dAQM: Derivative-based Active Queue Management

Saad Saleh[*][†], Sunny Shu[*], Boris Koldehofe[†][‡]
[*]Bernoulli Institute, University of Groningen, Netherlands
[†]CogniGron (Groningen Cognitive Systems and Materials Center), University of Groningen, Netherlands
[‡]Department of Computer Science and Automation, Technische Universität Ilmenau, Germany
s.saleh@rug.nl, s.shu@student.rug.nl, boris.koldehofe@tu-ilmenau.de

***Abstract*—The network systems build heavily on Active Queue Management (AQM) algorithms for maintaining an optimal queue size and avoiding issues like Bufferbloat. Despite promising performance, the current AQM algorithms face a major challenge of estimating the accurate congestion due to bursty network traffic. The major reason is the use of baseline traffic features like average delay and sojourn time. In this paper, we propose a novel AQM algorithm, called dAQM, which uses advanced traffic features like three higher-order derivatives of sojourn time and buffer size for computing the packet drop probability based on the network congestion. The higher-order derivatives of sojourn time and buffer size provide the rate of increase of packet accumulation in the queues. We show the programmability and performance analysis of dAQM over five traffic distribution models (Pareto, Poisson, etc.) for five widely used traffic classes. The analysis over ns-3 simulations showed that dAQM provides at least 28% and 62% improvement in delay and queue length, respectively, for an increase in traffic load as compared to the traditional AQM algorithms. dAQM also reduces the flow completion time for long flows, i.e., FTP traffic, by at least 39.7% as compared to the prior AQM algorithms.**

***Index Terms*—Active Queue Management; Congestion control;**

I. Introduction

The network systems require an optimal queue size inside switches, routers, and data centers to provide satisfactory Quality of Service (QoS) to end-users. A large queue size increases the end-to-end latency resulting in issues like Bufferbloat [1]. On the contrary, a small queue size increases the packet losses resulting in decreased throughput. In order to maintain an optimal queue size, Active Queue Management (AQM) algorithms like Random Early Detection (RED) [2], Controlled Delay (CoDel) [3] etc., are used for selectively dropping the packets based on congestion. Despite the promising performance, the state-of-the-art AQM algorithms face a major challenge of accurate congestion estimation due to bursty network conditions. The major reason is the use of baseline queue statistics for congestion estimation e.g., average delay and sojourn time for RED and CoDel, respectively. As a result, packets in the queues with large delays are dropped even if the trends show that the packet arrival rate is decreasing and the queue can manage the packets. It deteriorates the QoS due to excessive packet losses and large delays. These limitations motivate the use of advanced traffic statistics like derivatives of packet delay for computing the Packet Drop Probability (PDP) based on the rate of change of queue congestion.

In this paper, we propose a novel derivative-based AQM algorithm called dAQM. Building on [4], dAQM uses advanced traffic features like higher-order derivatives of sojourn time and buffer size for computing the PDP. The higher-order derivatives provide an insight into the rate of change of network congestion. For example, the first-order derivative of sojourn time shows the sharp arrival or departure rate of network traffic. It also provides an insight into the local minima or maxima of the sojourn time which aids in estimating the PDP. The second and third-order derivatives of sojourn time provide additional insight into the bursty periods of the network traffic. The traditional AQM algorithms do not take into account the buffer capacity and require separate buffer management algorithms to avoid packet losses due to buffer overflows [5]. The dAQM algorithm includes the first, second, and third-order derivatives of buffer size for estimating the PDP based on varying buffer sizes and avoiding buffer overflows.

The incorporation of advanced traffic features for dAQM algorithm brings several challenges like the design and configuration of higher-order derivatives based AQM. It motivates the first research question, "How can we design and configure the dAQM algorithm relying on higher-order derivatives of sojourn time and buffer size?". Since the estimation of derivatives relies strongly on the incoming traffic statistics, it requires an understanding of various traffic distribution models like Poisson, Weibull, and Pareto, etc., for different traffic classes like Streaming, Gaming, VoIP, HTTP, and FTP. It motivates the next research question, "How would the dAQM perform under various traffic classes and variation in traffic loads, packet sizes, and drop rates building on different transport-layer and application-layer protocols?". Lastly, it requires an understanding of the feasibility and performance of the dAQM. It motivates the last research question, "What are the tradeoffs and performance gains in using dAQM for long and short flows as compared to the traditional AQM algorithms?".

Contributions and Research Findings. In this paper, we develop a novel derivative-based dAQM algorithm for accurate estimation of the PDP. Our major contributions are as follows; (1) Incorporating advanced traffic features, like higher-order derivatives of sojourn time and buffer size, for the development of a novel programmable dAQM algorithm; (2) Understanding of the performance of dAQM by variation in traffic loads, packet sizes, and drop rates; (3) Analysis of the dAQM over five traffic distribution models (Poisson, Weibull, Pareto,

ISBN 978-3-903176-63-8 © 2024 IFIP

CBR, VBR) for Streaming, Gaming, VoIP, HTTP, and FTP; (4) Performance comparison of dAQM with state-of-the-art AQM algorithms including RED, PIE, CoDel, FQ-CoDel, and COBALT. We've publicly released all our research artifacts to aid reproducibility and future work at [6]. Our analysis shows that dAQM provides maximum programmable configurations in Packet Loss Ratio (PLR), queue length, and sojourn time based on the application requirements. Under heavy traffic loads, dAQM provides at least 28% and 62% improvement in delay and queue length, respectively, as compared to the traditional algorithms. For smaller packet size transmissions, dAQM shows an improvement in queue length and sojourn time by at least 57% and 76% over the prior algorithms. dAQM reduced the Flow Completion Time (FCT) from 25-30 s to 15.3 s for FTP traffic which is a reduction of at least 39.7% as compared to the prior algorithms. The research shows that no single AQM fits the needs of all traffic classes and models, and the use of derivative-based dAQM can adapt to network conditions for handling the needs of various traffic classes.

Paper Organization. Sec-II summarizes the related work. The problem statement and proposed approach have been presented in Sec-III and Sec-IV, respectively. Sec-V and Sec-VI show the modeling and performance analysis of dAQM. Sec-VII presents the discussion including feasibility and salient features. Finally, Sec-VIII concludes the paper.

II. Literature Review

In this section, we present some of the most widely used AQM algorithms and the recent research in AQM techniques.

RED uses an Exponential Weighted Moving Average (EWMA) of packet delay to drop packets with a programmed probability [2]. Despite the remarkable performance, the algorithm suffers from the fair-queuing issue where non-responsive flows increase their share of queue. An improvement of RED, called FQ-RED [7], has introduced the fair-queuing mechanism for handling different traffic classes separately. However, the programming complexity and handling of bursty network traffic are major limitations of RED and its variants.

CoDel uses the sojourn time to estimate the queue congestion [3]. If the sojourn time is greater than the target delay (5 ms) for 100 ms, it starts dropping the packets until the delay is less than the target. An improvement to CoDel, called Fair/Flow Queue CoDel (FQ-CoDel) [8], introduced fair queuing to handle all traffic classes separately. Despite being a knobless protocol (with two already configured parameters), strong reliance on sojourn time limits its ability to view the accurate congestion state and handle bursty network traffic.

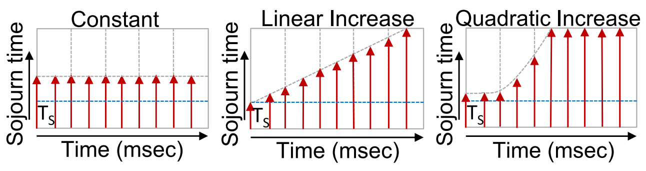
Proportional Integral Controller Enhanced (PIE) is a lightweight AQM algorithm [9]. It uses the average dequeue rate to compute the PDP. Based upon the congestion after packet drops, PIE periodically adjusts the drop probability. An improvement, PI^2, simplifies the tuning method by using the squared drop probability metric [10]. The major limitation is its poor performance for bursty network traffic. Another algorithm, BLUE, uses the PLR and link idle statistics for calculating the PDP. As compared to RED, BLUE provides better PLR and manages the buffer size optimally. Several AQM techniques combine multiple algorithms e.g., CoDel and BLUE Alternate (COBALT) combines the CoDel and BLUE for leveraging the advantages of both algorithms [11]. CAKE [12], combines the COBALT with a traffic shaper and flow isolation module to implement fair queuing AQM.

AQM techniques have also been developed by using specialized traffic features like disturbance observer and smith predictor [13], traffic load pattern [14], buffer size with delay [15], traffic labels [16], traffic classes like best effort, real-time and low latency [17], traffic categorization and prioritization [18], and actively sensing queuing delay [19]. Many researches focused on developing techniques for the PDP computation like reinforcement learning-based drop decisions [20], policy-oriented drop probability based upon the delay and resource utilization [21], flow statistics-based drop estimation [22], and multiple priority-based queues [23]. Several researches focused on parameter tuning using specialized algorithms like reinforcement learning [24], model predictive control theory [25], and reparametrization techniques [1]. A major challenge for the traditional AQM algorithms is the handling of bursty network traffic. It necessitates the development of AQM algorithms relying on a wider range of network statistics, like higher-order derivatives of delay, for accurately estimating the state of congestion in the queues and computing the PDP.

III. Problem Statement

The estimation of accurate queue congestion is a quite challenging task due to the continuously varying network traffic containing bursts, jitters, and delays. In this paper, we focus on the understanding of advanced traffic features, like the higher-order derivatives of packet delays, for computing queue congestion. The accurate queue congestion can be used for the estimation of precise PDP to minimize the packet losses and enhance the QoS for end applications.

In order to understand the use of advanced traffic features and limitations of traditional AQM algorithms, consider the corner cases presented in Fig. 1, and shown below. A high sojourn time greater than the threshold time gives an indication of network congestion, but the precise metric is the rate of change of sojourn time. If the rate of change is zero (case-1), it indicates a less congested situation than the scenario where the rate of change of sojourn time is a positive constant (case-2). A positively increasing rate of change of sojourn time provides information about the sharp increase in sojourn time and the PDP has to be the highest (case-3). The traditional



(a) Case-1. (b) Case-2. (c) Case-3.

Fig. 1: The corner cases for variable changes in sojourn times.

protocols like RED, CoDel, etc. are unable to differentiate between the various scenarios due to their strong reliance on raw statistics of average delay and sojourn time, respectively. These shortcomings can be bypassed by using additional queue features, like the derivatives of sojourn time and buffer size. However, the algorithm design, configurability, and performance of additional features requires further research.

Corner cases for queue management
S_j= Sojourn time; t_j= Timestamp; T_s= Threshold; c = Constant.
Case-1: $S_j > T_s$ & $\Delta S_j/\Delta t_j = 0$
Case-2: $S_j > T_s$ & $\Delta S_j/\Delta t_j = c$ & $c = positive$
Case-3: $S_j > T_s$ & $\Delta S_j/\Delta t_j = Increasing$
Case-4: $S_j > T_s$ & $\Delta S_j/\Delta t_j = c$ & $c = negative$
Case-5: $S_j > T_s$ & $\Delta S_j/\Delta t_j = Decreasing$
Similarly, the 5 corner cases for change in buffer size.

IV. Proposed dAQM Algorithm

We have proposed a novel queue management algorithm, called dAQM, for improving the robustness and adaptiveness of AQM techniques. In this section, we describe the details of the features, algorithm, and programmability of dAQM.

A. Traffic features for dAQM

dAQM uses eight queue features that can estimate the congestion in a robust network. These features include the sojourn time, buffer size, and the consecutive three higher-order derivatives of sojourn times and buffer sizes. The raw sojourn time and buffer size are used as baseline statistics for managing the target limits for lower and upper thresholds of delay and packet loss. An extremely high sojourn time should result in a maximum PDP. If sojourn time is less than the threshold limit, the PDP is a function of the rate of increase of packet delay. The higher-order derivatives of sojourn time and buffer size provide information about the rate of change of congestion in the queues. The first-order derivative (similar to *velocity*) provides the rate of increase of packet accumulation in the queues. Using first-order derivative, the drop decisions can be made timely by observing the trends of increase or decrease in packet delay. The second-order derivative (similar to *acceleration*) provides an enhanced congestion view by showing the rate of change of first-order derivative. Using second-order derivative, the PDP can also change sharply based upon the rate of change of packet delay and buffer size. Lastly, the third-order derivative (similar to *jerk*) provides information about network traffic fluctuations, like packet bursts, by showing the change in the second-order derivative. The higher-order derivatives act as advanced traffic features for estimating the trends of network congestion.

B. dAQM Algorithm

The queue management algorithm of dAQM built over the higher-order derivatives of sojourn time and buffer size, is shown in Algo-1. Before the execution of dAQM, the threshold parameters (T_S, T_B, T, D_r, D_d, N) are initialized. In these parameters, T_S and T_B correspond to the maximum allowed sojourn time and buffer size. T refers to an array containing the derivative thresholds for sojourn time and buffer size. D_r and D_d are the drop rates and drop durations for the N (eight) traffic features. After the initialization, the incoming packets are timestamped while entering the queues to record the sojourn time. The execution of dAQM consists of four phases; (1) Preprocessing phase, (2) Feature extraction phase, (3) Detection phase, and (4) Dropping phase.

Preprocessing Phase. The raw statistics of sojourn time and buffer size are collected at the dequeuing by using the marked timestamps and length of the queues. Sojourn time is collected for every dequeuing packet while the buffer size is computed after a programmed interval t_b.

Feature Extraction Phase. Based upon the raw sojourn time and buffer size, the advanced six features are computed by the dAQM algorithm. These features include the three consecutive higher-order derivatives of sojourn time and buffer size. The first, second, and third-order derivatives are computed by using at least 2, 3, and 4 packets, respectively.

Detection Phase. The detection phase matches the N extracted features with the programmed thresholds (T, T_S, T_B) to identify the state of congestion in queues. The match process is executed based on the sensitivity and critical nature of the features such that the higher-order derivatives have higher priority than the lower-order derivatives. For example, the third-order derivative shows fluctuations in traffic bursts and it carries higher priority in the match process than the second-order derivative. To handle traffic bursts, dAQM checks that the raw sojourn time and buffer size must be greater than the programmed threshold. If any features match the programmed thresholds, dAQM marks the corresponding flags and enters the dropping phase. In case of no match, the algorithm exits without entering the dropping phase.

Dropping Phase. The role of the dropping phase is to estimate the PDP by using the programmed drop rates and drop durations for the respective queue features. The drop rates can vary based on the respective traffic features and packets are dropped based on the PDP. As every feature has a different priority, dAQM allows the provision of unique drop rates and durations. For example, sojourn time crossing the threshold can have a higher drop rate than the third-order derivative which shows only fluctuations in network traffic. After dropping the packets, dAQM recalculates the features to update the programmed drop probability based on the feedback. Finally, it exits the dropping phase after resetting the corresponding marked flags of the network congestion.

C. Programmability of dAQM

The programmability of dAQM strongly relies on the underlying network properties and traffic requirements. In this section, we discuss the programmability of dAQM with mathematical notations shown in Tab. I.

(a) Programming the sojourn time limit. At any instant, the sojourn time $s_j(t)$ must be less than the maximum allowed latency limit $l_{max,j}$ for any flow (Eq. 1). It ensures that the packets reach the destinations within the guaranteed QoS. The latency limit $l_{max,j}$ is defined by the delay requirement of the traffic class ($d_{flow,j}$) and number of hops (n) between sender

Algorithm 1 The proposed dAQM algorithm.

```
Initialization:
Program $T_S, T_B, T[1:8], D_r[1:8], D_d[1:8], N$;
Enqueue:
Timestamp the packet;
Dequeue:
Preprocessing Phase:
Calculate the Sojourn time ($S_j$) and Buffer size ($B_j$);
Feature Extraction Phase:
Compute the 1st, 2nd and 3rd derivatives of Sojourn time;
Compute the 1st, 2nd and 3rd derivatives of Buffer size;
Detection Phase :
if $d^3/dt_3(S_j) > T[8] \wedge S_j > T_S$ then
    f[8] ← 1;
    Enter Dropping phase;
else if $d^3/dt_3(B_j) > T[7] \wedge B_j > T_B$ then
    f[7] ← 1;
    Enter Dropping phase;
else if $d^2/dt_2(S_j) > T[6] \wedge S_j > T_S$ then
    f[6] ← 1;
    Enter Dropping phase;
else if $d^2/dt_2(B_j) > T[5] \wedge B_j > T_B$ then
    f[5] ← 1;
    Enter Dropping phase;
else if $d/dt(S_j) > T[4] \wedge S_j > T_S$ then
    f[4] ← 1;
    Enter Dropping phase;
else if $d/dt(B_j) > T[3] \wedge B_j > T_B$ then
    f[3] ← 1;
    Enter Dropping phase;
else if $S_j > T[2] \wedge S_j > T_S$ then
    f[2] ← 1;
    Enter Dropping phase;
else if $B_j > T[1] \wedge B_j > T_B$ then
    f[1] ← 1;
    Enter Dropping phase;
end if
Dropping Phase:
for $k = 1 \leftarrow 1$ to $N$ do
    if (f[k]==1) then
        Calculate PDP for the drop rate $D_r[k]$;
        Drop packets for the drop duration $D_d[k]$;
        Recompute the derivatives after packet drop;
        Drop again if high derivative;
        Update the $D_r[k]$ and $D_d[k]$;
        f[$k$] ← 0;
        Exit the Dropping phase;
    end if
end for
```

and receiver (Eq. 2). Moreover, the multiple of the number of queued packets $b_j(t)$ and the processing time per packet $T_{proc,j}(t)$ must be less than the latency limit (Eq. 3). It ensures that dAQM drops the packets that cannot be delivered to the receiver within the required latency limit.

$$s_j(t) \leq l_{max,j}; \quad \forall t \in T, \forall j \in J \tag{1}$$

$$l_{max,j} = d_{flow,j}/n; \quad \forall j \in J \tag{2}$$

$$T_{proc,j}(t) * b_j(t) \leq l_{max,j}; \quad \forall t \in T, \forall j \in J \tag{3}$$

TABLE I: The notations used in the description of dAQM.

Notation	Meaning	Notation	Meaning
l_{max}	Latency limit in queue	$T_{proc}(t)$	Processing time
t_d	Processing duration	$T_{tr}(t)$	Transmission rate
$s_j(t)$	Sojourn time	$b(t)$	Buffer capacity
$T_{arr}(t)$	Packet arrival rate	L_b	Upper buffer limit
L_s	Upper latency limit	b_{max}	Max. buffer size
t_b	Sampling interval	n	Nodes in network
d_{flow}	Delay requirement/flow	J	Traffic classes

(b) Programming the buffer size limit. In dAQM, the number of packets $b_j(t)$ in the queue must be less than the capacity of the queue $b_{max,j}$ (Eq. 4). This constraint ensures that dAQM drops the packets that cross the available capacity limit. The difference of packet arrival rate ($T_{arr}(t)$) and transmission rate ($T_{tr}(t)$), and the already stored packets ($b_j(t-1)$) must also be less than the maximum queue capacity (Eq. 5). It ensures that incoming packets and already queued packets never cross the upper capacity limit of the queue to avoid buffer overflow. The threshold of $b_j(t)$ is determined by the hardware configuration and capacity limit of the flow per queue. Critical flows like real-time applications are assigned more capacity than the non-real-time flows to provide better QoS to end applications. A large value results in a lower drop rate than a higher value and vice versa.

$$b_j(t) \leq b_{max,j}; \quad \forall t \in T, \forall j \in J \tag{4}$$

$$(T_{arr}(t,j) - T_{tr}(t,j)) * t_d + b_j(t-1) \leq b_{max,j}; \quad \forall t \in T, \forall j \in J \tag{5}$$

(c) Programming the derivatives of sojourn time. The first, second, and third-order derivatives of sojourn time are calculated by computing the rate of change of $s_j(t)$, $s'_j(t)$ and $s''_j(t)$, respectively, as shown in Eq. 6-8. The sampling interval is related to the packet arrival at the queue.

$$s'_j(t) = \frac{s_j(t) - s_j(t-1)}{\Delta t}; \quad \forall t \in T, \forall j \in J \tag{6}$$

$$s''_j(t) = \frac{s'_j(t) - s'_j(t-1)}{\Delta t}; \quad \forall t \in T, \forall j \in J \tag{7}$$

$$s'''_j(t) = \frac{s''_j(t) - s''_j(t-1)}{\Delta t}; \quad \forall t \in T, \forall j \in J \tag{8}$$

The threshold of $s'_j(t)$ is programmed by ensuring that the queue can handle packets at the current rate for $t_{max,j}$ time, as shown in Eq. 9 and Eq. 10. According to Eq. 9, the first-order derivative of sojourn time $s'_j(t)$ must be less than the upper delay bounds of the latency $L_{s,j}$. The concerning delay bound $L_{s,j}$ for the latency limits is expressed in Eq. 10, where $s_{in,j}$ and $t_{in,j}$ are the captured sojourn times and time indexes for the first packet present in the buffer, and $l_{max,j}$ and $t_{max,j}$ are the projected upper limits. This programmability constraint ensures that the dAQM can capture a sudden increase in delay and drop packets timely without any buffer overflow.

$$s'_j(t) \leq L_{s,j}; \quad \forall t \in T, \forall j \in J \tag{9}$$

$$L_{s,j} = \frac{l_{max,j} - s_{in,j}}{t_{max,j} - t_{in,j}}; \quad \forall j \in J \tag{10}$$

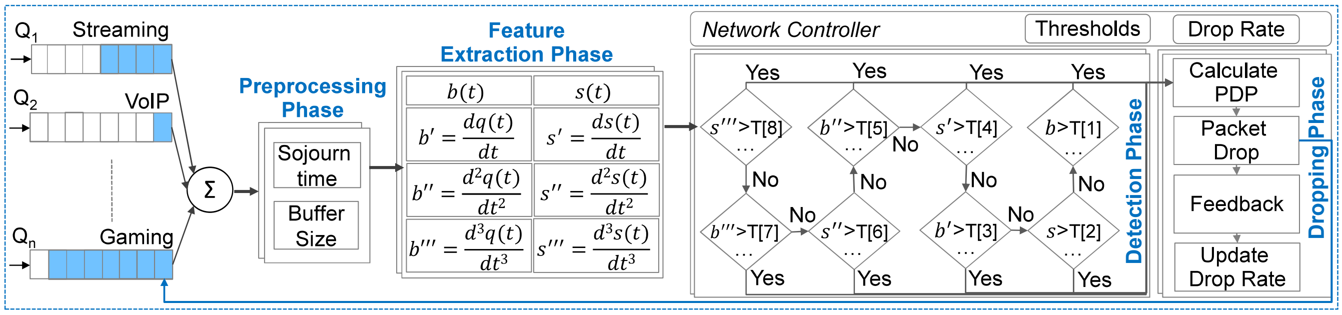


Fig. 2: The system architecture for the execution of dAQM.

Similar to $s_j'(t)$, the threshold of $s_j''(t)$ is defined by calculating the sharp influx of traffic. A small threshold of $s_j''(t)$ drops packets more frequently than a large threshold and reshapes the traffic with equal arrival rates. The $s_j'''(t)$ is the most sensitive parameter as it can measure any change in $s_j''(t)$ which is not possible in other metrics. Analytically, it implies that the integrals of second and third-order derivatives $(s_j''(t), s_j'''(t))$ must be within the upper delay bounds $(L_{s,j})$, as shown in Eq. 11 and Eq. 12, respectively. These constraints ensure that the packets can be processed with the required latency limits of the flow.

$$\int s_j''(t)dt \leq L_{s,j}; \quad \forall j \in J \tag{11}$$

$$\int\int s_j'''(t)dtdt \leq L_{s,j}; \quad \forall j \in J \tag{12}$$

(d) Programming the derivatives of buffer size. Similar to the sojourn time limits, the higher-order derivatives of buffer size are calculated by measuring the rate of change of $b_j(t)$, $b_j'(t)$ and $b_j''(t)$ at fixed sampling intervals t_b, respectively, as shown in Eq. 13-15.

$$b_j'(t) = \frac{b_j(t) - b_j(t-1)}{\Delta t_b}; \quad \forall t \in T, \forall j \in J \tag{13}$$

$$b_j''(t) = \frac{b_j'(t) - b_j'(t-1)}{\Delta t_b}; \quad \forall t \in T, \forall j \in J \tag{14}$$

$$b_j'''(t) = \frac{b_j''(t) - b_j''(t-1)}{\Delta t_b}; \quad \forall t \in T, \forall j \in J \tag{15}$$

The rate of change of queue size $(b_j'(t), b_j''(t), b_j'''(t))$ must be less than the capacity limit of the buffer $(L_{b,j})$, as shown in Eq. 16-18. It ensures that the packets are dropped when dAQM cannot handle them in the required time. The capacity limit of the buffer $L_{b,j}$ is a function of maximum buffer size $(b_{max,j})$ in comparison to the statistics of the first packet $(b_{in,j}, t_{in,j})$, as shown in Eq. 19. These properties cater to the sharp influx of network traffic and make sure that the traffic burst does not occupy buffer beyond the handling limit.

$$b_j'(t) \leq L_{b,j}; \quad \forall t \in T, \forall j \in J \tag{16}$$

$$\int b_j''(t)dt \leq L_{b,j}; \quad \forall j \in J \tag{17}$$

$$\int\int b_j'''(t)dtdt \leq L_{b,j}; \quad \forall j \in J \tag{18}$$

$$L_{b,j} = \frac{b_{max,j} - b_{in,j}}{t_{max,j} - t_{in,j}}; \quad \forall j \in J \tag{19}$$

V. Modeling and Evaluation Setup

In this section, we present the system architecture and the evaluation setup for the proposed dAQM technique.

System Architecture. The system model for a queue management architecture containing the dAQM is shown in Fig. 2. The incoming packets are split into separate queues based on the traffic classes. The packets entering the queues are timestamped to record the sojourn time inside the queues. On the dequeue, the buffer size is measured after regular intervals. The sojourn time and buffer size are fed to the feature extraction module which calculates the higher-order derivatives. Later, dAQM applies a series of *if-else* conditions on the sojourn time, buffer size, and the higher-order derivatives. Based upon the matches, dAQM calculates the PDP and drops the packets.

Evaluation Setup. We developed the simulation model of dAQM in *ns*-3 [26]. The setup consists of 100 clients communicating with 5 servers through a common switch. The switch uses a fair-queuing mechanism where flows of different categories are split into separate queues. The simulation parameters are consistent with the prior researches [27], [28] and are shown in Tab. II. The evaluation of dAQM was performed by simulating clients with five traffic distribution models including constant bit rate (CBR) and variable bit rate (VBR) traffic. The distribution functions for the Poisson process based exponential distribution, Pareto and Weibull distributions are shown in Eq. 20, Eq. 21, and Eq. 22, respectively. The models

TABLE II: Simulation parameters of the network.

Parameter	Value	Parameter	Value
Clients/Servers	100/5	Pkt Size (S/F/H)	1400 B
Link Bandwidth	2 Gbps	Pkt Size (VoIP)	300 B
Trans. Rate (pc.)	10/5/1.5/.128/.08 Mbps	Pkt Size (Gam.)	200 B
Link Latency	1 ms	CBR	ON/OFF=1/0s
Buffer Size	2000 p	VBR	ON/OFF=1/1.5s
Poisson λ (pc.)	892/446/133/50 p/s	Poisson	Mean = $1/\lambda$
Pareto	$k/\alpha = 1.5/2.5$	Weibull	$b/c = 1/1.5$

TABLE III: The programmed parameters of state-of-the-art AQM algorithms and the three configurations of proposed dAQM.

AQM	Parameters				AQM	Parameters		
PIE	Th_{deq} = 20	T_{update} = 15	alpha = 0.125	beta = 1.25	CoDel	Int = 100 ms	Target = 5 ms	
COBALT	P_{drop} = 0	Inc. = 0.08	Dec. = 0.04	BlueTh. = 400	FQ-CoDel	Int = 100 ms	Target = 5 ms	Flows = 1024
RED	Th_{max} = 1000	Th_{min} = 500	QW = 0.002	Target = 5 ms	alpha = 0.01	beta = 0.9		
dAQM$_1$	s = 1000 ms $D_r[2]$ = 0.05 $D_d[2]$ = 400 ms	s' = 10 $D_r[4]$ = 0.05 $D_d[4]$ = 400 ms	s'' = 10 $D_r[6]$ = 0.05 $D_d[6]$ = 400 ms	s''' = 10 $D_r[8]$ = 0.05 $D_d[8]$ = 400 ms	b = 120 p $D_r[1]$ = 0.05 $D_d[1]$ = 400 ms	b' = 1 $D_r[3]$ = 0.05 $D_d[3]$ = 400 ms	b'' = 10 $D_r[5]$ = 0.05 $D_d[5]$ = 400 ms	b''' = 10 $D_r[7]$ = 0.05 $D_d[7]$ = 400 ms
dAQM$_2$	s = 600 ms $D_r[2]$ = 0.5 $D_d[2]$ = 400 ms	s' = 0.1 $D_r[4]$ = 0.5 $D_d[4]$ = 400 ms	s'' = 0.1 $D_r[6]$ = 0.5 $D_d[6]$ = 400 ms	s''' = 0.1 $D_r[8]$ = 0.5 $D_d[8]$ = 400 ms	b = 80 p $D_r[1]$ = 0.5 $D_d[1]$ = 400 ms	b' = 0.1 $D_r[3]$ = 0.5 $D_d[3]$ = 400 ms	b'' = 0.1 $D_r[5]$ = 0.5 $D_d[5]$ = 400 ms	b''' = 0.1 $D_r[7]$ = 0.5 $D_d[7]$= 400 ms
dAQM$_3$	s = 100 ms $D_r[2]$ = 0.98 $D_d[2]$ = 400 ms	s' =0.01 $D_r[4]$ = 0.98 $D_d[4]$ = 400 ms	s'' = 0.01 $D_r[6]$ = 0.98 $D_d[6]$ = 400 ms	s''' = 0.01 $D_r[8]$ = 0.98 $D_d[8]$ = 400 ms	b = 20 p $D_r[1]$ = 0.98 $D_d[1]$ = 400 ms	b' = 0.01 $D_r[3]$ = 0.98 $D_d[3]$ = 400 ms	b'' = 0.01 $D_r[5]$ = 0.98 $D_d[5]$ = 400 ms	b''' = 0.01 $D_r[7]$ = 0.98 $D_d[7]$ = 400 ms

used the traffic flows of Streaming, Gaming, VoIP, HTTP, and FTP. UDP protocol was used for Gaming and VoIP, while TCP protocol was used for Streaming (Video on demand), FTP, and HTTP applications. Since the output is a function of the distribution function and input parameters, the average (normalized) metrics have been shown in the results.

Building on Eq. 1-19, dAQM was programmed in three configurations (dAQM$_1$, dAQM$_2$, dAQM$_3$) with maximum throughput, but different threshold limits. dAQM$_3$ had the lowest thresholds for higher-order derivatives in order to drop packets with minor changes in higher-order derivatives. On the contrary, dAQM$_1$ had the highest threshold limits for higher-order derivatives. The major parameters of dAQM and the prior algorithms are shown in Tab. III.

$$f(t;\lambda) = \lambda e^{-\lambda t}, \quad t \geq 0 \tag{20}$$

$$f(t;k;\alpha) = \begin{cases} \frac{\alpha k^{\alpha}}{t^{\alpha+1}} & t \geq k > 0, \\ 0 & t < k \end{cases} \tag{21}$$

$$f(t;b;c) = \begin{cases} \frac{c}{b}\left(\frac{t}{b}\right)^{c-1} e^{-\left(\frac{t}{b}\right)^{c}}, & t \geq 0 \\ 0, & t < 0 \end{cases} \tag{22}$$

VI. Performance Analysis

In this section, we analyze the performance of dAQM by varying the traffic load, packet size, drop rate, traffic distribution models, flow types, and configurable parameters.

Performance with increased traffic load. The performance of dAQM was analyzed by increasing the traffic load for the HTTP traffic. The results, presented in Fig. 3, show that the increase in load (incoming traffic per unit time) increases the delay until the queues reach their maximum capacity. The increase in load also increases the queue length and sojourn time due to an increase in incoming packets. The results show that the dAQM$_1$ provides the lowest delay and queue length for heavy traffic loads which are at least 28% and 62% less than the traditional AQM algorithms, respectively, except for COBALT which has high PLR and sojourn time. There is no compromise on throughput due to the fair bandwidth sharing of TCP flows. dAQM$_1$ and dAQM$_3$ also provide at least 49%

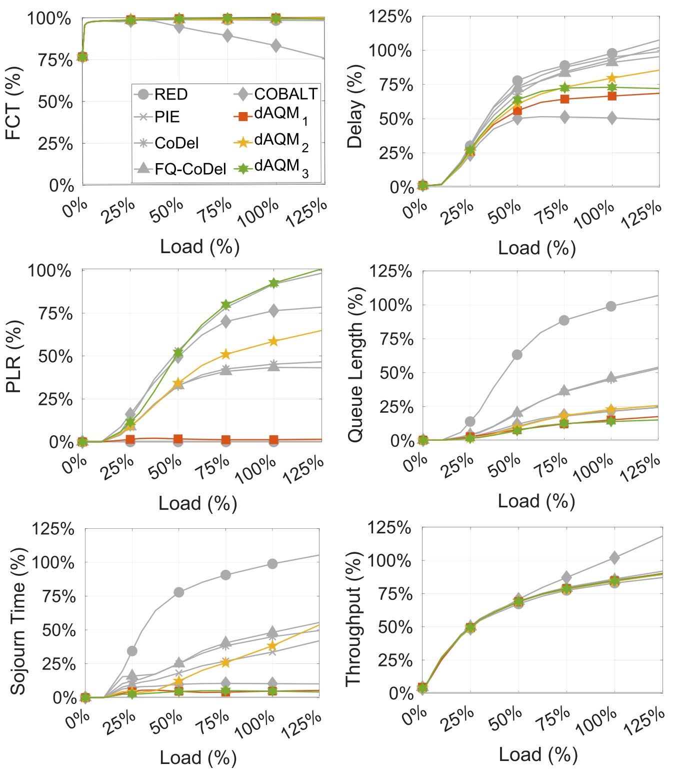


Fig. 3: dAQM performance with increasing traffic load.

improvement in the sojourn time for heavy traffic loads as compared to the traditional protocols. By observing the rate of change of congestion and available queue capacity, dAQM$_1$ provides the lowest PLR which also results in lower end-to-end delay. It increases the flow handling capacity and improves the QoS for end users.

Performance with decreased packet sizes. The performance of dAQM was analyzed by decreasing the packet size from 1400 B (100%) to 50 B, as shown in Fig. 4. By decreasing the packet size, clients send more packets for the same data rate. As a result, the throughput decreases due to an increase in packet handling by the packet processors. Moreover, the sojourn time and queue length also increases which

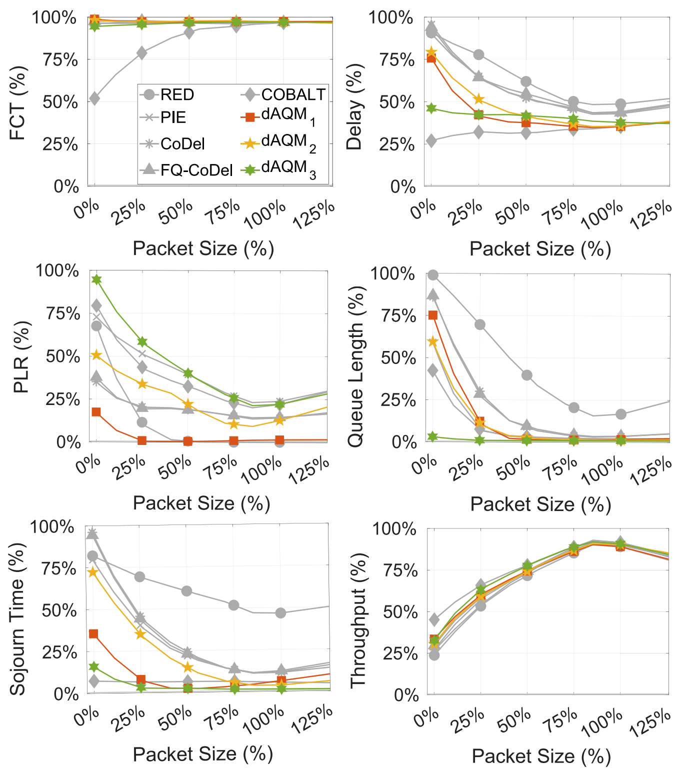


Fig. 4: dAQM performance with decreasing packet sizes.

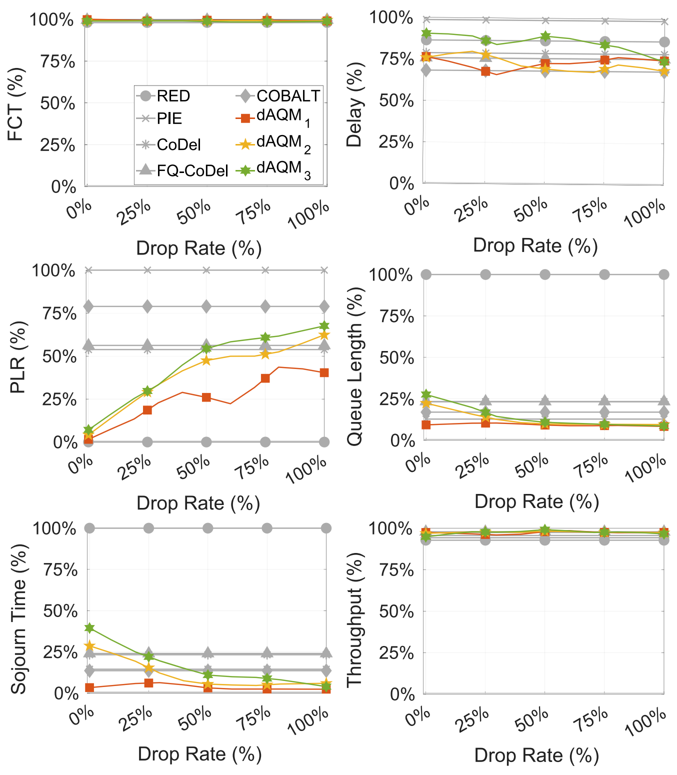


Fig. 5: Performance with increasing dAQM drop rate.

increases the PLR and delay. By increasing the packet size beyond the optimum value (100%), the throughput decreases due to an increase in network congestion. In comparison to the traditional protocols, dAQM showed an improvement in delay and queue length by at least 2.7% and 57%, respectively, except for COBALT which showed poor PLR.

Performance with increased drop rate. The drop rate of the dAQM was increased from minimum to maximum (1%-99%) to understand the network performance, as shown in Fig. 5. The analysis shows that the increase in drop rate increases the PLR, and decreases the sojourn time and queue length in the queues. However, the drop rate only impacts the packets matching the programmed ranges for dAQM features. The traditional AQM algorithms have been simulated with the fixed optimal parameters. The comparison with prior approaches shows that dAQM provides up to 7.7% and 37.01% improvement in delay and queue length, respectively, without any compromise on the throughput.

Variation in traffic distribution models. Considering the requirement of various applications, the performance of dAQM was analyzed over various traffic distribution models and AQM algorithms. The results, presented in Fig. 6, show that dAQM$_1$ provides the lowest PLR for various models. The comparison with prior approaches shows that RED also provides very low PLR, but it compromises on the queue length especially over Poisson distributed flows. Since HTTP is using TCP with short flows, throughput is fairly shared among multiple flows and all AQM techniques have been optimized for maximum throughput. However, the traditional techniques provide up to 60% higher PLR than the dAQM. PIE and RED provide the worst sojourn time, especially over Poisson and CBR flows. The comparison shows that no algorithm fits the needs of all traffic models, but dAQM provides programmability and configurations for catering multiple network flows.

Variation in traffic flows. Considering the different QoS requirements of various applications, Fig. 7 presents the performance of dAQM with prior AQM algorithms for Gaming, VoIP, HTTP, Streaming, and FTP applications under optimal conditions. The analysis shows that dAQM$_1$ provides the lowest PLR which is close to RED and FQ-CoDel. However, RED provides the highest sojourn time which also impacts the end-to-end delay. FQ-Codel provides high throughput, but its FCT is much higher than the dAQM$_1$. Moreover, dAQM provides the configurability of various parameters which can provide a very low sojourn time. Among the other approaches, RED and FQ-CoDel provide high queue length, while PIE, CoDel and COBALT provide high PLR. In comparison to the prior AQM algorithms, dAQM can provide multiple configurations based on the performance requirements of various flows.

Statistical analysis of long vs short flows. The statistical analysis of FTP-based long flows vs HTTP-based short flows for dAQM is presented in Tab. IV. The results show that traditional AQM algorithms like RED and FQ-CoDel have an FCT of 30.2 s and 25.4 s, respectively. However, dAQM requires an FCT from 15.3 to 15.8 s depending upon the configuration. It shows that dAQM provides an improvement

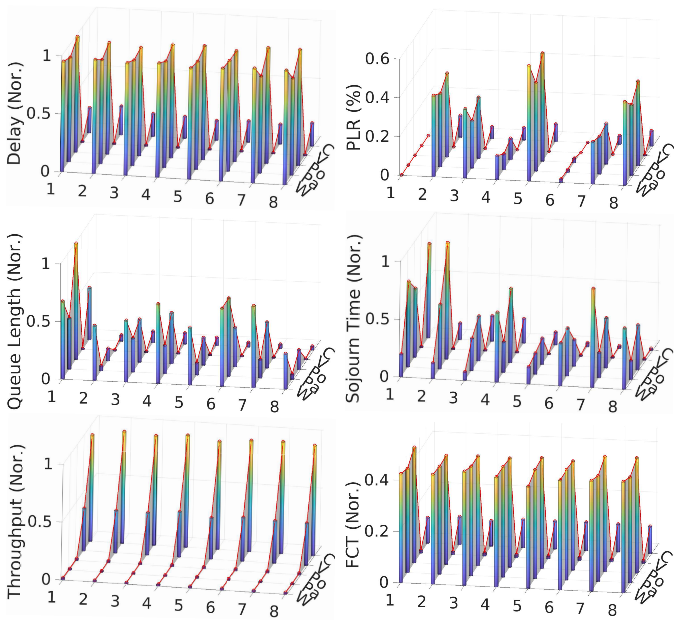


Fig. 6: Performance of HTTP traffic over CBR (C), VBR (V), Poisson (Po), Pareto (Pa) and Weibull (W) traffic distribution models for (1) RED, (2) PIE, (3) CoDel, (4) FQ-CoDel, (5) COBALT, (6) dAQM$_1$, (7) dAQM$_2$, (8) dAQM$_3$.

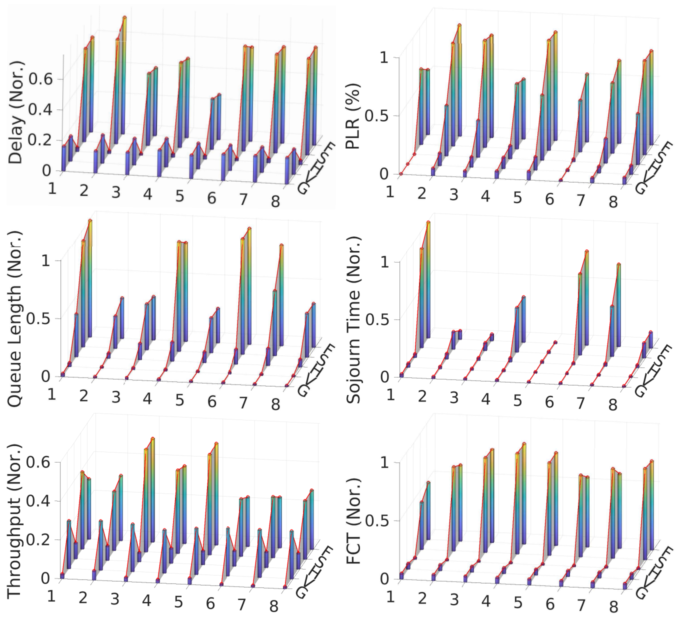


Fig. 7: Performance of Gaming (G), VoIP (V), HTTP (H), Streaming (S), and FTP (F) over Poisson distributed flows. AQM algorithms include (1) RED, (2) PIE, (3) CoDel, (4) FQ-CoDel, (5) COBALT, (6) dAQM$_1$, (7) dAQM$_2$, (8) dAQM$_3$.

TABLE IV: Performance of CBR-based flows for dAQM.

AQM	**FTP Flows**				
	Delay (ms)	TP. (kbps)	QL (pkts)	PLR %	FCT (s)
RED	2403.5	263.6	677.5	0.4	30.2
PIE	1908.4	225.9	94.5	0.6	29.2
CoDel	1349.2	221.2	103.8	0.5	27.8
FQ-CoDel	1955	206.7	303.7	0.6	25.4
COBALT	668.2	302.3	67.2	0.6	25.7
dAQM$_1$	1082.5	292.7	122.9	**0.3**	**15.4**
dAQM$_2$	1138.1	278.2	144.5	**0.5**	**15.3**
dAQM$_3$	1341	291.5	124.5	**0.5**	**15.8**
AQM	**HTTP Flows**				
	Delay (ms)	TP. (kbps)	QL (pkts)	PLR %	FCT (s)
RED	209.2	437.4	112.7	< 0.01	2.81
PIE	239.0	459.5	14.4	0.15	2.84
CoDel	190.7	450.9	26.2	0.08	2.81
FQ-CoDel	183.4	461.3	26.3	0.08	2.85
COBALT	165.2	444.4	19.1	0.12	2.84
dAQM$_1$	182.2	455.7	**9.0**	**0.05**	**2.83**
dAQM$_2$	**152.9**	454.4	**10.9**	**0.09**	**2.84**
dAQM$_3$	176.7	450.6	**9.7**	**0.11**	**2.83**

in FCT by at least 39.7%. For HTTP-based short flows, dAQM improves the queue length and PLR as compared to the traditional protocols. The results show that dAQM has a major impact on FCT for FTP-based long flows.

Variation in individual configurable parameters. All eight dAQM features were varied individually to understand the priority of various features. Fig. 8 shows the 11 histograms referring to 8 dAQM features and the 3 dAQM configurations. Since all configurations have been optimized to maximize the throughput, the results show that the sojourn time is the most effective parameter for reducing the PLR. However, the combination of sojourn time with the higher-order derivatives can minimize the packet delay (sojourn time) in the queues. The combination of all 8 features can provide the lowest PLR or sojourn time based on the traffic flow requirements.

VII. Discussion

In this section, we discuss the feasibility, salient features, and deployment scenarios for the dAQM algorithm.

No One-Size-Fits-All AQM algorithm. The notion of an optimal algorithm that fits the needs of all traffic classes and traffic distribution models is not feasible. The traditional algorithms, like RED, CoDel, define some optimal parameters for queue management. However, our study shows that an algorithm suitable for one traffic class performs worse for another traffic class. It necessitates the use of derivative-based dAQM algorithm which can tailor itself to the dynamics of the network for providing maximum QoS to end users.

Computational complexity and feasibility. dAQM can be implemented using Network Function Virtualization (NFV) which supports more expressive network functions in packet processors. The line-rate switches can support dAQM by incorporating the derivative estimation in the match-action tables. Moreover, the current line of research focuses on more expressive match-action pipelines [29]–[32]. dAQM does not bring any additional computational overhead for the emerging match-action pipelines because advanced traffic features, like higher-order derivatives, are already supported by them through specialized components like Op-amps [33], [34].

Splitting of traffic flows. Since dAQM deals with different traffic flows separately, the incoming traffic can be split into various queues based on their traffic classes. These classes are

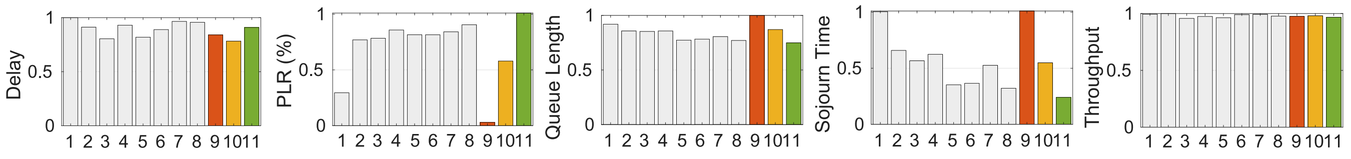


Fig. 8: dAQM analysis by variation in individual parameters of sojourn time (s), buffer size (b) and their higher order derivatives. The normalized histograms are (1) s, (2) s', (3) s'', (4) s''', (5) b, (6) b', (7) b'', (8) b''', (9) dAQM$_1$, (10) dAQM$_2$, (11) dAQM$_3$.

identified using various techniques like protocols and ports-based separation e.g., SIP, RTP for VoIP etc.

Dynamic adaptation in various environments. Instead of the static configuration, dAQM can also dynamically adapt among various configurations. It requires a controller that can monitor the network traffic flows in various queues and adapt between the various configurations. The controller can generate many other configurations based on the requirements.

VIII. Conclusion and Future Work

In this paper, we proposed a novel dAQM algorithm that uses the higher-order derivatives of sojourn time and buffer size for the estimation of drop probability. The analysis showed that dAQM provided an improvement in queue length by 62% and 57% for an increase in traffic load and decrease in packet size, respectively, as compared to the prior AQM algorithms. dAQM provides the lowest PLR with an increase in traffic load. It also effectively caters to the requirements of various traffic classes and models including Poisson, Pareto, etc. dAQM improves the FCT for FTP-based long flows by up to 39.7% as compared to the prior algorithms. In the future, we will study the use of deep learning techniques for the configuration and optimization of dAQM features.

The authors made artifacts publicly accessible at [6].

Acknowledgment

This research is supported by the CogniGron research center and the Ubbo Emmius Funds (University of Groningen).

References

[1] C. Kulatunga, N. Kuhn, G. Fairhurst, and D. Ros, "Tackling Bufferbloat in Capacity-limited Networks," in *Proc. European Conf. Networks and Communications*. IEEE, 2015, pp. 381–385.
[2] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Transactions on Networking*, 1993.
[3] K. Nichols, V. Jacobson, A. McGregor, and J. Iyengar, "Controlled Delay Active Queue Management," RFC 8289, Jan. 2018.
[4] S. Saleh, S. Shu, and B. Koldehofe, "Adaptive In-Network Queue Management using Derivatives of Sojourn Time and Buffer Size," in *Proc. Network Operations and Management Symp.* IEEE, 2024.
[5] S. Das and R. Sankar, "Broadcom Smart-Buffer Technology in Data Center Switches for Cost-Effective Performance Scaling of Cloud Applications," *Broadcom White Paper*, 2012.
[6] dAQM Artifacts. [Online]. Available: https://github.com/rug-ds-lab/2024_Saleh_dAQM-Artifacts
[7] B. Suter, T. Lakshman, D. Stiliadis, and A. K. Choudhury, "Design Considerations for Supporting TCP with Per-Flow Queueing," in *Proc. Int. Conf. Computer Communications*, vol. 1. IEEE, 1998, pp. 299–306.
[8] T. Høiland-Jørgensen, P. McKenney, D. Taht, J. Gettys, and E. Dumazet, "The Flow Queue CoDel Packet Scheduler and Active Queue Management Algorithm," RFC 8290, Jan. 2018.
[9] 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.
[10] R. P. Tahiliani and H. Tewari, "Implementation of PI^2 Queuing Discipline for Classic TCP Traffic in ns-3," in *IFIP Networking Conf.*, 2017.
[11] J. Palmei *et al.*, "Design and Evaluation of COBALT Queue Discipline," in *Int. Symp. Local and Metropolitan Area Networks*. IEEE, 2019.
[12] T. Høiland-Jørgensen, D. Täht, and J. Morton, "Piece of CAKE: A Comprehensive Queue Management Solution for Home Gateways," in *Int. Symp. Local and Metropolitan Area Networks*. IEEE, 2018.
[13] R. Hotchi, H. Chibana, T. Iwai, and R. Kubo, "Active Queue Management Supporting TCP Flows using Disturbance Observer and Smith Predictor," *IEEE Access*, vol. 8, pp. 173 401–173 413, 2020.
[14] A. Adamu, V. Shorgin, S. Melnikov, and Y. Gaidamaka, "Flexible Random Early Detection Algorithm for Queue Management in Routers," in *Int. Conf. Distributed Computer & Communication Networks*, 2020.
[15] V. Addanki, M. Apostolaki, M. Ghobadi, S. Schmid, and L. Vanbever, "ABM: Active Buffer Management in Datacenters," in *Proc. SIGCOMM Conf.* ACM, 2022, pp. 36–52.
[16] G. White and D. Rice, "Active Queue Management in DOCSIS 3.x Cable Modems," *Technical report, CableLabs*, 2014.
[17] G. Park, B. Jeon, and G. M. Lee, "QoS Implementation with Triple-Metric-Based Active Queue Management for Military Networks," *Electronics*, vol. 12, no. 1, p. 23, 2022.
[18] M. Yanev and P. Harvey, "Herding the FLOQ: Flow Optimised Queueing," in *IFIP Networking Conference*. IEEE, 2022, pp. 1–9.
[19] D. M. Havey and K. C. Almeroth, "Active Sense Queue Management (ASQM)," in *IFIP Networking Conference*. IEEE, 2015, pp. 1–9.
[20] M. Kim, M. Jaseemuddin, and A. Anpalagan, "Deep Reinforcement Learning based Active Queue Management for IoT Networks," *Journal of Network and Systems Management*, vol. 29, no. 3, p. 34, 2021.
[21] R. Bless, M. Hock, and M. Zitterbart, "Policy-oriented AQM Steering," in *IFIP Networking Conference*. IEEE, 2018, pp. 1–9.
[22] X. Chen *et al.*, "Fine-Grained Queue Measurement in the Data Plane," in *ACM Int. Conf. Emerging Networking Experiments & Tech.*, 2019.
[23] T. Braud, M. Heusse, and A. Duda, "The Virtue of Gentleness: Improving Connection Response Times with SYN Priority Active Queue Management," in *IFIP Networking Conference*. IEEE, 2018.
[24] D. A. Alwahab, G. Gombos, and S. Laki, "On a Deep Q-Network-based Approach for Active Queue Management," in *Proc. Joint European Conf. Networks and Communications & 6G Summit*. IEEE, 2021.
[25] Q. Xu, G. Ma, K. Ding, and B. Xu, "An Adaptive Active Queue Management based on Model Predictive Control," *IEEE Access*, 2020.
[26] Ns3 network simulator. [Online]. Available: https://www.nsnam.org/
[27] M. Hock, R. Bless, and M. Zitterbart, "Experimental Evaluation of BBR Congestion Control," in *Int. Conf. on Network Protocols*. IEEE, 2017.
[28] X. Du *et al.*, "R-AQM: Reverse ACK Active Queue Management in Multitenant Data Centers," *IEEE Transactions on Networking*, 2023.
[29] S. Saleh and B. Koldehofe, "The Future is Analog: Energy-Efficient Cognitive Network Functions over Memristor-Based Analog Computations," in *Proc. Workshop on Hot Topics in Networks*. ACM, 2023.
[30] S. Saleh and B. Koldehofe, "On Memristors for Enabling Energy Efficient and Enhanced Cognitive Network Functions," *IEEE Access*, vol. 10, pp. 129 279–129 312, 2022.
[31] S. Saleh, A. S. Goossens, T. Banerjee, and B. Koldehofe, "TCAmM$^{\text{CogniGron}}$: Energy Efficient Memristor-Based TCAM for Match-Action Processing," in *Int. Conf. on Rebooting Computing*, 2022.
[32] S. Saleh, A. S. Goossens, T. Banerjee, and B. Koldehofe, "Towards Energy Efficient Memristor-based TCAM for Match-Action Processing," in *Proc. Int. Green and Sustainable Computing Conf.* IEEE, 2022.
[33] S. Saleh, A. S. Goossens, S. Shu, T. Banerjee, and B. Koldehofe, "Analog In-Network Computing through Memristor-based Match-Compute Processing," in *Int. Conf. on Computer Communications*. IEEE, 2024.
[34] S. Saleh and B. Koldehofe, "Memristor-based Network Switching Architecture for Energy Efficient Cognitive Computational Models," in *Proc. Int. Symp. on Nanoscale Architectures*. ACM, 2023, p. 4 pages.