periodically recompute this threshold to adapt to changing traffic patterns. This could be accomplished by running a background process that collects flow size statistics and updates the threshold as needed, and it was shown to be possible in realistic conditions [5]. In our experiments for simplicity we use a static threshold computed only once from the distribution used to generate the traffic and the load factor specific to each experiment.

Let us note that a 2LPS system can not provide worse performance than the equivalent one with only one queue. This is because the principle of 2LPS is to give priority to short flows, compared to long ones. If the threshold is not correctly set, the proportion between the number of high-priority and low-priority flows would not be optimal but still, it would provide priority to some short flows against long ones. In the worst case the threshold is too big or too low, one queue will be always empty, then the system degenerates to the equivalent

with only one queue. This said, we will leave the sensitivity analysis on the choice of the threshold to future works.

IV. Experimental Setup

We conducted our experiments on two different setups with network cards supporting 1 Gb/s and 10 Gb/s, using traffic flow sizes randomly generated with a bounded Pareto distribution. In both cases, we used two bare-metal nodes, a client and a server, directly connected with a dedicated Ethernet cable and controlled via a separate network card. It is fundamental to note that our goal is to study the improvement provided by age-based scheduling to AQM policies, that are enforced on the outgoing traffic in the Linux operating system. These policies are applied to any bottleneck condition that takes place on any link from the client to the server, so we can safely run our tests in the simplest and most controllable setup made of one link only.

Table I reports the base parameters of the two setups together with the mean RTT measured with the `ping` utility on 100 samples, and the throughput measured using `iperf` over a single 60 seconds TCP stream.

Setup	Controller	RTT	Throughput
1	Intel I219-LM, 1000baseT	0.710 ms	864 Mb/s
2	Intel X540-AT2, 10000baseT	0.270 ms	9.34 Gb/s

TABLE I
Testbed parameters.

Following recommendations from the bufferbloat team [17], many hardware offloads have been disabled on the test interfaces, as to not introduce extra latency during the tests. These include RX and TX checksumming, scatter-gather, TCP segmentation and generic fragmentation, generic receive offload, and VLAN acceleration.

A. Traffic generation

To generate traffic, we use a closed-loop test configuration. It consists in N simulated actors, each one independently and concurrently alternate a thinking phase with an action phase. During the thinking phase, an actor sleeps for an exponentially distributed random amount of time, to then enter the action phase. During the action phase, an actor uploads a random amount of data using TCP, and returns to the thinking phase on flow completion.

To simulate a long-tailed traffic distribution, the amount of data to upload is sampled from a bounded Pareto random variable. Together with the exponential thinking time, this results in traffic with a Poisson arrival process and Pareto job size. The stability condition is always guaranteed in a closed-loop test, and utilization can be tuned by changing the amount of actors or the parameters of the arrival process or job distribution. The parameters used for the traffic generation are shown in Table II.

Setup	Arrival Process		Flow Size Distr.		Load	a
	N	Thinking Time	α	Min-Max Size	Factor	(MB)
1	80	1 s	1.2	300 KiB-1 GiB	0.88	6.33
2	150	0.35 s	1.1	373 KiB-62 GiB	0.92	19.5

TABLE II
Traffic generation parameters.

For the first setup, which uses a 1 Gb/s link, we configured the test with 80 actors. Each actor has a mean thinking time of 1 second. The job sizes are sampled from a bounded Pareto distribution with shape parameter $\alpha = 1.2$, and bounded between 300 KiB and 1 GiB. This configuration results in a load factor (average B/s effectively generated during the experiment) of 0.88 with respect to the nominal link rate of the interface.

For the second setup, with a 10 Gb/s link, we used 150 actors and set the mean thinking time to 0.35 seconds. The job sizes are again sampled from a bounded Pareto distribution, with shape parameter $\alpha = 1.1$, and bounded between 373 KiB and 62 GiB. We decreased the shape parameter so to have slightly less short connections, that end in a matter of milliseconds on a 10 Gb/s link, and are subject to a higher measure error. This setup achieves a load factor of 0.92. As the two setups have different traffic distribution, we set the threshold a for age-based scheduling to 6.33 MB for the 1 Gb/s setup and 19.5 MB for the 10 Gb/s following the method described in subsection III-A.

Before settling on these parameters, we experimented with a range of different configurations. When the load factor was too low, the choice of scheduling algorithm had little impact on performance. Conversely, when the load factor was too high, performance degraded rapidly for all algorithms due to packet loss that triggered TCP congestion control. Reducing the minimum job size increased the number of flows per unit time, which led to higher CPU usage. Increasing the maximum job size raised the run-to-run variance, since very large flows became increasingly rare. Additionally, due to implementation details, the RAM required by the configuration parser scales with the number of actors N, making it impractical to test much larger values.

B. Antler

To run the tests we used Antler [18], a tool for network and congestion control testing written in the Go language, which allows the definition of tests using configuration files written in the CUE language. The test configuration is kept on a coordinator machine, which parses it and connects to the test nodes via SSH to start the processes necessary to run the test. Test data collected by the remote machines is sent back over the SSH connection to be post-processed and stored by the coordinator machine.

An Antler test consists of a tree-like combination of configurable pre-defined runners. A runner can, for example, execute shell commands on a node, generate TCP or UDP

traffic, or execute other runners sequentially or in parallel. We made several extensions to Antler, available in the public repositories.

C. Queue Configuration

While CoDel claims to be parameterless for common Internet deployments, our setup differs from the typical multi-hop Internet Path, so we reduced both `interval` (10 ms) and `target` (0.5 ms). While a lower `interval` might have been more appropriate for our sub 1 ms RTT link, we wanted to avoid potential issues related to low interrupt rates [17], and we expect an `interval` of 10 ms to have excellent performance for connections in the 1-30 ms RTT range [3]. CAKE only supports specifying a `rtt` parameter: it sets the CoDel `target` to 5% of the configured value, and the CoDel `interval` equal to it. We configured it to use a value of 10 ms, to match the other CoDel based algorithms. For PIE, the configurable values include a delay `target`, similar to that of CoDel, and an update interval, both defaulting to 15 ms. We set the `target` to 0.5 ms to match CoDel, and the update interval to 3 ms.

D. Test process

The queueing disciplines we are testing are 6: FIFO, CoDel, FQ-CoDel, CAKE, PIE and FQ-PIE, and for each we test both the standard one and the age-based variant. Each test in a set of twelve uses the same seed to generate traffic, and runs for 6 hours (1 Gb/s setup) or 4 hours (10 Gb/s setup). As the 10 Gb/s setup generates a higher number of flows per second, we achieve very small confidence intervals in a shorter time. The first hour of data for the first setup, and half hour of data for the second setup, is discarded as warm-up period. The test duration was chosen based on when the mean flow completion time stabilized, and the startup period was set by observing when the relative performance of the different algorithms stopped changing significantly. Each test is repeated three times, for a total of 18 hours (1 Gb/s setup) and 12 hours (10 Gb/s) per each of the 12 disciplines.

V. Results

A. Mean Flow Completion Time

The first interesting metric to consider is mean Flow Completion Time, which is what an age-based scheduling algorithm aims to optimize. It is obtained by taking the mean of the completion time over each flow. We highlight the difference between each age-based algorithm and its base version by computing the relative reduction in mean FCT between the two, shown in the result tables as Gain, defined as $T_{size}/T_{base} - 1$.

The results for the 1 Gb/s setup, listed in Table III and plotted for clarity in Figure 1, show that the mean flow completion time for age-based variants are always smaller when compared do the non-age-based versions. The biggest improvements

Algorithm	Age-based version		Base version		Gain
	Mean FCT	95% CI	Mean FCT	95% CI	
FIFO	123	1.46	154.3	0.748	-20 %
CAKE	127.2	1.37	142.1	0.906	-10 %
FQ-CoDel	119.4	1.5	137.6	0.966	-13 %
CoDel	118.1	1.47	122.2	0.971	-3.3 %
FQ-PIE	126	1.14	138.7	0.824	-9.1 %
PIE	119.4	1.27	130.9	0.813	-8.8 %

TABLE III
MEAN FLOW COMPLETION TIME FOR THE 1 GB/S SETUP (MILLISECONDS) AND RELATIVE GAIN.

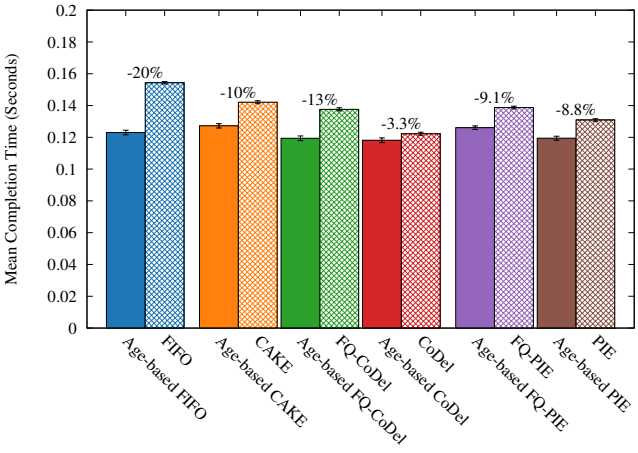

Fig. 1. Mean Flow Completion Time for the 1 Gb/s setup (milliseconds) and relative gain. Pairs of data points show data for the same AQM solution, with and without age-based scheduling. Lower is better. Error shown as 95% confidence intervals on the mean.

(20%) can be seen for FIFO, as it has the worst base performance, followed by FQ-CoDel (13%), CAKE (10%), PIE and FQ-PIE (around 9%), and finally CoDel with 3.3% reduction in FCT. All confidence intervals are separated, indicating stability in the results. It is important to note that the worst age-based algorithm (FIFO) has better performance of all the non-age-based variants, with the exception of CoDel, that performs very close to age-based FIFO. The best performing one is age-based CoDel.

In this setup, the flow-queue algorithms perform worse than their single-queue versions, CAKE is slightly worse than FQ-CoDel in both the size and non-size variants, despite being a very similar algorithm. All the age-based algorithm tend to have similar performance, with the age-based versions of CoDel, FQ-CoDel, FIFO, and PIE, having mean FCT within 4% of each other. Age-based AQM solutions still beat age-based FIFO, but age-based CAKE and FQ-PIE are slightly worse than no age-based FIFO in this setup.

For the 10 Gb/s setup results are listed in Table IV and visible in Figure 2, the situation is slightly different. Most noticeably, CAKE behaves erratically, with FCT that are 5 to 15 times larger than the other algorithms. This behaviour is repeatable, and is explained by the high CPU usage for soft IRQ observed using CAKE in the 10 Gb/s setup, that we comment later on.

Algorithm	Age-based version		Base version		Gain
	Mean FCT	95% CI	Mean FCT	95% CI	
FIFO	40.59	0.398	44.39	0.317	-8.6 %
CAKE	184.8	10.3	659.9	16.4	-72 %
FQ-CoDel	37.83	0.464	39.17	0.417	-3.4 %
CoDel	38.66	0.441	41.98	0.366	-7.9 %
FQ-PIE	37.47	0.468	40.81	0.414	-8.2 %
PIE	37.58	0.447	41.3	0.395	-9 %

TABLE IV
MEAN FLOW COMPLETION TIME FOR THE 10 GB/S SETUP (MILLISECONDS) AND RELATIVE GAIN.

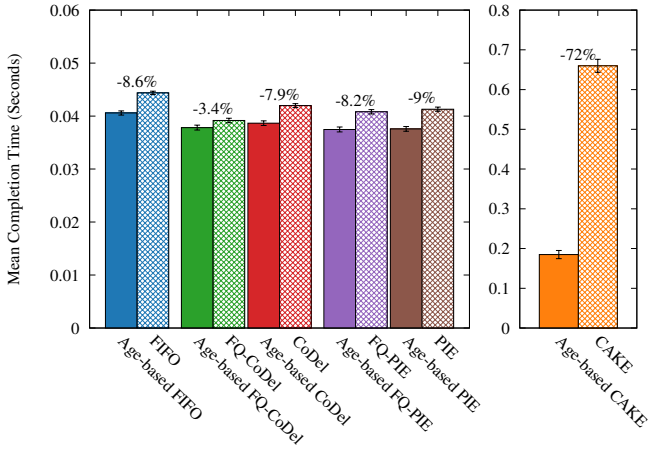

Fig. 2. Mean Flow Completion Time for the 10 Gb/s setup, organized by algorithm. The same color is used for the same AQM solution, with and without age-based scheduling. Error shown as 95% confidence intervals on the mean.

While the behaviour of CAKE is very much an outlier, the other disciplines are stable and interesting to analyse.

Every age-based discipline is still better than its base variant, but the relative gain is distributed differently compared to the 1 Gb/s setup. Overall, the best performers are age-based FQ-PIE and age-based PIE, closely followed by age-based FQ-CoDel. The base performance of FQ-CoDel is also very good, as it is very close to the performance of the other age-based AQM algorithms and only 3% behind that of age-based FQ-CoDel. PIE and FQ-PIE have very similar performance, both with about 8% difference between the age-based and base versions. The worst performing age-based algorithm is FIFO, in both its variants. While the 1 Gb/s favored non-flow-queueing AQM algorithms, CoDel and PIE, in the 10 Gb/s setup FQ-CoDel and FQ-PIE have better performance.

Overall our results show that in both setups, whatever is the chosen AQM discipline, the age-based version always improves the flow completion time. Age-based scheduling can be successfully coupled with any AQM discipline to solve Bufferbloat and further improve the performance.

B. Flow Completion Time Percentiles

High percentiles of flow completion times provide deeper insights into the performance of the tested algorithms, as shown in Table V for the 1 Gb/s setup and Table VI for the 10 Gb/s setup. In all cases, the age-based versions of the algorithms achieve lower P50, P75, and P95 values compared to their base versions. This is expected, as 97.4% of flows in the 1 Gb/s setup and 99.1% in the 10 Gb/s setup are smaller than the threshold, meaning these flows benefit from priority. However, P99 behaves a bit differently: often performance is much closer between versions, and notably age-based FIFO performs worse than FIFO. In the 1 Gb/s setup, the P99 of age-based CoDel and age-based FQ-CoDel significantly outperform the respective base algorithms.

Algorithm	P50	P75	P95	P99
Age-b. FIFO	33.04	51.86	145.9	1466
FIFO	85.16	149.2	389.6	1147
Age-b. CAKE	42.24	69.32	218.3	1138
CAKE	59.19	105.2	364.7	1334
Age-b. FQ-CoDel	35.02	56.37	176.2	951.2
FQ-CoDel	52.18	94.06	351.8	1339
Age-b. CoDel	34.07	53.77	184.4	974.6
CoDel	38.97	76.13	303.7	1202
Age-b. FQ-PIE	40.48	64.61	201.8	1326
FQ-PIE	54.22	96.3	355.6	1327
Age-b. PIE	32.97	51.94	146.6	1216
PIE	44.28	89.34	366.9	1231

TABLE V
FLOW COMPLETION TIME PERCENTILES (MILLISECONDS) FOR THE 1 GB/S SETUP.

Algorithm	P50	P75	P95	P99
Age-b. FIFO	11.89	23.49	79.29	545.7
FIFO	18.25	34.06	106.2	529.7
Age-b. CAKE	4.423	7.82	32.59	968.4
CAKE	162.6	308.1	1351	5948
Age-b. FQ-CoDel	10.67	21.31	75.4	507.6
FQ-CoDel	11.35	22.55	87.87	527.4
Age-b. CoDel	11.47	22.98	79.2	522.4
CoDel	14.73	29.44	101.5	509.5
Age-b. FQ-PIE	9.523	17.91	71.15	551.4
FQ-PIE	12.17	22.24	91.71	557.8
Age-b. PIE	9.717	17.46	68.05	553.7
PIE	14.38	25.41	94.61	528

TABLE VI
FLOW COMPLETION TIME PERCENTILES (MILLISECONDS) FOR THE 10 GB/S SETUP.

C. Flow Completion Time versus Flow Size

By looking at how the flow completion time varies depending on the flow size, we can better understand where the performance differences of subsection V-A and V-B come from.

The 1 Gb/s setup is shown in Figure 3. The most noticeable feature of the plot, which also serves as good validation for our setup, is the sharp angle at the 6 MB mark for curves corresponding to two-level algorithms. This is where the size threshold is located, and it separates flows that only receive high priority service from flows that eventually end up in the low priority level of age-based queues.

Overall, the two-level disciplines always have better performance than all the normal disciplines for flows that are smaller

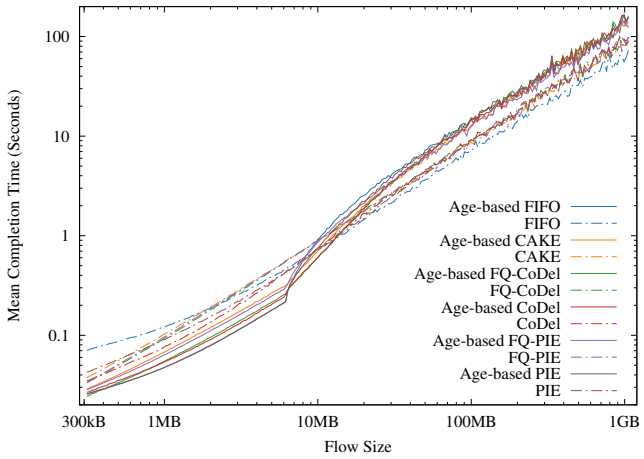

Fig. 3. Mean Flow Completion Time versus Flow Size for the 1 Gb/s setup, organized by algorithm. Lower is better.

than the threshold. Soon after the threshold, the performance of two-level disciplines degrades, to finally become always worse than that of normal disciplines around the 20 MB mark.

Starting from age-based CoDel and age-based FQ-CoDel, the top performers of the 1 Gb/s setup, we can see that their behaviour is indistinguishable for flows larger than the threshold, where they have the best performance over the two-level algorithms. For flows smaller than the threshold, FQ-CoDel holds the advantage on flows smaller than 800 KB, while CoDel is better in the 800 KB to 6 MB range. From this we can deduce that the effect of flow-queueing is most beneficial to the very small flows.

Two-level PIE behaves identically to two-level FIFO before the threshold, but beats it in the 7 MB to 30 MB range. These two algorithms together have the best overall performance for flows smaller than the threshold, and perform worse than the other two-level algorithms for flows larger than the threshold.

Overall, while two-level FIFO, CoDel, FQ-CoDel and PIE all have comparable mean performance, we can see that the two CoDel based solutions are more balanced, and two-level FIFO and PIE favour smaller flows more strongly. Two-level FQ-PIE has consistently poor performance when compared to other two-level algorithms, except for the biggest flows.

Similarly, two-level CAKE is consistently worse than two-level FQ-CoDel by a constant factor for flows smaller than the threshold, to eventually become equivalent after the 20 MB mark. When comparing normal CAKE to normal FQ-CoDel, a similar behaviour emerges. Normal CAKE is worse than normal FQ-CoDel by a constant factor up to around the 1 MB mark, the gap then shrinks, until they have equal performance after the 5 MB mark. A potential explanation to this performance gap in otherwise very similar algorithms, could be found in the fact that CAKE is slightly more complex, and thus slightly slower at processing packets, by a constant amount. As the RTT of our setup is very small, this extra delay is not insignificant, and ends up affecting the slow-start phase of the TCP connections.

Age-based FIFO has the best performance for flows below the threshold, and the worst performance for flows above it. Inversely, normal FIFO is by far the worst discipline for flows smaller than 1 MB, to then become the best for flows larger than 20 MB. These two algorithms are the most extreme, swapping places around the threshold.

Due to space reasons and to void repetitions, we don't report the same graph for the 10 Gb/s setup, that show a similar behavior.

D. Fairness Analysis: the Lorenz Curve Gap

As age-based scheduling treats flows differently based on the amount of data they carry, it is interesting to measure the effect on fairness in completion time. We use the Lorenz Curve Gap, that is the maximum euclidean distance between the Lorenz Curve and the curve of maximum fairness, rescaled by its maximum value [19]. The Lorenz curve reports the normalized mean throughput on the Y axes, and the percentile of flows in the X axes, rescaled between 0 and 1.

A Gap value of 0 would indicate that the Lorenz curve is a straight line, so that the completion time is distributed evenly among the flows. A value close to 1 would indicate that a very small minority of flows received the vast majority of the capacity. Figure 4 shows, for example, the Lorenz Curve for CoDel in the 1 Gb/s setup, highlighting the gap.

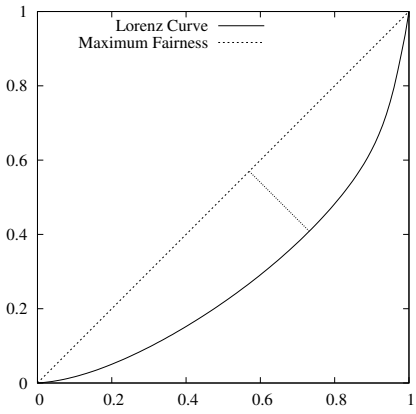

Fig. 4. Lorenz Curve for CoDel in the 1 Gb/s setup.

Figure 5 summarizes the values of the Lorenz curve Gap for the 1 Gb/s setup, the straight lines report also the average for all age-based and non age-based variants. In all six cases the age-based variants show better fairness than their base versions. To justify this behavior it is important to note that on a short connection over a fast link, the TCP connection set-up and the slow-start phase has a strong impact on the average throughput, while it has a lower impact on a longer connection. The average throughput measure then has a bias that favors longer connections. Age-based policies tend to

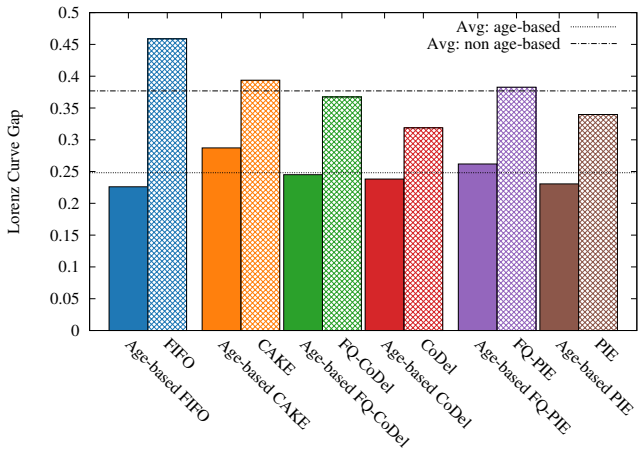

Fig. 5. Lorenz Curve Gap for the 1 Gb/s setup, computed on the mean throughput, organized by algorithm. Values are between 0 and $1 - \frac{1}{n}$, lower is better.

balance this effect, because they provide more capacity to short flows overall improving fairness across all flows.

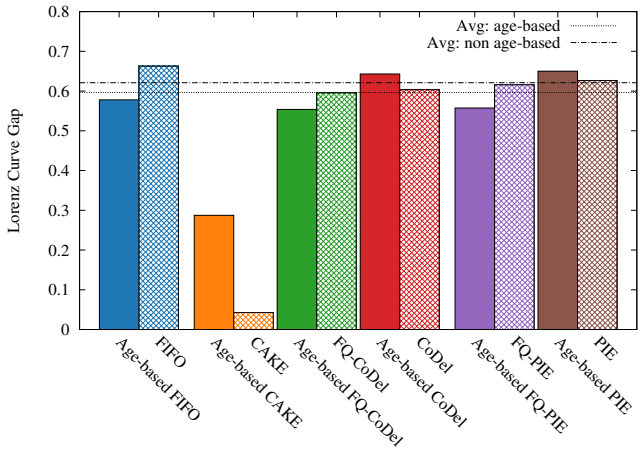

Fig. 6. Lorenz Curve Gap for the 10 Gb/s setup, computed on the mean throughput, organized by algorithm. Values are between 0 and $1 - \frac{1}{n}$, lower is better.

The situation is slightly different in the 10 Gb/s setup, shown in Figure 6. Other than the two CAKE algorithms behaving like outliers (and so they were not considered in the average computation), CoDel and PIE score better than age-based variants, while FIFO and FQ-CoDel score worse than the age-based variants. In average, the fairness of age-base variants is still better, but the difference is smaller. This is because in the 10 Gb/s setup we changed the exponent of the flow size distribution, this also changed the threshold computation so the flows remain in high priority till 19.5 MB transmitted. This smooths the effect of the three-way-handshake and of slow start on the average throughput, reducing the balancing effect we see at 1 Gb/s.

Algorithm	User	System	IO Wait	IRQ	Soft IRQ
Age-b. FIFO	0.126	0.139	3.09e-05	0	0.765
FIFO	0.124	0.131	3.09e-05	0	0.697
Age-b. CAKE	0.13	0.149	2.48e-05	0	0.844
CAKE	0.131	0.148	2.73e-05	0	0.835
Age-b. FQ-CoDel	0.129	0.142	2.39e-05	0	0.782
FQ-CoDel	0.132	0.14	2.75e-05	0	0.744
Age-b. CoDel	0.13	0.141	0.000103	0	0.783
CoDel	0.127	0.135	2.84e-05	0	0.744
Age-b. FQ-PIE	0.132	0.14	2.5e-05	0	0.761
FQ-PIE	0.133	0.14	3.23e-05	0	0.695
Age-b. PIE	0.128	0.142	2.85e-05	0	0.754
PIE	0.129	0.139	2.87e-05	0	0.706

TABLE VII
MEAN CPU USAGE OF THE CLIENT NODE ON THE 1 GB/S SETUP, SHOWN AS FRACTIONS OF CORE USED (OUT OF 4).

Algorithm	User	System	IO Wait	IRQ	Soft IRQ
Age-b. FIFO	0.256	0.38	0.00127	0	1.4
FIFO	0.249	0.365	0.00134	0	1.28
Age-b. CAKE	0.172	0.317	0.000522	0	2.9
CAKE	0.0877	0.121	0.00122	0	2.99
Age-b. FQ-CoDel	0.254	0.369	0.00125	0	1.41
FQ-CoDel	0.256	0.367	0.00132	0	1.3
Age-b. CoDel	0.258	0.38	0.00124	0	1.4
CoDel	0.253	0.364	0.0013	0	1.3
Age-b. FQ-PIE	0.251	0.376	0.00142	0	1.43
FQ-PIE	0.247	0.368	0.00146	0	1.29
Age-b. PIE	0.248	0.37	0.0013	0	1.4
PIE	0.24	0.353	0.00136	0	1.29

TABLE VIII
MEAN CPU USAGE OF THE CLIENT NODE ON THE 10 GB/S SETUP, SHOWN AS FRACTIONS OF CORE USED (OUT OF 4).

E. CPU Usage

Finally, we expect age-based scheduling to introduce some computational overhead, due to packet marking done by `nftables`. Table VII and Table VIII report the measured CPU load during the experiment and show that the increase in CPU load for age-based algorithms is barely noticeable. Also, it is important to note that a higher throughput implies a higher CPU usage, so the added CPU load due to computational overhead is negligible thanks to the efficiency of the `netfilter` and `nftables` subsystem. We also note that CAKE in the 10 Gb/s setup as a load that is at least twice the load of the other AQM policies.

VI. State of The Art

The literature on AQM is extremely large, in this section we add a brief review on papers that analysed the performance of recent AQM policies based on Linux, only to show the novelty of our approach.

Several works tested CoDel and PIE, as the works from Khademi, [20], Ramakhrisnan [21], Imputato [22], and Kuhn [23]. Other authors tried to optimize the configuration parameters of AQM algorithms [23], [24] and several studied the interplay between AQM and TCP such as Carlucci, [25], Casoni [26] and Mulla [27]. These papers often use a setup similar to the one we used and exploit the same open source

codebase, however they don't introduce age-based scheduling. Age-based scheduling is itself the subject of many works, some using Linux, such as Geng et al. [28] that provided a generalization of age-based scheduling policies and applied them to the Linux kernel, before AQM policies were integrated in the kernel. Also, the already mentioned papers by Marin et al. [5], [15] apply age-based scheduling to Linux, but they use only the FIFO policy.

To the best of our knowledge, this is the first paper that applies an age-based scheduler to AQM policies, with specific focus on the most popular and widespread policies of the Linux kernel

VII. Conclusions and Future Works

Our work has successfully demonstrated the effectiveness of combining AQM techniques with age-based scheduling disciplines in order to optimize network performance while reducing the impact of bufferbloat. We developed custom hybrid queueing solutions, implemented using standard Linux utilities like traffic control (`tc`) and nftables (`nft`). Specifically, two-level processor sharing was implemented using `nft` to flag packets based on bytes sent in the flow, and multi-band queues were assembled with `tc-prio` to enforce priority. To evaluate these configurations under realistic workload conditions, we extended the Antler network testing tool to generate traffic with Pareto-distributed flows. Extensive experiments were conducted on dedicated 1 Gb/s and 10 Gb/s links. Our results show that applying an age-based scheduling policy on top of state-of-the-art AQM strategies significantly reduces mean flow completion time, from 3% to 20% depending on the AQM policy and setup. This approach not only scientifically validates the benefits of combining AQM and age-based scheduling but also offers a solution easily implementable on any recent Linux kernel without requiring kernel modifications. This ensures both scientific reproducibility and immediate applicability in real-world scenarios. For future work, the sensitivity analysis on the choice of the threshold for two-level scheduling remains an area of interest, as well as further optimization using more than 2 queues.

References

[1] J. Gettys and K. M. Nichols, "Bufferbloat: Dark buffers in the internet: Networks without effective aqm may again be vulnerable to congestion collapse." *Queue*, vol. 9, no. 11, pp. 40–54, Nov. 2011. [Online]. Available: https://doi.org/10.1145/2063166.2071893

[2] ——, "Bufferbloat: dark buffers in the internet," *Commun. ACM*, vol. 55, no. 1, pp. 57–65, Jan. 2012. [Online]. Available: https://doi.org/10.1145/2063176.2063196

[3] K. M. Nichols and V. Jacobson, "Controlling queue delay," *Commun. ACM*, vol. 55, no. 7, pp. 42–50, 2012. [Online]. Available: https://doi.org/10.1145/2209249.2209264

[4] I. Rai, E. Biersack, and G. Urvoy-keller, "Size-based scheduling to improve the performance of short tcp flows," *IEEE Network*, vol. 19, no. 1, pp. 12–17, 2005.

[5] A. Marin, S. Rossi, and C. Zen, "Size-based scheduling for tcp flows: Implementation and performance evaluation," *Computer Networks*, vol. 183, p. 107574, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128620312172

[6] L. Zhang, D. C. Partridge, S. Shenker, J. T. Wroclawski, D. K. K. Ramakrishnan, L. Peterson, D. D. D. Clark, G. Minshall, J. Crowcroft, R. T. Braden, D. S. E. Deering, S. Floyd, D. B. S. Davie, V. Jacobson, and D. D. Estrin, "Recommendations on Queue Management and Congestion Avoidance in the Internet," RFC 2309, Apr. 1998. [Online]. Available: https://www.rfc-editor.org/info/rfc2309

[7] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, 1993.

[8] K. M. Nichols, V. Jacobson, A. McGregor, and J. Iyengar, "Controlled Delay Active Queue Management," RFC 8289, Jan. 2018. [Online]. Available: https://www.rfc-editor.org/info/rfc8289

[9] L. Kleinrock and R. Gail, "An invariant property of computer network power," *IEEE International Conference on Communications*, vol. 3, pp. 63.1.1–63.1.5, 1981. [Online]. Available: https://www.lk.cs.ucla.edu/data/files/Gail/power.pdf

[10] T. Høiland-Jørgensen, P. McKenney, dave.taht@gmail.com, J. Gettys, and E. Dumazet, "The Flow Queue CoDel Packet Scheduler and Active Queue Management Algorithm," RFC 8290, Jan. 2018. [Online]. Available: https://www.rfc-editor.org/info/rfc8290

[11] CoDel Overview. [Online]. Available: https://www.bufferbloat.net/projects/codel/wiki/

[12] 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)*, 2013, pp. 148–155.

[13] R. Pan, P. Natarajan, F. Baker, and G. White, "Proportional Integral Controller Enhanced (PIE): A Lightweight Control Scheme to Address the Bufferbloat Problem," RFC 8033, Feb. 2017. [Online]. Available: https://www.rfc-editor.org/info/rfc8033

[14] L. Schrage, "A proof of the optimality of the shortest remaining processing time discipline," *Operations Research*, vol. 16, no. 3, pp. 687–690, 1968. [Online]. Available: https://www.jstor.org/stable/168596

[15] A. Marin, S. Rossi, and C. Zen, "A matlab toolkit for the analysis of two-level processor sharing queues," in *Quantitative Evaluation of Systems*, M. Gribaudo, D. N. Jansen, and A. Remke, Eds. Cham: Springer International Publishing, 2020, pp. 144–147.

[16] A. Horváth and M. Telek, "Phfit: A general phase-type fitting tool," in *Computer Performance Evaluation: Modelling Techniques and Tools*, T. Field, P. G. Harrison, J. T. Bradley, and U. Harder, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 82–91.

[17] Best Practices for Benchmarking CoDel and FQ CoDel (and almost any other network subsystem!). [Online]. Available: https://www.bufferbloat.net/projects/codel/wiki/Best_practices_for_benchmarking_Codel_and_FQ_Codel/

[18] P. Heist. Antler. [Online]. Available: https://github.com/heistp/antler

[19] J.-Y. Le Boudec, *Performance Evaluation of Computer and Communication Systems*. EPFL Press, Lausanne, Switzerland, 2010.

[20] N. Khademi, D. Ros, and M. Welzl, "The new aqm kids on the block: An experimental evaluation of codel and pie," in *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2014.

[21] G. Ramakrishnan, M. Bhasi, V. Saicharan, L. Monis, S. D. Patil, and M. P. Tahiliani, "FQ-PIE Queue Discipline in the Linux Kernel: Design, Implementation and Challenges," in *IEEE LCN Symposium on Emerging Topics in Networking*, 2019.

[22] P. Imputato, S. Avallone, M. P. Tahiliani, and G. Ramakrishnan, "Revisiting design choices in queue disciplines: The PIE case," *Computer Networks*, vol. 171, 2020.

[23] N. Kuhn, D. Ros, A. B. Bagayoko, C. Kulatunga, G. Fairhurst, and N. Khademi, "Operating ranges, tunability and performance of CoDel and PIE," *Computer Communications*, vol. 103, 2017.

[24] M. Dery, O. Krupnik, and I. Keslassy, "Queuepilot: Reviving small buffers with a learned aqm policy," in *IEEE Conference on Computer Communications (INFOCOM)*, 2023.

[25] G. Carlucci, L. De Cicco, and S. Mascolo, "Controlling queuing delays for real-time communication: the interplay of e2e and aqm algorithms," *SIGCOMM Computer Communications Review*, vol. 46, no. 3, Jul. 2018.

[26] M. Casoni, C. A. Grazia, M. Klapez, and N. Patriciello, "How to avoid tcp congestion without dropping packets: An effective aqm called pink," *Computer Communications*, vol. 103, 2017.

[27] Y. Mulla and I. Keslassy, "Per-cca queueing," in *2024 20th International Conference on Network and Service Management (CNSM)*, 2024, pp. 1–7.

[28] H. Feng, V. Misra, and D. Rubenstein, "PBS: a unified priority-based scheduler," *SIGMETRICS Performance Evalation Review*, vol. 35, no. 1, Jun. 2007.