university of groningen

faculty of science and engineering

Adaptive In-Network Queue Management based on the Stochasticity of Network Flows

Bachelor Thesis

University of Groningen

Author: Sunny Shu

Student No.: S3925447

Primary Supervisor: Prof. Dr. Boris Koldehofe

Secondary Supervisor: Saad Saleh

November 16, 2023

Contents

Abstract

There is a vast growing trend in the number of Internet users and Internet of Things (IoT) devices, however, due to buffer bloat and buffer overflow issues, ensuring a consistent Quality of Service (QoS) to the end-users cannot be guaranteed. Active Queue Management (AQM) techniques are designated to address these buffer issues, yet, they have some challenges in accurately estimating the current congestion level within the network buffers. Like Controlled Delay (CoDel), drop packets based on a fixed interval, while Random Early Detection (RED), drops packets based on the calculation of its average queue length. The state-of-the-art algorithms proved to have limited traffic features in indicating the congestion level precisely. Furthermore, they also have limited programmability and configurability. This motivates our research for an innovative AQM technique called Derivative-based Active Queue Management (dAQM). It provides an understanding of the rate of change of congestion in the current network, using the calculation of high-order derivatives of sojourn time and buffer size. The dAQM algorithm uses eight advanced traffic features, including the constant value, and the first, second, and third derivatives of sojourn time and buffer size. Each of the eight traffic features provides unique programmability and configurability for their drop rates, thresholds, and drop durations.

In this thesis, we perform network simulations for five state-of-the-art AQMs (RED, Proportional Integral Controller Enhanced (PIE), CoDel, Fair/Flow Queue CoDel (FQ-CoDel), CoDel and BLUE Alternate (COBALT)) in conjunction with the dAQM under various traffic distribution models (Poisson, Pareto, Weibull, Constant Bit Rate (CBR) and Variable Bit Rate (VBR)). In addition, we would evaluate the effectiveness and adaptiveness of the dAQM algorithm for various flow categories (File Transfer Protocol (FTP), Streaming, Hypertext Transfer Protocol (HTTP), Voice over Internet Protocol (VoIP) and Gaming), and under different transportation layer protocols (Transmission Control Protocol (TCP) & User Datagram Protocol (UDP)). Additionally, we would also evaluate the performance of dAQM under varying network parameters, such as dAQM drop rate, load, and packet size. The performance analysis shows that dAQM is particularly efficient in employment with TCP protocol, since TCP's congestion control mechanism can increase variations in the sojourn time and buffer size, thus aiding its higher-order derivatives calculation. The performance of AQM techniques is highly influenced by the traffic patterns. dAQM has shown superior effectiveness under Pareto-distributed traffic, CBR traffic, and VBR traffic, for both short and long flows. Additionally, dAQM adapts well across varying network parameters and conditions, dAQM consistently ranks among the top performers in key metrics such as low Packet Loss Ratio (PLR), or low queue length, sojourn time, and delay, showing its adaptiveness and robustness.

Acronyms

AQM Active Queue Management.

CAKE Common Applications Kept Enhanced.

CBR Constant Bit Rate.

CDF Cumulative Distribution Function.

COBALT CoDel and BLUE Alternate.

CoDel Controlled Delay.

CWMD Congestion Window.

dAQM Derivative-based Active Queue Management.

DRR Deficit Round Robin.

FCT Flow Completion Time.

FIFO First-In-First-Out.

FQ-CoDel Fair/Flow Queue CoDel.

FQ-RED Fair Queueing with Random Early Detection.

FTP File Transfer Protocol.

HTTP Hypertext Transfer Protocol.

IID Independent and Identically Distributed.

IoT Internet of Things.

PDF Probability Density Function.

PDP Packet Drop Probability.

PIE Proportional Integral Controller Enhanced.

PLR Packet Loss Ratio.

QoS Quality of Service.

RED Random Early Detection.

TCP Transmission Control Protocol.

UDP User Datagram Protocol.

VBR Variable Bit Rate.

VoIP Voice over Internet Protocol.

1 Introduction

Network system has become an integral part of our daily lives, enabling us to communicate, access information, and carry out various tasks. The IoT allows everyday objects, beyond standard computing devices, such as vehicles and cameras, to connect to the network. According to the statistics, there is an exponential increase in the number of Internet users, rising from 4.12 billion since 2019 [1], with projections to 7.5 billion by 2030 [2]; and growing demand for high-speed connectivity, IoT connections are expected to grow by 242% from 2019 to 2030 [3].

1.1 Motivation

The connection of these devices requires the provision of high QoS, to ensure efficient and reliable network performance [4, 5]. However, due to the growing number of devices, maintaining a consistent QoS cannot be guaranteed [6]. The main shortfalls in the optimal management of network queues. In the network system, a buffer is used to temporarily hold the data before it gets transmitted from one place to another. As Gettys [7] points out, problems like the full buffer issue and buffer bloat are evident consequences of buffers that are too large. When packets accumulate in these large buffers, the time they spend waiting to be transmitted or processed increases, leading to increased latency. In reference to the dominant protocol TCP, it uses packet loss as a primary indication of network congestion [7]. When it detects packet loss, it assumes there's congestion and accordingly adjusts its data transmission rate. However, this can be problematic in the context of buffer bloat. Large buffers will delay the packet loss, hence also preventing TCP from recognizing and responding to congestion. Additionally, problems like buffer overflows occur when more data is received than the physical memory can handle, resulting in packet loss [8, 9]. Both buffer bloat and buffer overflow cause degradation to the service quality, such as longer delays, lower connectivity speed, loss of data, and unstable network connection, hence, reducing user experience.

AQM techniques are based on routers or switches, they are effectively controlling congestion, by seeking to control packet drop before a buffer becomes full, thereby reducing latency and improving the QoS.

Furthermore, it is important to offer the right QoS to various types of network traffic. Each traffic category, whether it's video streaming, data transfers, or other types, has its own QoS requirements [10, 11, 12]. These requirements can only be met with precise network traffic modeling and analysis.

AQM Techniques like RED can help diminish persistent queues, yet there's a lack of clear guidance on its parameter configuration in the current network settings, and challenges remain in buffer sizing [13]. As noted by Feng et al. [14], while queue length has been historically used, it's not always an accurate predictor of congestion. In [15], it highlights the problems with mechanisms like RED and CoDel. Such mechanisms, relying heavily on baseline queue statistics, can sometimes result in excessive packet losses and the global synchronization of sources when multiple packets are dropped simultaneously [15]. Furthermore, the AQM techniques, including RED, CoDel, PIE, FQ-CoDel, COBALT, examined in Section 2 offer potential solutions to certain challenges, but they too grapple with inherent limitations. Predominant issues with many of these techniques are their limited configurability and programmability.

These insights have directed our motivation towards innovative AQM technique. Such technique should prioritize better accuracy in estimating the current congestion level within the buffer, also addressing the constraints related to limited features, programmability, and configurability inherent in most AQMs. We developed a new method to precisely estimate the Packet Drop Probability (PDP) by leveraging advanced traffic features such as higher-order derivatives of sojourn time and buffer size [16]. This novel AQM technique is called dAQM. It provides an understanding of the rate of change of congestion in the current network.

The aim of this study is, to investigate the various probabilistic and stochastic traffic models together with the congestion control techniques, in modeling and analyzing network traffic flows in the real-world situation. Specifically, the study will evaluate the performance of different traffic models in simulating real-world network traffic; investigate the applicability of established AQM techniques in conjunction with the novel proposed algorithm dAQM in regulating network congestion and maintaining QoS metrics.

1.2 Problem Statement

Previous AQM algorithms, like CoDel, drop packets on fixed intervals, or RED, drop packets if the average queue size exceeds some thresholds, therefore cannot provide a dynamic and accurate estimation of the congestion level in the network queue. However, with the dAQM algorithm, we can provide a more precise estimation of the PDP using the calculation of the first, second, and third derivatives of sojourn time and buffer size [16].

For example, the first derivative of the sojourn time can provide information on the rate of change of delay, a positive derivative larger than the threshold means the time packets spend waiting in the queue is increasing, indicating an early stage of congestion. Then, the first derivative of buffer size provides information on the rate of change of buffer occupancy, a positive derivative larger than the threshold means the buffer's filling rate is increasing, also an indication of an early stage of congestion. The dAQM mechanism drops packets based on the advanced traffic features while indicating congestion.

1.3 Research Questions

To summarize, this thesis focuses on the following problems:

Q1. How can we design a novel AQM mechanism (dAQM) based on the higher-order derivatives traffic features?

Q2. What measurable impact do the enhanced programming ability and configurability of the proposed AQM algorithm have on network performance (e.g., maximized throughput, optimized buffer size, minimal delay)?

Q3. How would the dAQM perform under different transportation layer protocols (TCP & UDP), flow types (short flow vs long flow), and variations in network parameters (dAQM drop rate, load, packet size)?

Q4. How does the performance of dAQM algorithm compare to traditional AQMs in handling variable network flows?

1.4 Thesis Outline

The thesis is structured into seven main sections. Section 1 introduces the background and motivation of the research and sets out the problem statement, research objectives, and research significance. Section 2, introduces the literature review of the existing related work of the AQM techniques. Section 3 introduces the relevance of using traffic models in our research, and defines the multiple traffic models we would use. Section 4, describes the research methodology and proposed solutions to our problem statement, including the design and implementation of dAQM. Section 5, describes the simulations setup, including the tools and configurations used. Section 6, analyzes the performance of the simulations results, discusses research findings, and answers our research questions. Section 7, concludes the thesis and proposes directions for future research.

2 Background Literature Review

Congestion control implementation can be broadly categorized into two types: end-to-end congestion control and router (or switch) based congestion control [17]. The former is accomplished using the transport layer protocols. For example, TCP uses a sliding window mechanism, which reduces its transmission rate upon detecting congestion often inferred from variations in the Round-Trip Time (RTT). The latter is based on (AQM) mechanisms, which actively drop or mark packets to signal the end system of potential upcoming congestion, and it relies on cooperation with the TCP algorithm to make the congestion control more effective [18, 19]. AQM mechanisms are designed to effectively address the buffer bloat and buffer overflow issues as mentioned in the introduction (Section 1.1).

In this thesis, we focus on the various congestion control algorithms based on the router. The aim is to maintain high throughput and low delay while keeping average queue sizes low. The field of AQM has been extensively studied in the previous research. This section provides a literature review of the related work, specifically their queue discipline, features, working principle, and limitations.

2.1 Random early detection

Random Early Detection (RED) is one of the oldest active queue management techniques, developed in 1993 [18, 20, 21]. It is based on First-In-First-Out (FIFO) queuing discipline. By marking or dropping packets, RED mechanism signals to end systems of upcoming congestion, which can help in avoiding or mitigating congestion if the end systems adjust their transmission rate [19].

The fundamental concept behind the RED algorithm is to reduce congestion by dropping packets with a specific probability in the early stages of congestion [18]. It works on a Drop-Before-Enqueue basis. The probability of dropping a new packet upon its arrival, denoted as P_{drop} is bounded by four parameters:

P_{max}: the maximum dropping probability
Th_{min}: the lower threshold
Th_{max}:the upper threshold
Q_{avg}: the average queue length

The drop probability formula is given by [18, 22]:

$$P_b = \begin{cases} 0 & \text{if } Q_{avg} \in [0, Th_{min}] \\ P_{max}\left(\frac{Q_{avg}-Th_{min}}{Th_{max}-Th_{min}}\right) & \text{if } Q_{avg} \in [Th_{min}, Th_{max}) \\ 1 & \text{if } Q_{avg} \in [Th_{max}, +\infty] \end{cases} \tag{1}$$

• When Q_{avg} is below Th_{min}, P_{drop} there is no packet drop, and the drop probability is 0.
• If Q_{avg} is between Th_{min} and Th_{max}, P_{drop} increases proportionally with the average queue length.
• Otherwise, all incoming packets are dropped. The average queue length, Q_{avg}, is regulated by a weighting factor.
• w_q stands for the queue size weight factor, the current queue length is given by:

$$Q = (1-w_q)\cdot Q_{avg} + w_q \cdot q \tag{2}$$

When multiplex connections reduce their windows at the same time, global synchronization occurs. RED drops packets from multiple TCP flows at different times, to effectively eliminate the synchronization issue [20]. Additionally, since RED is capable of disrupting TCP flow synchronization, it reduces the average queue length, and subsequently shortening end-to-end delays [19]. Some features of RED are designed to be particularly suitable for networks that use the TCP/IP protocol, it can be gradually implemented to the TCP/IP networks with no additional tempering to the underlying transport protocols, and it can be utilized with various packet scheduling and dropping algorithms [18].

2.2 Controlled Delay

Controlled Delay (CoDel) is an AQM mechanism [13, 23]. Unlike RED, CoDel operates independently of network queue size and drop probability [23, 24, 25]. CoDel uses the packet sojourn time, which is the actual delay experienced by a packet in the router queue, to predict congestion [23, 25, 26].

The fundamental concept behind the CoDel algorithm is to continuously monitor the queuing delay of packets in the router, signal the congestion if it exceeds the target (by default of 5 ms) [13, 27]. It works on a Drop-After-Dequeue basis:

In order to distinguish between good and bad queues, CoDel waits for an ‘interval’ of 100 ms (by default) after detecting a delay above the target. If the sojourn time exceeds the target but does not reach a 100 ms ‘interval’, no packets would be dropped. Otherwise, CoDel enters the dropping phase, it maintains a ‘count’ that increases after each (packet) drop. CoDel calculates the next drop time based on ‘count’. The higher the ‘count’, the more frequently the packets would be dropped. Therefore, resulting in a linearly increasing packet drop rate. To reduce feedback delay to the sender, CoDel suggests dropping packets from the head of the queue. When the queuing delay falls below the target delay, CoDel exits the dropping phase and resets its ‘count’.

There are multiple variants for the formula used to determine the value of the count during the recall [27]. One introduced for the next drop time (after the 1st packet has dropped) is calculated as [23, 25]:

$$next\_drop\_time = time + \frac{interval}{\sqrt{count}} \tag{3}$$

2.3 Proportional Integral controller Enhanced

Proportional Integral controller Enhanced (PIE) is an AQM technique [28, 29]. The technique involves three key components [28]: a) enqueuing with random drops; b) regular updates to drop probability; c) estimating the rate of dequeuing.

While enqueuing with random drops, PIE randomly drops packets based on a drop probability p, derived from the ‘drop probability calculation’ component. Similar to RED, PIE also works on a Drop-Before-Enqueue basis.

At regular updates to drop probability. The queue’s average drain rate, denoted as avg_{drate}, is derived from the ‘departure rate estimation’ block. Where the departure rate refers to the rate of successfully dequeued packets (not dropped packets). With a high departure rate, it suggests the network is able to handle the current traffic load, and there is no need to increase the drop probability. The cur_{del} and old_{del}, refer to the current and prior estimated queuing delay. Moreover, the drop probability calculation is influenced by both the current queuing delay estimation and its trend ($\triangle(cur_{del}$ - $old_{del})$, 30 ms by default [24]), i.e., whether the delay is increasing or decreasing. When latency is consistent ($cur_{del} = old_{del}$) and matches the target (ref_{del}, 15 ms by default [24]), the drop probability stabilizes. The parameter α decides the impact of latency deviation from the target on drop probability, while β adjusts based on the latency's direction. The balancing act between latency deviation and jitter is based on the comparative weight of α and β, where this is the typical Proportional Integral controller design [28, 30]. Additionally, the parameters are designed to be self-tuning, ensuring PIE is optimized to its highest performance [28].

The current queuing delay estimation is done through Little's law [28]:

$$cur_{del} = \frac{qlen}{avg_{drate}} \tag{4}$$

Calculate drop probability p as:

$$p = p + \alpha * (cur_{del} - ref_{del}) + \beta * (cur_{del} - old_{del}) \tag{5}$$

Update the prior delay instance:

$$old_{del} = cur_{del} \tag{6}$$

While estimating the rate of dequeuing. The departure rate is only measured when the buffer contains enough data, specifically, when the queue length is larger than a set threshold. Once reaching this threshold, it updates the departure count dq_{count} (the number of bytes departed).

$$dq_{count} = dq_{count} + dq_{pktsize} \tag{7}$$

If dq_{count} is larger than $dq_{threshold}$, it updates the departure rate and reset the dq_{count} to zero and start time to now.

$$dq_{int} = now - start \tag{8}$$

$$dq_{rate} = \frac{dq_{count}}{dq_{int}} \tag{9}$$

$$avg_{drate} = (1 - \varepsilon) \times avg_{drate} + \varepsilon \times dq_{rate} \tag{10}$$

2.4 Fair/Flow Queue Controlled Delay

Fair/Flow Queue Controlled Delay (FQ-CoDel) is an AQM technique as a variant on the CoDel algorithm [31]. The fundamental concept behind the FQ-CoDel algorithm is to establish an even distribution of capacity [24, 31]:

FQ-CoDel classifies incoming traffic flows by hashing their five-tuple (source IP address, destination IP address, source port number, destination port number, and protocol number) and assigning them to one of 1024 sub-queues. The 'count' in FQ-CoDel has a similar concept to the 'count' in CoDel, but each of the sub-queues has its own. It manages each sub-queues individually by the CoDel algorithm and based on a Deficit Round Robin (DRR) queuing mechanism. DRR is a scheduling algorithm that aims to provide guaranteed bandwidth by using a deficit accumulator to determine whether a packet can be transmitted during a scheduling cycle [32]. In each cycle, a fixed amount of data (usually 1500 bytes), is added to the deficit accumulator of each queue. If the size of the pending packet is smaller than the accumulator value, then the packet is eligible to be transmitted. Once the packet is transmitted, its size is deducted from the deficit accumulator associated with its queue.

This approach indirectly enables FQ-CoDel to allocate bandwidth fairly across all sub-queues [24].

2.5 CoDel BLUE Alternate

COBALT, stands for “CoDel BLUE Alternate” [33], it is a variation of the CoDel algorithm [13], integrated with the principles of the BLUE algorithm [14]. COBALT was developed in response to improve the performance of CoDel. It aims to address the gentle behavior of CoDel when the queue delay decreases and then swiftly increases, and its difficulty in handling unresponsive flows [33]. Since the drop rate in CoDel increases linearly over time [23], maintaining the queue delay within the desired range becomes challenging [33].

The fundamental concept behind the COBALT algorithm [33]:

The ‘count’ in COBALT has a similar concept as it was in CoDel. However, COBALT decreases its ‘count’ after exiting the dropping phase instead of resetting it to zero. Leading to a linear reduction in the dropping frequency without actual drops.

COBALT has an integration with the BLUE algorithm. Unlike CoDel, BLUE uses a packet drop probability p_{drop} and adjusts it based on network conditions (queue is idle or queue overflow), making it more effective against a large numbers of unresponsive flows. BLUE increases p_{drop} when packets are dropped due to a full queue and decreases it when the link is idle. It has specific parameters, such as $freeze\_time$ (time interval 100 ms by default), increment (0.0025 by default), and decrement (0.00025 by default), to guide its operation.

COBALT seamlessly combines CoDel and BLUE, allowing them to operate concurrently [33]. BLUE acts only when the queue is at its extremes, ensuring it doesn’t disrupt CoDel’s normal operations [33].

2.6 Section Conclusion

In addition to the above-mentioned AQMs, there is also Fair Queueing with Random Early Detection (FQ-RED) as a variant of RED using fair queuing [34]. There is the BLUE algorithm which uses packet drop probability and the queue condition (idle or overflow), mentioned while introducing COBALT [14]. There is Common Applications Kept Enhanced (CAKE), which combines several traffic features as an integrated method, using bandwidth shaping, flow isolation and hashing, DiffServ handling, and ACK filtering [35]. Despite the many other queue management techniques, we would mainly focus on the performance analysis of the five extensively studied models compared to the new proposed algorithm (dAQM).

The choice of the congestion control model depends on the network characteristics, traffic patterns, and QoS requirements. The congestion control techniques may have limitations in ensuring fairness when multiple traffic flows share a congested network link. This is because the algorithms may prioritize certain flows over others, resulting in an unfair distribution of traffic. Furthermore, decisions for the parametric fitting, modeling assumptions, and the simplicity versus the complexity of the model are significant. For example, for RED it is hard to determine the queue threshold parameters for packet drops, and for FQ-CoDel the bandwidth distribution for different flows. The type of the network may also suggest which queue management technique and queue mechanism to use, thus affecting their performance.

3 Traffic Distribution Models

Analyzing the performance of the various AQM techniques requires deploying them under various traffic distribution models. The traffic models allow for mimicking the complexity and diversity of real-world network traffic and traffic patterns. Traffic distribution models can aid in understanding the performance of AQM under different conditions. Network traffic can be categorized into many types, each with its own traffic pattern, such as FTP, (Video) Streaming, HTTP (Web browsing), Gaming, VoIP (Voice call), etc. Using the traffic models, we can perform simulations under a controlled environment, to understand the behavior of AQMs under real-world conditions. Such as their stability, robustness, adaptiveness, etc., in response to different network conditions (idle traffic, constant traffic, or bursty traffic), and provide insights into the configuration and parameterization of AQMs.

Network traffic modeling can be categorized into two mathematical representations: Continuous-Time Source Models and Discrete-Time Source Models [20]. In continuous time, traffic is represented by its time-varying rate $X(t)$, or the sequence of packet arrival times $\{t_1, t_2, \dots\}$. In discrete time, the interest lies in the non-negative stochastic process X_n as a representation of the source rate sampled at discrete times $n = 1, 2, \dots$. Since users' actions are completely random, network traffic is generated continuously over time in nature. Sometimes with a sudden increase (burst) in the traffic, or with little or no traffic (idle). Thus, we need models that take these features into consideration. Continuous-Time Source Models are useful in terms of capturing the changes in network traffic dynamics, and they support simulations over a long period, i.e., a day, or a few days. Discrete-time models assume traffic is transmitted in discrete intervals, useful for simulations over a short period. However, the arrivals of packets in discrete intervals are a less realistic reflection of the real-world network. Yet, they remain popular models due to the ease of handling discrete data, whereas the Continuous-Time Source Models are usually used for their better precision in modeling and analysis.

3.1 Poisson distribution process

The Poisson distribution process is originally used for data and voice traffic modeling [20]. It is also the most used stochastic process model [36]. The packet arrivals are assumed to be exponentially distributed with arrivals rate λ, and arrivals are Independent and Identically Distributed (IID). The The expected time between each packet arrivals is $1/\lambda$. The derived process from the Poisson Distribution to the Exponential distribution associated with a Poisson process will be shown below [37].

Given the Poisson distribution equation, the probability of observing k events in a fixed interval of time is [37, 38]:

$$f(k;\lambda) = Pr(X = k) = \frac{\lambda^k e^{-\lambda}}{k!} \tag{11}$$

Now, replacing the unit time interval by an interval of arbitrary length t.

$$f(k;\lambda t) = Pr(X = k) = \frac{\lambda t^k e^{-\lambda t}}{k!} \tag{12}$$

In particular, the probability of no point in an interval of length t is $k = 0$, thus,

$$P(X = 0) = e^{-\lambda t} \tag{13}$$

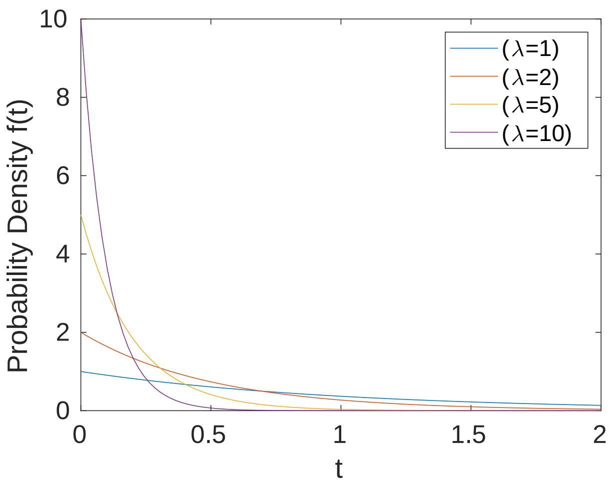


Figure 1: Poisson Distributions with Different λ.

In continuous-time source models like the exponential distribution, packets can arrive at any continuous moment rather than at fixed discrete times. If we are interested in the probability that the time until the first event is less than or equal to t, we have:

$$1 - e^{-\lambda t} \tag{14}$$

Differentiating this Cumulative Distribution Function (CDF) with respect to t gives the Probability Density Function (PDF) of Exponential distribution associated with a Poisson process [20, 37]:

$$f(t;\lambda) = \lambda e^{-\lambda t}, \quad t \geq 0 \tag{15}$$

Fig. 1 shows the Poisson process distribution with different λ values. The Poisson process model has these characteristics:

- It is 'memory-less', the past arrival rate does not interfere with its future arrival.
- Summing two independent Poisson processes with rates λ_1, λ_2, is equivalent to a Poisson process with rate $\lambda_1 + \lambda_2$.

An arrival process refers to a single or a sequence of packets arriving over time. The 'memory-less'property suggests the arrival time of the new packet is independent of the previous arrival time. This introduces an independency between the renewal processes and the former processes. Thus, failing to obtain the traffic burstiness (a sudden increase in the traffic flow) that closely characterizes data traffic [39], and it is inadequate for accurately predicting traffic flow under self-similar features [40].

While the Poisson process has its strengths in modeling various network traffic scenarios, Leland et al. [41] suggest that processes exhibiting statistical self-similarity offer a more precise representation for LAN traffic patterns than traditional Poisson process models. In addition, other recent studies over the past two decades have shown that network traffic exhibits self-similar and fractal characteristics [42, 43]. The distinction between the Poisson models and the self-similar models lies in their statistical characteristics. Furthermore, in various real-world network traffic scenarios, heavy tails have been observed, including situations like file transfer lengths, cellular call durations, and packet inter-arrivals in LANs [36]. The self-similarity here refers to the similarity in traffic patterns when viewed at different time scales. The heavy tail features here refer to the higher probability of getting

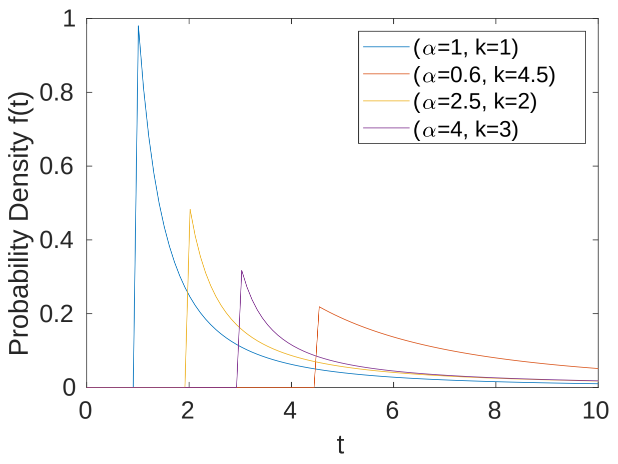


Figure 2: Pareto Distributions with Different Parameters.

extremely large traffic burst, or long periods of idle traffic. There are various approaches to generating self-similar traffic [20, 40, 41], one way to produce self-similar traffic involves multiplexing ON/OFF sources. These sources maintain a constant data transmission rate during ON periods, while the lengths of ON/OFF periods exhibit heavy-tailed characteristics [41].

Yet, the Poisson process is still frequently employed in network traffic engineering due to its ease of analysis using mathematical or statistical methods [36]. Additionally, Poisson arrivals offer a reasonably precise statistical representation for scenarios with multiplexing [36], e.g., when there are many sources sending data, but each source's contribution on its own is very small.

3.2 Pareto Distribution

The Pareto distribution is a probability distribution that exhibits self-similarity properties and a heavy-tailed probability distribution. The study have demonstrated the self-similarity of the Pareto arrival process and its long-range dependent correlation [42]. The Pareto distribution is described by the formula [44, 45]:

$$f(t;k;\alpha) = \begin{cases} \frac{\alpha k^{\alpha}}{t^{\alpha+1}} & t \geq k > 0, \\ 0 & t < k \end{cases} \tag{16}$$

As shown in Fig 2, α represents the shape parameter, and k is the scale parameter, with both $\alpha > 0$ and $k > 0$. The shape parameter decides the shape of the tail behavior of the distribution. As α increases, the distribution tail becomes lighter, and the probability of getting extremely large values decrease. Conversely, as α decreases, the distribution tail becomes heavier, and the probability of getting extremely large values increases. The scale parameter sets the minimual value of the random variable t below which the distribution does not appy, the distribtion is only defined for values greater than or equal to k. In network traffic modeling, this refers to the lower bound for the interarrival times between network packets or events.

Pareto distribution is effective in modeling a high degree of varying network behavior, and large-scale traffic. The model can be easily managed by a small number of parameters. Experiments utilizing the Pareto distribution for network modeling have also been conducted in many previous studies [46, 47, 48, 49].

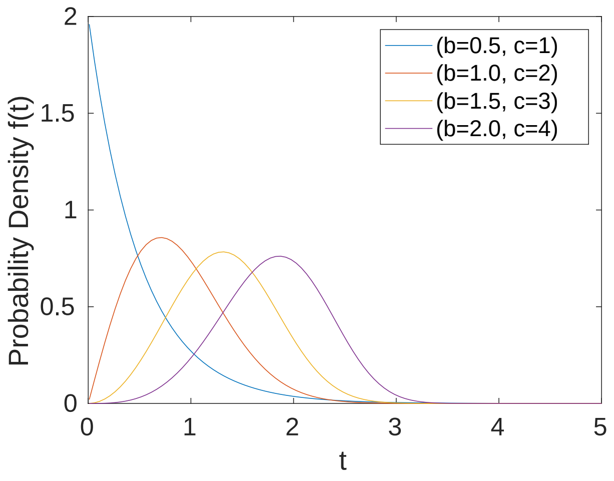


Figure 3: Weibull Distributions with Different Parameters.

3.3 Weibull Distribution

Another model popularly used in network traffic analysis is the Weibull distribution process. It is a heavy-tailed distribution. Described by the PDF [49, 50, 51, 52]:

$$f(t;b;c) = \begin{cases} \frac{c}{b}(\frac{t}{b})^{c-1}e^{-(\frac{t}{b})^c}, & t \geq 0 \\ 0, & t < 0 \end{cases} \tag{17}$$

As shown in Fig. 3. The c represents the shape parameter, and b is the scale parameter, with both $c > 0$ and $b > 0$. For shape $c < 1$, the Weibull distribution has a heavy tail, and the distribution exhibits an increasing failure as time passes. The failure refers to the problems that may arise and potentially degrade the network performance, like packet drop or packet loss. For shape $c > 1$, the failure decreases with time. When shape $c = 1$, the failure remains constant, and the distribution of lifetime is exponential. The scale parameter b decides how thick or thin the distribution along the x-axis would be. Increasing b spreads out the distribution.

The Weibull distribution, like the Pareto distribution, proves to be highly proficient in capturing large-scale and extensively fluctuating network traffic patterns. Prior studies have showcased the use of the Weibull distribution for network modeling [49, 50, 52].

3.4 Constant Bit Rate and Variable Bit Rate

Additionally, we have studied CBR and VBR traffic models [53, 54]. In a CBR model, the data is sent with a consistent bitrate. Whereas in the VBR model, the data is sent with a varying bitrate. The usage of CBR is widely in video streaming applications; VBR is more used in multimedia traffic and other types.

3.5 Section Conclusion

The distribution models mentioned above are commonly used in network traffic analysis. Each model may be better suited for certain types of network simulations. However, they also have limitations, especially in terms of parameter fitting and underlying modeling assumptions.

4 Proposed Technique

In our research, we propose a method to study the network traffic behavior specific to our chosen network type, LAN, across various network flow types. This method incorporates a mix of probabilistic and stochastic distribution models in conjunction with congestion control techniques.

Moreover, the primary focus of the study is on the proposed adaptive queue management algorithm. **dAQM**, with high programmability, configurability and using 8 statistical advanced features in estimating the congestion, as a solution improve the performance of AQM techniques.

We would also testify the performance of various congestion control techniques with different traffic models at the intermediary switch for various metrics, as a crucial step to ensure high QoS. Our methodology is done through the simulations on ns-3 [55], where diverse traffic distribution models, AQM methods, and network flow types were tested using either TCP or UDP transport protocols. Details on our simulations setup can be found in the subsequent sections, specifically in Section 5.

The dAQM algorithm utilizes eight statistical features: the instantaneous sojourn time and buffer size, the first, second, and third derivatives of the sojourn time, and the first, second, and third derivatives of the buffer size. In this section, we aim to find a solution for the research question: *How can we design a novel AQM mechanism (dAQM) based on the higher-order derivatives traffic features?*

4.1 Higher-order derivatives Calculation

In this section, we introduce the steps of the higher-order derivatives calculation of the dAQM algorithm.

4.1.1 Higher-order derivatives of Sojourn Time

The dAQM algorithm was designed with a particular workflow. For example, consider the computation of the first derivative of sojourn time. Where, sojourn time refers to the time a packet spends waiting in the queue, from the moment it enters til the moment it leaves. This required at least two packets in the queue, extracting the wait time of the first and second packets (s_{t1} and s_{t2}), and then computing the average rate of change (first derivative) between these two points as $s_{t1} - s_{t2}$ divided by the packet inter-arrival time ($t_{inter\_arrival}$),

$$r_{sj1} = \frac{s_{t1} - s_{t2}}{\triangle t_{inter\_arrival}} \tag{18}$$

If this ratio exceeded the drop threshold, packets would be dropped.

The dAQM algorithm subsequently also includes the second and third derivatives forms of the sojourn time, which incorporate four and five packets into their derivative ratio calculations, respectively.

Design and Implementation Decision

Calculating the desired derivative ratios required accessing the enqueue time of packets while they remained in the queue, pending a possible drop signal. However, the ns-3 Queue class does not provide a direct method to access the nth item in the queue. Instead, it offers the *DoPeek*() method, which retrieves only the first item without dequeuing it.

There are several potential solutions:

Using an Buffer: One idea was to use an extra buffer to keep track of the timestamps. However, this method would necessitate storing vast amounts of packet data, proving inefficient. Moreover, this approach did not yield the expected results.

Modifying the ns-3 Source: Another consideration was to modify the ns-3 source Queue-Disc class. This, however, would demand an in-depth understanding of the source code's core structure. While attempts were made, they led to several unsolvable bugs.

Temporary Dequeuing: A different approach was to temporarily dequeue the item to obtain the enqueue timestamp and then re-queue it. This method had the potential downside of disrupting the original order of packets in the queue, which could significantly impact computation results and exacerbate fairness issues.

Consequently, we suggest the following method: for retrieving the enqueue times and perform dAQM algorithm, dequeue the necessary number of packets, retrieve their enqueue times, and dequeue times to calculate their waiting time (sojourn time) in the queue. Then, perform calculation on the derivative ratios, and drop the next packet based on the dropping conditions.

The method begins with the detection of a necessary number of packets required by the calculation in the queue. Given our consideration for the calculation of higher-order derivatives of the sojourn time will take five values to obtain an accurate estimation. We check if there are at least five packets in the queue. For the first derivative of sojourn time, the algorithm dequeues the first two packets, calculates their wait times in the queue, and then computes the derivative ratio using the same equation as mentioned above, Eq.(18). The packet inter-arrival time between two consecutive packets is held constant within the algorithm. All packet inter-arrival times during the simulation are recorded without implementing AQM techniques, thereby using a FIFO model. The mode (most frequently appeared value in the data set) of these recorded inter-arrival times (rounded to integer milliseconds) is then utilized to set the inter-arrival time parameter for the dAQM algorithm. If the computed ratio surpasses the threshold, a packet-drop duration with a pre-defined value (e.g., 100ms) is initiated. During this duration, packets are discarded at a pre-defined probability (e.g., 40%).

The computation of the second and third derivatives follows a similar process but incorporates more packets into their calculations and has a more complex set of equations. For the second derivative, the equations involve the calculation of two values '*a*' and '*b*', where '*a*' is the first derivative of the 1st (oldest) and 3rd (oldest) packets, and '*b*' is the first derivative of the 2nd (oldest) and 4th (oldest) packets.

$$a = \frac{s_{t3} - s_{t1}}{2\triangle t_{inter\_arrival}} \tag{19}$$

$$b = \frac{s_{t4} - s_{t2}}{2\triangle t_{inter\_arrival}} \tag{20}$$

Followed by the computation of the second derivative ratio given by:

$$r_{sj2} = \frac{b-a}{\triangle t_{inter\_arrival}} \tag{21}$$

Similarly, for the third derivative, the equations are extended further to include the calculation of an additional value '*c*', where '*c*' is the first derivative of the 3rd (oldest) and 5th (oldest) queue size, and two second-order derivative equations '*d*' and '*e*':

$$c = \frac{s_{t5} - s_{t3}}{2\triangle t_{inter\_arrival}} \tag{22}$$

Eq. (22) calculates the difference in sojourn time between the fifth and third data points, then divides this by twice the packet inter-arrival time. This is the average rate of change (first derivative) of sojourn time between these two points.

$$d = \frac{b-a}{\triangle t_{inter\_arrival}} \tag{23}$$

$$e = \frac{c-b}{\triangle t_{inter\_arrival}} \tag{24}$$

Eq. (23) and Eq. (24) are calculations of the second derivatives.

$$r_{sj3} = \frac{e-d}{\triangle t_{inter\_arrival}} \tag{25}$$

Finally, Eq. (25) calculates the third derivative, by finding the difference between the two second derivative values (d and e), and then dividing by the packet inter-arrival time.

4.1.2 Higher-order derivatives of Buffer Size

The computation of the first, second, and third derivatives of the queue size, are although calculated from different attributes, the concept is similar. Queue Size refers to the size of the buffer, in terms of the number of packets waiting in the queue (not bytes). Instead of using the sojourn time of packets, we have implemented a circular buffer to store 5 queue sizes retrieved at constant time difference (by default 15ms, this can be configured to different values to adjust the frequency of update). As the current queue size is retrieved, it gets stored in this buffer. Once there are 5 values, the first, second, and third derivatives of the queue size will be calculated. Then, the iteration can start again, the sixth value will replace the first value in the circular buffer, ensuring a constant buffer size.

The first derivative of buffer size calculates the difference in queue size between the number of packets obtained at constant time difference (by default 15ms). It is the rate of change of the buffer occupancy. Therefore, the first derivative of buffer size is simply the difference between the 1st (oldest) retrieve queue size, and the 2nd (oldest) retrieve queue size divided by 1 unit (each unit represents a constant time difference, e.g., 15ms).

$$r_{qj1} = \frac{q_{l1} - q_{l2}}{1} \tag{26}$$

Then, the second derivative of buffer size involves the calculation of two values 'a_q' and 'b_q', where 'a_q' is the first derivative of the 1st (oldest) and 3rd (oldest) queue size, and 'b' is the first derivative of the 2nd (oldest) and 4th (oldest) queue size.

$$a_q = \frac{q_{l3} - q_{l1}}{2} \tag{27}$$

$$b_q = \frac{q_{l4} - q_{l2}}{2} \tag{28}$$

Followed by the computation of the second derivative ratio given by:

$$r_{qj2} = \frac{b_q - a_q}{1} \tag{29}$$

Similar to the calculation of sojourn time derivatives, the third derivative of buffer size includes the calculation of an additional value 'c_q', where 'c_q' is the first derivative of the 3rd (oldest) and 5th (oldest) packets, and two second-order derivative equations 'd' and 'e'

$$c_q = \frac{q_{l5} - q_{l3}}{2} \tag{30}$$

$$d_q = \frac{b_q - a_q}{1} \tag{31}$$

$$e_q = \frac{c_q - b_q}{1} \tag{32}$$

Eq. (31) and Eq. (32) are calculations of the second derivatives.

$$r_{qj3} = \frac{e_q - d_q}{1} \tag{33}$$

Eq. (33) calculates the third derivative, by finding the difference between the two second derivative values, and then dividing by 1 unit.

The reason for using five data points for the third derivative calculation falls into a few aspects. Firstly, we want to ensure the accuracy of the result, using five data points can smooth out the noise while still being responsive to the accurate changes in the data, and provide estimation of the congestion in a timely manner. If we use fewer points, the derivative estimation might be too sensitive to transient changes, and using too many points can overly smooth out the data. We want to maintain a balance between them. Additionally, the computation consumption rises with the number of points involved, more points involved in the calculation, would slow down the speed of reacting to congestion.

4.2 Working Principle and Flowchart

The fundamental working of the dAQM is outlined as follows:

- The current time now, is compared to a pre-defined target. This target establishes the time threshold (in milliseconds) beyond which the drop conditions can be activated.

- Subsequently, the algorithm checks for a sufficient number of packets in the queue to facilitate the higher-order derivative calculations of the sojourn time and buffer size. In this implementation, a count of five packets was deemed optimal for the calculation, ensuring accurate derivative ratio results that effectively estimate the current congestion level within the queue. Furthermore, it must also be satisfied that it is in the non-dropping phase, and no drop conditions in this iteration have been met yet.

- The algorithm then invokes functions to compute the first, second, and third derivatives of sojourn time and buffer size. The specifics of these calculations were detailed in Section 4.1. Additionally, the current sojourn time and buffer size are retrieved for comparison against constant thresholds.

- During the comparison of advanced derivative features with their corresponding pre-defined thresholds, the process begins by comparing the absolute value of the third derivative of the sojourn time against its designated threshold (or alternatively, comparing the derivative ratio with both the positive and negative threshold to achieve the same effect).
  - The derivatives derived from the baseline statistics can be in either positive or negative directions, indicating an increase or a decrease in the metric over time.
  - Since we are measuring the rate of change and both rapid increases and rapid decreases are problematic, then considering both positive and negative thresholds makes sense.
  - We want to ensure that if the derivative ratio remains within the range of negative threshold and positive threshold, no action is taken. Otherwise, packets should be dropped.

- If the drop condition of the third derivative of sojourn time is not met, the third derivative of the queue size is considered. If still unmet, the algorithm proceeds to the second derivative of the sojourn time, and so forth, until a condition is satisfied, or no conditions are met. Once a condition is met, the subsequent conditions are not evaluated. Instead, it will signal to drop the packet while dequeuing, and set up a scheduler to start the dropping phase (where the dropping phase is to estimate the PDP) for a pre-defined duration, to drop packets with a pre-defined probability at enqueue stage. Each of the eight features offers configurability for drop duration, drop probability, and threshold values.

The following flowchart in Fig. 4 (made with PlantUML [56]) provides a visual representation of the dAQM algorithm's working process. Where, derivatives of Sjs refer to the 1st, 2nd, and 3rd $(Sj1, Sj2, Sj3)$ derivatives of sojourn time; derivatives of Qjs refers to the 1st, 2nd, 3rd $(Qj1, Qj2, Qj3)$ derivatives of buffer size; Sj refers to the current sojourn time of the oldest packet in the queue; and

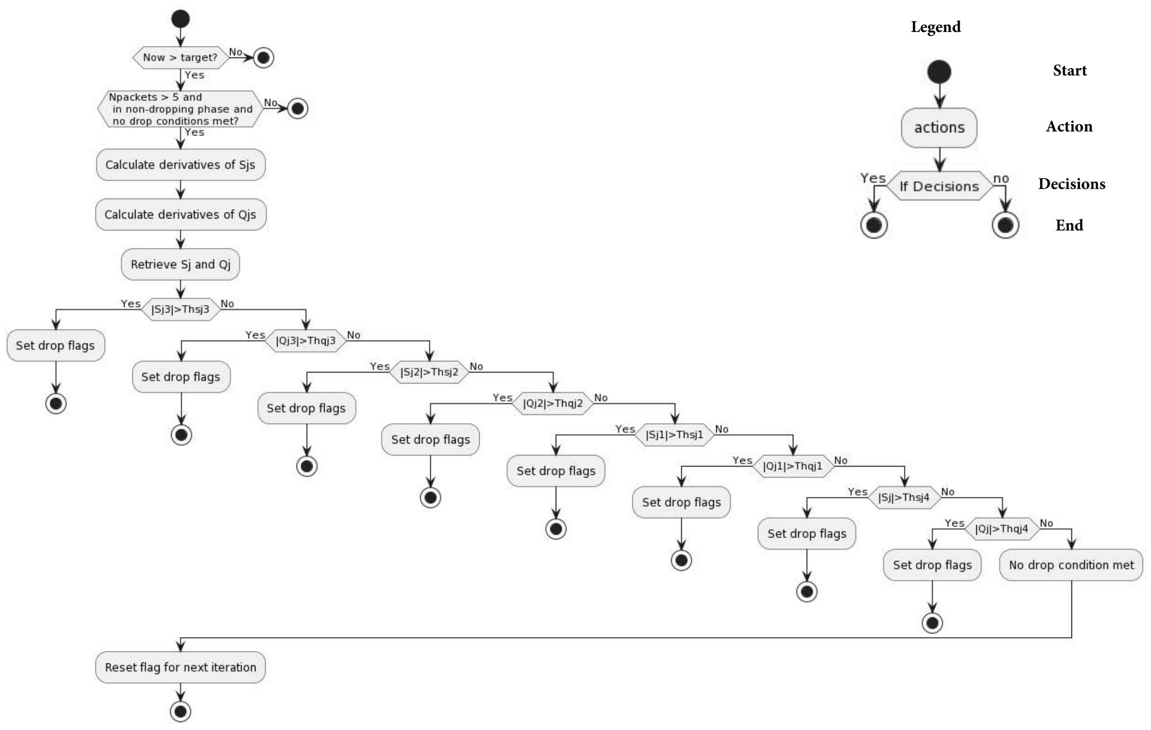


Figure 4: Flowchart: The decision-making process of the dAQM Packet Drop Mechanism based on sojourn time and buffer size derivatives.

Qj is the current number of packets waiting in the queue. Their absolute values indicate the magnitude of the increase/decrease in the rate of change and are compared to their specific thresholds, $Thsj3, Thqj3, Thsj2, ...Thqj4$ corresponds to the threshold of each feature.

4.3 Operational Design and Pseudocode

Hereafter, we delve deeper into the details of the dAQM queue discipline by outlining its core operations: enqueue, dequeue, and the dAQM drop mechanism. Presented in pseudocode format, these algorithms offer a clear and structured insight into the sequential steps and logic behind each function.

4.3.1 Stage 1: Enqueue

In the enqueue stage, it ensures that the queue does not exceed its maximum size and the system handles packet dropping (for the incoming new packets) through a controlled method (dropping phase) based on certain conditions.

dAQM works on a drop-before enqueue approach. Similar to RED, but not entirely the same. RED also decides whether an incoming packet should be dropped at its enqueue stage, with a certain probability before the queue becomes full, based on the average queue size calculated. While CoDel, makes the drop decisions during the dequeue process, not before enqueuing. It can be either in a dropping phase or non non-dropping phase. It checks if the sojourn time of packets exceeds a target for a pre-defined interval, and then enters a dropping phase to decide whether the packet should be dropped or not. It leaves the dropping phase when the sojourn time is below the target. Similar to CoDel, dAQM also has a dropping and non-dropping phase. The mechanism for controlling the dropping phase is set up in the dequeue stage, and packet dropping is in the enqueue stage based on the dropping phase.

Algorithm 1 Enqueue

```
 1: function ENQUEUE
 2:     if current queue size + incoming packet size > max queue size then
 3:         drop the packet
 4:     end if
 5:     if in the dropping phase ($drop_{packets} == true$) then
 6:         Drop the packet based on a pre-defined probability
 7:         Reset drop flags
 8:     end if
 9:     Enqueue packet to queue
10:     Record the enqueue time
11: end function
```

The above algorithm shows the pseudocode of the enqueue stage.

- Check Queue Size (L: 2-4): It checks if the current queue size plus the size of the new packet will exceed the maximum queue size. If yes, it drops the packet.
- Check Dropping Phase (L: 5-8): If the system is in a dropping phase, it drops the packet based on a predefined probability and resets any flags related to dropping. The flags tell information about which of the eight features has triggered the drop, because it is programmable to set different thresholds, drop rates, and drop durations for each of the features, thus, drop triggered by different conditions can have different drop probability.

- Enqueue Packet and Timestamp Enqueue Time (L: 9-10): If the packet hasn't been dropped due to the above conditions, it proceeds to add the packet to the queue.
- After enqueuing the packet, it timestamps the time at which the packet was added to the queue.

4.3.2 Stage 2: Dequeue

In the dequeue stage, it decides whether the packet will be dropped or dequeued based on certain conditions. The dropping phase is set up in the dequeue stage. If any of the drop conditions of the eight features have been met. It will signal the system to drop the packet and start up a dropping phase so that new incoming packets can be dropped with a certain probability at the enqueue stage. It will exit the dropping phase after a pre-defined period.

Algorithm 2 Dequeue

```
1:  function DEQUEUE
2:      Get the packet aiming to be dequeued
3:      if queue is empty then
4:          Return 0
5:      end if
6:      if one of the dAQM drop condition has been met then
7:          Drop the packet
8:          Start a dropping phase (set $drop_{packets} == true$)
9:          Schedule to exit dropping phase (set a call to a scheduler which sets $drop_{packets} == false$)
    after a pre-defined duration
10:         Get the next packet from queue
11:     end if
12:     return dequeued packet
13: end function
```

The above algorithm shows the pseudocode of the dequeue stage.

- Get the Packet from Queue (L: 2): Get the packet from the queue aiming to be dequeued.
- Check Empty Queue (L: 3-5): If the queue is empty, the packet cannot be dequeued.
- Check dAQM Drop Conditions status (L: 6-11): if one of the dAQM drop conditions has been met, it drops the current packet. Then, it starts a dropping phase by setting the variable $drop_{packets} == true$, for a pre-defined duration, by calling a ns-3 scheduler that sets $drop_{packets} == false$ after the pre-defined duration ends. Then, it attempts to get the next packet from the queue.
- Dequeue Packet (L: 12): Return the packet aiming to dequeue.

4.3.3 Stage 3: Packet Drop Mechanism

In the packet drop mechanism stage, it checks for several conditions before calculating the higher-order derivatives and comparing them for each drop condition. It follows a hierarchical order while checking for each drop condition. Because the third derivatives can indicate more severe fluctuations in the network traffic and congestion, the system checks firstly the third derivatives, then the second derivatives, then the first derivatives, and lastly, their constant values. When a particular condition has been met, it will set its flag to true, thus, in the enqueue stage, the system will know which drop condition it should follow.

Algorithm 3 dAQM Packet Drop Mechanism

```
function DropPacket
    if Now > target then
        if $N_{\text{packets}} > 5$ and $drop_{packets} == false$ and $flag_{condition} == false$ then
            Calculate the 1st, 2nd, and 3rd ($S_{j1}$, $S_{j2}$, $S_{j3}$) derivatives of Sojourn time
            Calculate the 1st, 2nd, and 3rd ($Q_{j1}$, $Q_{j2}$, $Q_{j3}$) derivatives of buffer size
            Retrieve the current sojourn time $S_j$ of the oldest packet and the current queue size $Q_j$
            if drop condition $|S_{j3}| > Th_{sj3}$ is met then
                Set drop flags
            else if drop condition $|Q_{j3}| > Th_{bj3}$ is met then
                Set drop flags
            else if drop condition $|S_{j2}| > Th_{sj2}$ is met then
                Set drop flags
            else if drop condition $|Q_{j2}| > Th_{bj2}$ is met then
                Set drop flags
            else if drop condition $|S_{j1}| > Th_{sj1}$ is met then
                Set drop flags
            else if drop condition $|Q_{j1}| > Th_{bj1}$ is met then
                Set drop flags
            else if drop condition $|S_j| > Th_{sj4}$ is met then
                Set drop flags
            else if drop condition $|Q_j| > Th_{bj4}$ is met then
                Set drop flags
            end if
        end if
    end if
end function
```

The above algorithm shows the pseudocode of the packet drop mechanism stage.

- Check The time of Now (L: 2, 25): Check if the current time now is greater than the target. The algorithm should not start the calculation of the higher-order derivatives at the early stage when there is less likely to be congestion.
- Check Queue Size, Dropping Phase Status, Dropping flag status (L: 3, 24): It checks if there are enough packets for the calculations. There is no need to do new rounds of calculation if the system is currently in an active dropping phase. It must also satisfy that no drop conditions flags have been set to true, meaning no drop conditions have been met in this iteration so far.

- Calculate higher-order derivatives of Sojourn Time (L: 4): Call to functions that perform the calculations of higher-order derivatives of sojourn time.
- Calculate higher-order derivatives of Buffer Size (L: 5): Call to functions that perform the calculations of higher-order derivatives of buffer size.
- Check the Constant Values of Sojourn Time and Buffer Size (L: 6): Retrieve the current sojourn time of the oldest packet and the current number of packets in the queue.
- Check Drop Conditions (L: 7-23): Following the hierarchical order, each derivative ratio is compared to its threshold, once any condition is satisfied, it changes the specific drop flag related to that feature, and also sets the $flag_{condition} == true$ indicating that a condition has already been met in this iteration, and it is ready to signal the deuque stage for setting up a new dropping phase.

4.3.4 Dropping Phase and Non-dropping Phase

The terms dropping phase and non-dropping phase of the dAQM have been occurring multiple times. To make it clear, in this subsection, we explain their usage in the system.

The dropping phase is set up in the dequeue stage. It allows the system to drop packets in the enqueue stage based upon certain probability, it lasts for a pre-defined period.

The non-dropping phase is the state when the system is not within a dropping phase. During this phase, the packets are enqueued normally, only checking if the new incoming packet would overflow the buffer, and it doesn't drop packets with a probability.

4.3.5 Choices of Uniform Dropping Conditions

While deploying and testing the dAQM algorithm, it was discovered that the performance of the dAQM algorithm demonstrated noticeable improvement when a uniform drop condition was applied across all eight traffic features. This finding was consistent irrespective of the specific drop rate, such as 40%, applied uniformly across the algorithms. This approach proved superior compared to scenarios where unique drop rates were designated to each derivative, such as assigning drop rates of 20%, 30%, 40%, and 50% to the constant, first, second, and third derivatives, respectively. Assigning different drop rates leads to performance loss due to extra computational steps. Furthermore, designating an equal 40% drop rate individually to each feature did not yield similar performance gains as applying the same rate collectively, has supported our statement. This discrepancy arises from the additional computational overheads to evaluate separate conditions, which subsequently impaired the performance of the dAQM algorithm. This deviation in performance became increasingly pronounced as the number of clients involved in the simulations increases. Therefore, although it is programmable to assign different drop rate and drop duration conditions to each of the eight traffic features, we keep the design of using a uniform drop rate and drop duration across all features to enhance and optimize dAQM's performance.

5 Simulations Setup

In this section, we introduce all the specifics of setting up the simulations, including the tools and technologies; network topology, configurations and parameterizations of the traffic models, flow types, and AQM techniques; and the performance metrics used by our analysis.

5.1 Tools and Technologies

• **Operating System**: Ubuntu-20.04.

• **Software**: open-source network simulator 3 (ns-3) [55] of version ns-3.35 under the terms of the GNU General Public License version 2; gnu-plot [57] and Matlab [58] for data analysis and visualisation; source code editor gedit [59].

• **Libraries**: ns-3 has a modular architecture, the modules (ns-3 libraries) we have used include: Core Module, Network Module, Internet Module, Traffic Control Module, CSMA Module, Applications Module, Flow Monitor Module, Random Variable Stream Module, Queue Disc Module, Object Factory Module, Drop Tail Queue Module, Simulator Module and Log Module.

• **Languages**: C++, Python, gnu-plot script, MATLAB

• **GitHub Repository**: for archiving the complete database, available at GitHub Repository [60].

5.2 Configurations and Parameterization

This section introduces the setup related to network topology, configurations of the traffic distribution models, AQM techniques, and the various traffic flow categories.

5.2.1 Topology

The simulations were conducted using ns-3 to emulate a Local Area Network (LAN). Within this emulation, we interconnected 100 clients and 5 servers through a single switch employing a Carrier Sense Multiple Access (CSMA) channel. A visual representation of this configuration can be found in our network topology, in Fig. 5.

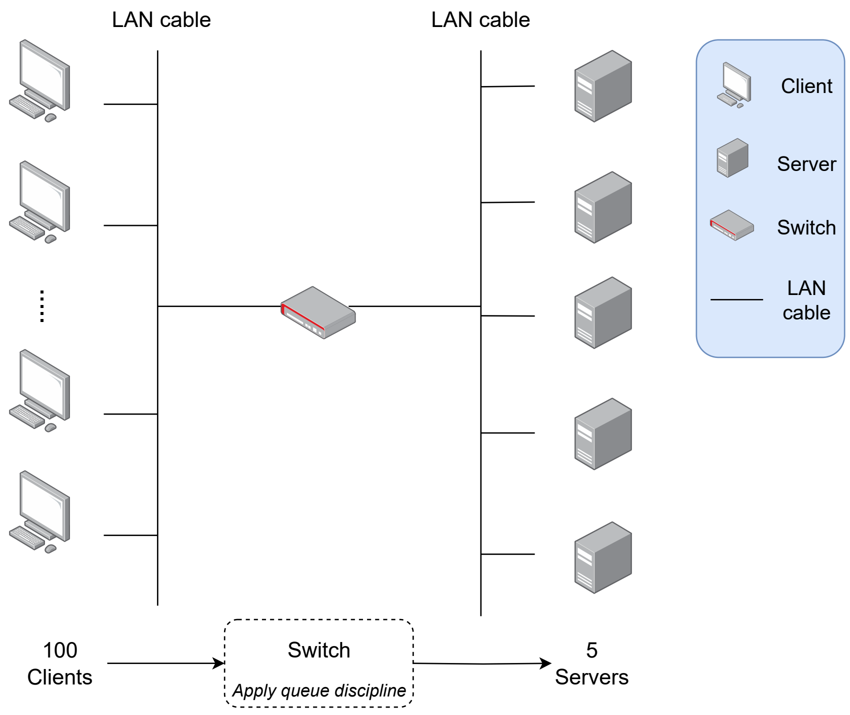


Figure 5: Network Topology.

5.2.2 Traffic Models

Our study employed various traffic distribution models, namely the Poisson distribution process, Pareto distribution, Weibull distribution, Constant Bit Rate (CBR), and Variable Bit Rate (VBR), with their specific parameter settings detailed in Table 1.

Table 1: Traffic Models Configuration Setup.

Traffic Models	**Formula**	**Eq.**	**Parameter**	**Value**
Poisson	$f(t;\lambda) = \lambda e^{-\lambda t}, \quad t \geq 0$	(3)	λ	892/446/133/50 (packets/s)
Pareto	$f(t;k;\alpha) = \begin{cases} \frac{\alpha k^{\alpha}}{t^{\alpha+1}} & t \geq k > 0 \\ 0 & t < k \end{cases}$	(5)	k, α	$k = 1.5, \alpha = 2.5$
Weibull	$f(t;b;c) = \begin{cases} \frac{c}{b}(\frac{t}{b})^{c-1} e^{-(\frac{t}{b})^{c}}, & t \geq 0 \\ 0, & t < 0 \end{cases}$	(6)	b, c	$b = 1, c = 1.5$
CBR	-	-	ON/OFF	ON=1s, OFF=0s
VBR	-	-	ON/OFF	ON=1s, OFF=1.5s

In the simulations, each single client will follow a distribution model, not to be confused with, multiple clients setting up using one distribution model. Moreover, multiple clients would be set up simultaneously for the stochastic models (one client undergoes one distribution model): Poisson, Pareto, and Weibull distribution. They will be set up at the same time, but, their actual start time depends on the result of the random variables involved. Nevertheless, they should still start at similar times, as the (distribution model) parameters used for each client would be the same. Conversely, the clients in the VBR and CBR traffic will not be set up simultaneously, instead, the start time of each client will have some time differences. For example, a CBR client would start at $time = 0s$, the next CBR client would start at $time = 1s$, the third CBR client would start at $time = 2s$, etc.

5.2.3 AQM Models

Our analysis aimed to compare the effectiveness and performance of various state-of-the-art AQM techniques, along with eight programmable features (dAQM$_{f1}$ - dAQM$_{f8}$) used individually, and three configurations (dAQM$_{c1}$ - dAQM$_{c3}$) with all features used simultaneously. The configurations and their specific details are provided in Table 2.

Table 2: Active Queue Management Techniques Configuration Setup.

AQM	**Parameters Details**							
RED	Min.Th=500	Max.Th=1000	QW=0.002	Target=5 ms	α=0.01	β=0.9		
PIE	Deq.Th=20	Tupdate=15	α=0.125	β=1.25				
CoDel	Int.=100 ms	Target=5 ms						
FQ-CoDel	Int.=100 ms	Target=5 ms	Flows=1024					
COBALT	Pdrop=0	Incr.=0.08	Decr.=0.04	Blue.Th=400				
	const. Sj	**1st** Sj_1	**2nd** Sj_2	**3rd** Sj_3	**const.**Qj	**1st** Qj_1	**2nd** Qj_2	**3rd** Qj_3
dAQM$_{f1}$	t=200 ms d_r=0.9 d_u=200 ms	- - -	- - -	- - -	- - -	- - -	- - -	- - -
dAQM$_{f2}$	- - -	t= 0.01 d_r=0.9 d_u=200 ms	- - -	- - -	- - -	- - -	- - -	- - -
dAQM$_{f3}$	- - -	- - -	t= 0.01 d_r=0.9 d_u=200 ms	- - -	- - -	- - -	- - -	- - -
dAQM$_{f4}$	- - -	- - -	- - -	t= 0.01 d_r=0.9 d_u=200 ms	- - -	- - -	- - -	- - -
dAQM$_{f5}$	- - -	- - -	- - -	- - -	t=20 p d_r=0.9 d_u=200 ms	- - -	- - -	- - -
dAQM$_{f6}$	- - -	- - -	- - -	- - -	- - -	t= 0.01 d_r=0.9 d_u=200 ms	- - -	- - -
dAQM$_{f7}$	- - -	- - -	- - -	- - -	- - -	- - -	t= 0.01 d_r=0.9 d_u=200 ms	- - -
dAQM$_{f8}$	- - -	- - -	- - -	- - -	- - -	- - -	- - -	t= 0.01 d_r=0.9 d_u=200 ms
dAQM$_{c1}$	t= 1000 ms d_r= 0.05 d_u= 400 ms	t= 10 d_r= 0.05 d_u= 400 ms	t= 10 d_r= 0.05 d_u= 400 ms	t= 10 d_r= 0.05 d_u= 400 ms	t= 120 p d_r= 0.05 d_u= 400 ms	t= 1 d_r= 0.05 d_u= 400 ms	t= 10 d_r= 0.05 d_u= 400 ms	t= 10 d_r= 0.05 d_u= 400 ms
dAQM$_{c2}$	t= 600 ms d_r= 0.5 d_u= 400 ms	t= 0.1 d_r= 0.5 d_u= 400 ms	t= 0.1 d_r= 0.5 d_u= 400 ms	t= 0.1 d_r= 0.5 d_u= 400 ms	t= 80 p d_r= 0.5 d_u= 400 ms	t= 0.1 d_r= 0.5 d_u= 400 ms	t=0.1 d_r= 0.5 d_u= 400 ms	t=0.1 d_r= 0.5 d_u= 400 ms
dAQM$_{c3}$	t=100 ms d_r= 0.98 d_u= 400 ms	t=0.01 d_r= 0.98 d_u= 400 ms	t= 0.01 d_r= 0.98 d_u= 400 ms	t= 0.01 d_r= 0.98 d_u= 400 ms	t= 20 p d_r= 0.98 d_u= 400 ms	t= 0.01 d_r= 0.98 d_u= 400 ms	t= 0.01 d_r= 0.98 d_u= 400 ms	t= 0.01 d_r= 0.98 d_u= 400 ms

In this table, the dAQM parameter settings are as follows: 't' stands for the feature threshold, 'd_r' denotes the drop rate, and 'd_u' indicates the drop duration. These parameters are essential for an accurate estimation of the PDP. Additionally, dAQM$_{f1}$, corresponds to the constant value of sojourn time; dAQM$_{f2}$, is the first derivative of sojourn time; dAQM$_{f3}$, is the second derivative of sojourn time; dAQM$_{f4}$, is the third derivative of sojourn time; dAQM$_{f5}$, is the constant value of buffer size;

dAQM$_{f6}$, is the first derivative of buffer size; dAQM$_{f7}$, is the second derivative of buffer size, and dAQM$_{f8}$, is the third derivative of buffer size. Then, the dAQM$_{c1}$, dAQM$_{c2}$, dAQM$_{c3}$ are three configurations set up with different feature parameters settings.

5.2.4 Traffic Flows

We simulated various types of network flows to ensure a comprehensive study. These included FTP, Streaming, HTTP, VoIP, and Gaming, with their specific configurations detailed in Table 9. Where these parameter configurations align with those established in prior studies [61, 62, 63, 64]. In our simulations, FTP, Streaming, and HTTP flows operate using the TCP protocol. In contrast, flows like VoIP and Gaming are based on the UDP protocol. One important aspect worth highlighting is the parameter λ of the Poisson process model. Contrary to other models whose parameters remain consistent across all flow types, the value of λ is specific to each type. It is derived from the client data rate. For instance, with an FTP client data rate at 10 Mbps, we would divide 10 Mbps by the product of 8 bits per byte and 1400 bytes per packet to get the appropriate λ value.

Table 3: Traffic Flows Configuration Setup.

	Flow Types				
Parameters	**FTP**	**Streaming**	**HTTP**	**VoIP**	**Gaming**
Data Rate per client Poisson λ per client	10 Mbps 892 (p/s)	5 Mbps 446 (p/s)	1. 5 Mbps 133 (p/s)	128 kbps 50 (p/s)	80 kbps 50 (p/s)
Packet Size	1400 B	1400 B	1400 B	300 B	200 B
Transport layer protocols	TCP			UDP	
Flows type	Long Flow		Short Flow		
Simulation Duration	60 s		10 s		
Clients/Servers	100/5				
Link Bandwidth	2 Gbps				
CSMA Delay	1 ms				
Buffer Size	2000 p				

Finally, in the networking domain, flows are often categorized based on their duration and data volume: long flows and short flows. In our study, FTP serves as an example of a long flow, typically characterized by significant data transfer and numerous packets. On the other hand, HTTP represents short flows, which typically involve fewer packets.

HTTP simulations are designed to emulate short flows, typically characterized by their transient nature. Consider scenarios where a user fetches a webpage along with its associated elements. Consequently, we've setup a short simulation duration of 10s to accurately represent such short flows, mirroring real-world HTTP interactions. Conversely, FTP is used for file transfers, which often involve longer durations and sizeable files. To provide a realistic representation of FTP transfers, we have set up a simulation duration of 60s. While longer simulations may provide a more accurate reflection, they come with computational challenges. Extended simulations inherently demand more computational resources and time. Consequently, a 60s duration was chosen for its balance of realism and feasibility.

To delve deeper into the behavior of these flows, we conducted a series of simulations with each flow type, varying specific parameters to observe their impact on the performance [65]. For instance, for the long flows (FTP), we varied the packet size and observed its impact on the flow's performance. Following this, similar examinations were conducted, but for altering the offered load (client data rate), and then the dAQM drop rate. This systematic approach allowed us to identify the sensitivity and responsiveness of the various AQMs to long flows, in response to changes in different network parameters.

Similar sets of simulations were conducted for the short flows (HTTP). By adjusting the same network parameters packet size, offered load, and dAQM drop rate, we could draw comparisons. This comparative analysis between the reactions of long flows (like FTP) and short flows (like HTTP) to varying conditions provided us with insights into the dynamics and intricacies of network traffic and adaptiveness of the dAQM technique.

5.3 Performance Metrics

In our study, we specifically focused on evaluating the performance of various AQM techniques in response to different traffic models and flow types. Through these evaluations, we sought to understand how each AQM technique reacts and adapts to diverse traffic conditions. Our primary goal was to maximize throughput while achieving minimal latency, in order to provide high QoS to the end-users. Each performance metric, thus, offers a unique perspective on the buffer optimization, network traffic adaptiveness, and robustness of the AQM techniques.

1. End to End Delay: The time taken for a packet to travel from the source to the destination, including all forms of delay like transmission, propagation, queuing, and processing delays.

2. PLR: The ratio (in the form of percentage) of the number of packets that were lost to the total number of packets sent during transmission.

3. Queue Length: The number of packets waiting in a queue to be processed or transmitted (we measured at the switch).

4. Sojourn Time: The total time a packet spends in a queue from the moment it enters until the moment it is transmitted or processed.

5. Throughput: The rate at which data is successfully delivered over a communication link. It accounts for successfully delivered packets and excludes dropped or re-transmitted packets.

6. Flow Completion Time (FCT): The time taken from when the first packet of a flow is sent to when the last acknowledgment or relevant signal indicating completion is received.

6 Performance Analysis

In this section, we will explore the analytical outcomes derived from the simulations by conducting a systematic evaluation of the AQM algorithms against various performance metrics. By interpreting the simulations results, we discuss how they may answer our research questions.

6.1 HTTP CBR Traffic

We begin by evaluating the performance of the five state-of-the-art AQMs (RED, PIE, CoDel, FQ-CoDel, COBALT), the eight standalone dAQM features used individually, and three configurations of the dAQM using all the eight features simultaneously. All AQMs parameters configurations used are described in Table 2. The HTTP flow CBR traffic is described in Table 1 and Table 9. The data has been normalized, with the columns representing the average values of each component in its normalized form relative to the highest value of its kind. These visual results can be seen in Fig. 6a and Fig. 6b. As indicated in the caption, column numbers ranging from 1 to 16 correspond to: RED, PIE, CoDel, FQ-CoDel, COBALT, dAQM$_{f1}$, dAQM$_{f2}$, dAQM$_{f3}$, dAQM$_{f4}$, dAQM$_{f5}$, dAQM$_{f6}$, dAQM$_{f7}$, dAQM$_{f8}$, dAQM$_{c1}$, dAQM$_{c3}$, and dAQM$_{c1}$. Fig. 6a displays the performance of all 16 configurations, whereas Fig. 6b focuses on the 11 configurations related to dAQM only, highlighting the influence of each feature within the dAQM system.

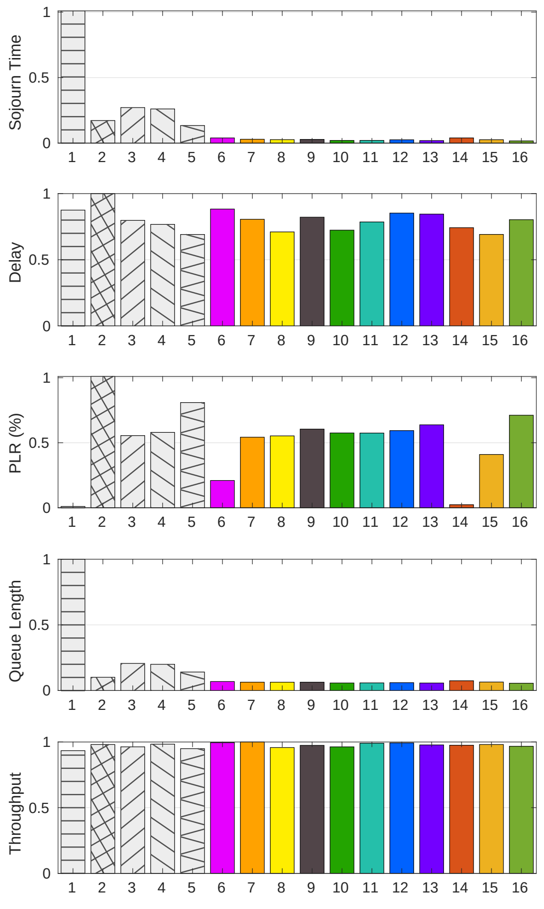
Evaluating Fig. 6a, it is clear that, with the specified parameter settings for the AQM techniques in the HTTP network under CBR conditions: all 11 dAQMs outperformed the other state-of-the-art AQM approaches: RED, PIE, CoDel, FQ-CoDel, and COBALT, in terms of achieving the minimal sojourn time and queue length. Specifically, dAQM$_{c2}$ achieved the lowest delay, on par with COBALT. while dAQM$_{c1}$ attained the lowest PLR, comparable to RED. All AQMs managed to maximize throughput.

6.1.1 Tailoring Guidance for dAQM Configuration

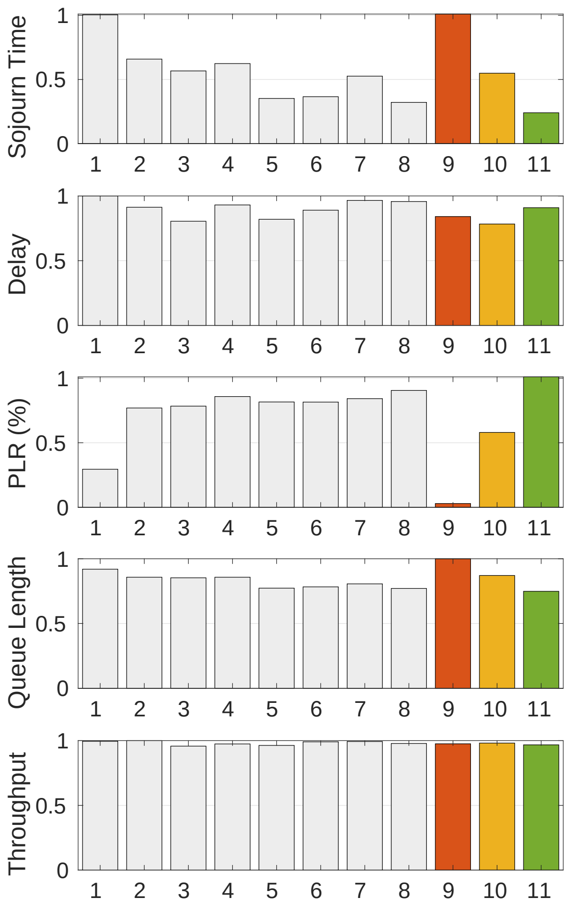
Evaluating Fig. 6b, we can see, dAQM$_{c3}$ obtained the lowest sojourn time and queue length among all configurations, and dAQM$_{c1}$ has obtained the lowest PLR and highest queue length among all, and dAQM$_{c2}$ has obtained the lowest delay, while all 11 configurations managed to maximize their throughput.

Depending on the specific requirements of the QoS for a flow type, users might prioritize certain performance metrics. For instance, for HTTP requests, achieving low delay and high throughput means that web pages can load quickly, enhancing user experience. Furthermore, a low PLR can reduce re-transmissions, which in turn contributes to decreased delay and increased throughput.

As previously mentioned, the metrics prioritized depend on the user's focus. These 8 features can serve as guidance. For instance, when examining the sojourn time sub-figure in Fig. 6b, it is observable that features 5, 6, and 8 (columns 5, 6, and 8 in the histogram) delivered the lowest sojourn time. Where feature 5 refers to the constant queue size, feature 6 refers to the first derivative of buffer size, and feature 8 refers to the third derivative of buffer size. Therefore, if the objective is to further reduce the sojourn time for dAQM$_{c3}$, one could adjust features 5, 6, and 8 by either lowering the threshold 't', increasing the drop rate 'd_r', or augmenting the drop duration 'd_u'. Similarly, now if we are exam-



(a) 16 AQMs configurations.

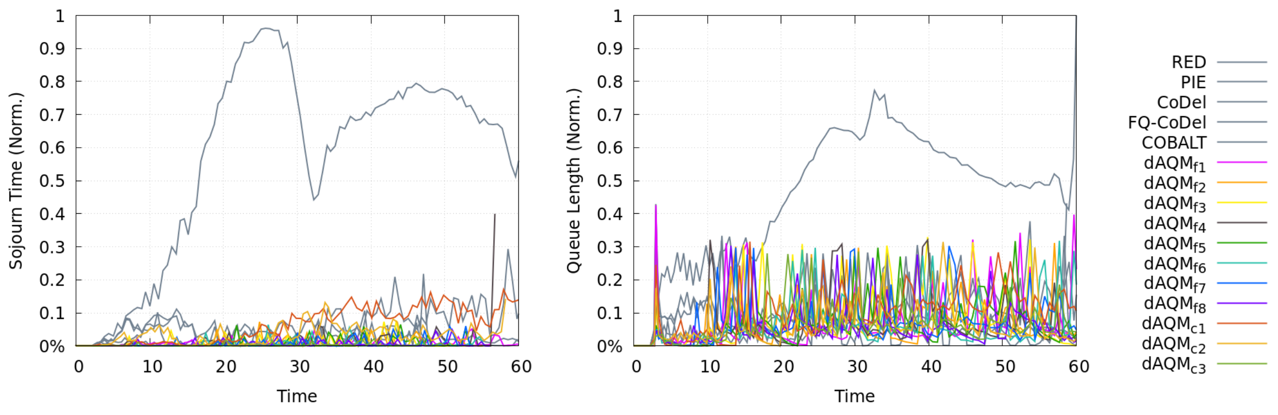


(b) 11 dAQM configurations.

Figure 6: CBR HTTP traffic. (a) Normalised Performance for: 1. RED; 2. PIE; 3. CoDel; 4. FQ-CoDel; 5. COBALT; 6. dAQM$_{f1}$; 7. dAQM$_{f2}$; 8. dAQM$_{f3}$; 9. dAQM$_{f4}$; 10. dAQM$_{f5}$; 11. dAQM$_{f6}$; 12. dAQM$_{f7}$; 13. dAQM$_{f8}$; 14. dAQM$_{c1}$; 15. dAQM$_{c2}$; 16. dAQM$_{c3}$; (b) Viewing results only for the 3 dAQM configurations and the eight features. Normalised Performance for: 1. dAQM$_{f1}$; 2. dAQM$_{f2}$; 3. dAQM$_{f3}$; 4. dAQM$_{f4}$; 5. dAQM$_{f5}$; 6. dAQM$_{f6}$; 7. dAQM$_{f7}$; 8. dAQM$_{f8}$; 9. dAQM$_{c1}$; 10. dAQM$_{c2}$; 11. dAQM$_{c3}$;

ining the PLR sub-figure in Fig. 6b. If the aim is to reduce the PLR for dAQM$_{c1}$ (excluding packets dropped due to a full queue and only considering reducing those dropped via the dAQM mechanism), it is clear that feature 1 (column 1 in the histogram) has the lowest PLR. Thus, one might modify the parameters of feature 1 for dAQM$_{c1}$ by either raising the threshold 't', decreasing the drop rate 'd_r', or reducing the drop duration 'd_u'. These findings provide proof regarding the high programmability and configurability of the dAQM system. The analysis answers the following research question: *What measurable impact do the enhanced programming ability and configurability of the proposed AQM algorithm have on network performance (e.g., maximized throughput, optimized buffer size, minimal delay)?*

Furthermore, Fig. 7a and Fig. 7b illustrates the performance of the AQM techniques from a different



(a) Average Sojourn Time over Time. (b) Average Queue Length over Time. (c) Legend.

Figure 7: CBR Streaming traffic viewed over time with its Performance Normalised.

perspective. This figure displays the average sojourn time and queue length plotted against the simulation time for Streaming CBR traffic. Although it doesn't provide detailed analytical feedback, it allows us to observe how each feature contributes to network changes, aiding our understanding of dAQM. The five state-of-the-art AQMs are greyed out because we want to focus on seeing the changes done by the dAQMs. The worst performance (highest sojourn time and queue length) in both graphs is given by RED. This is because RED does not drop packets until its average queue length calculated is larger than the threshold. This is observable in Fig. 7a and Fig. 7b (0 s to 28 s), packet sojourn time and queue length are only increasing until the RED drop mechanism kicks in. Therefore unlike other AQM algorithms, RED is not actively controlling packet drop at the early stage of congestion.

6.2 Flows under various Traffic Distributions

In this subsection, we will analyse the performance metrics of various Poisson-distributed traffic flows, and the performance of various traffic distribution models for HTTP and Streaming.

6.2.1 Poisson Distributed Traffic

Here, we are evaluating the performance of the AQMs for various traffic flows under the Poisson process distribution model.

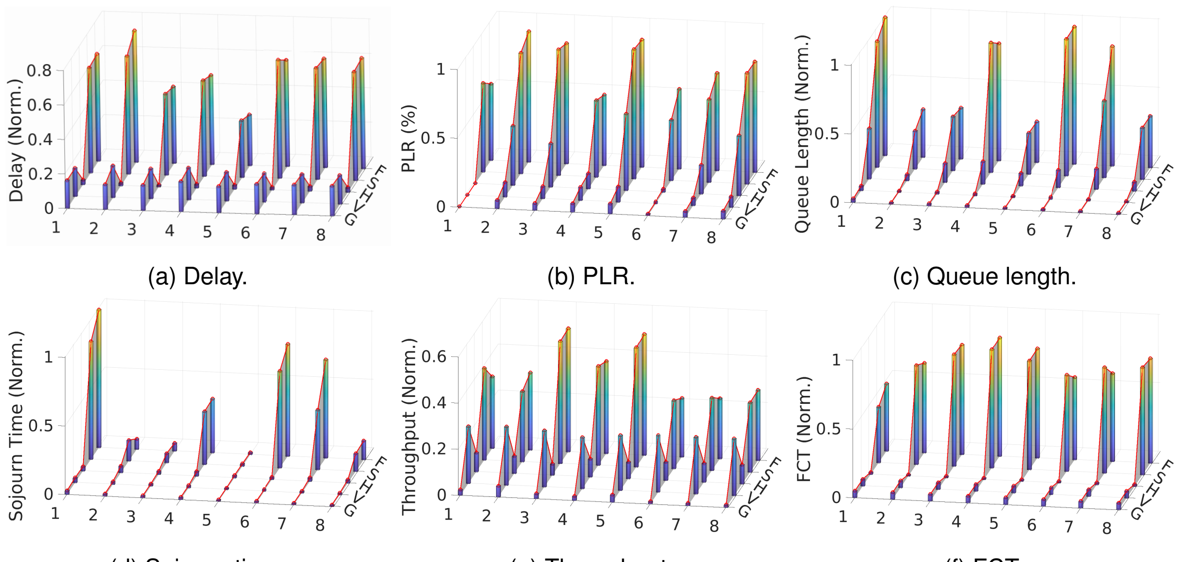
Fig. 8a - Fig. 8f present the performance of various Poisson-distributed flows: FTP (F), Streaming (S), HTTP (H), VoIP (V), and Gaming (G). The data have been normalized to the highest value within each flow type. For visualization, the data sets are plotted in 3D (derived from the five traffic flow types tested across 8 AQMs), allowing for easier comparison.

Analysis:

dAQM$_{c1}$: has the lowest PLR, comparable to both RED and FQ-CoDel. It offers a relatively low FCT, slightly higher than that of RED, but outperformed PIE, CoDel, FQ-CoDel, and COBALT. However, it presents a high queue length and sojourn time, and it has a relatively high delay, outperformed by CoDel, FQ-CoDel, and COBALT, but lower than RED and PIE.

dAQM$_{c2}$: does not stand out in any metric compared to configuration 1 (dAQM$_{c1}$) or configuration 3 (dAQM$_{c3}$). It has a relatively low PLR, better than PIE, CoDel, and COBALT, and it has a relatively low FCT, better than PIE, CoDel, FQ-CoDel, and COBALT. However, its delay, queue length, and sojourn time, are relatively high.

dAQM$_{c3}$: presents the lowest queue length comparable to PIE, CoDel, and COBALT. It has a very



(a) Delay.
(b) PLR.
(c) Queue length.
(d) Sojourn time.
(e) Throughput.
(f) FCT.

Figure 8: Various Poisson distributed traffic flows. Performance for: 1. RED; 2. PIE; 3. CoDel; 4. FQ-CoDel; 5. COBALT; 6. dAQM$_{c1}$; 7. dAQM$_{c2}$; 8. dAQM$_{c3}$

low sojourn time comparable to PIE and CoDel, but not as low as COBALT, yet, it outperforms RED and FQ-CoDel. In terms of lower FCT, it surpasses CoDel, FQ-CoDel, and COBALT. Its PLR is more favorable than that of PIE, CoDel, and COBALT. However, it has a relatively high delay, outperformed by CoDel, FQ-CoDel, and COBALT, but lower than RED and PIE.

Across all dAQM configurations, there is a noticeable reduction in throughput for heavy traffic and long flows traffic (FTP, Streaming). It is not ideal, because we aim to maximize the throughput and obtain a minimal delay. Under short flows traffic like HTTP, VoIP, and Gaming, the throughput is more consistent across all AQM techniques.

RED: has the lowest FCT, lowest PLR together with FQ-CoDel and dAQM$_{c3}$, but very high delay, highest queue length and sojourn time, and also reduced throughput for heavy load traffic.

PIE: has the lowest queue length together with CoDel, COBALT, dAQM$_{c3}$, low sojourn time, and relatively low FCT, however, the highest delay, highest PLR together with CoDel and COBALT, and reduced throughput for heavy traffic.

CoDel: has the lowest queue length together with PIE, COBALT, dAQM$_{c3}$, low sojourn time, highest throughput, relatively low delay, however very high FCT.

FQ-CoDel: has the lowest PLR together with dAQM$_{c1}$ and RED, relatively low delay, relatively high throughput, very high queue length, and a little high sojourn time, and the highest FCT.

COBALT: has the lowest delay, lowest queue length with serval others, lowest sojourn time, and high throughput, but highest PLR, and very high FCT.

Discussion:

The analysis shows dAQMs have a reduced throughput for long-flow traffic and a more consistent throughput for short-flow traffic. However, dAQM$_{c3}$ performed very well in obtaining a small queue length and sojourn time. A small queue length can be very beneficial, it means that less buffer space is needed to hold the packets. Small buffers use less physical memory, thus saving resources. It can also decrease buffer bloat issues because smaller buffer space will be used. A smaller sojourn time means the packet does not spend much time waiting in the queue, indicating the network is less likely to experience congestion.

Effectiveness of dAQM in TCP and UDP Protocols

The analysis shows that the dAQM is more effective in varying the PDP of the TCP-connected traffic. The observations indicate that the dAQM algorithm is more efficient when employed with the TCP as opposed to the UDP. The primary reason for this performance disparity lies in the dAQM's reliance on sojourn time and queue length variations. Higher variability in sojourn time, or queue length, results in larger derivative ratios, which in turn enhances the performance of the dAQM algorithm. TCP, with its inherent re-transmission and drop mechanisms, tends to exhibit greater sojourn time variability, making it an ideal fit for dAQM. Conversely, UDP, being connection-less and lacking native re-transmission or drop mechanisms, tends to maintain a more consistent sojourn time and queue lengths. This consistency results in lower derivative ratios, thereby reducing the chances of triggering the dAQM's drop conditions. Partially answering the third research question: *How would the dAQM* *perform under different transportation layer protocols (TCP & UDP)?*

The impact of the use of TCP or UDP on the performance of dAQM is mainly expressed in terms of PLR, sojourn time, and queue length. The impact on the throughput, delay, or FCT, are more related to the traffic patterns. Whether it is short flow or long flow, whether it is light or burstiness and heavily loaded. Long flows traffic, which often is also heavily loaded and expresses burstiness, does not have outstanding performance with dAQM for throughput, delay, or FCT; whereas, for short flows traffic, which often are light and short-lived, has better performance for throughput, delay, or FCT with dAQM.

6.2.2 Various Distribution Models

In this subsection, we assess the performance of the AQMs tested against various traffic distribution models, including CBR (C), VBR (V), Poisson (Po), Pareto (Pa), and Weibull (W). Specifically, our evaluation focuses on the performance difference between short flows traffic (HTTP) and long flows traffic (Streaming).

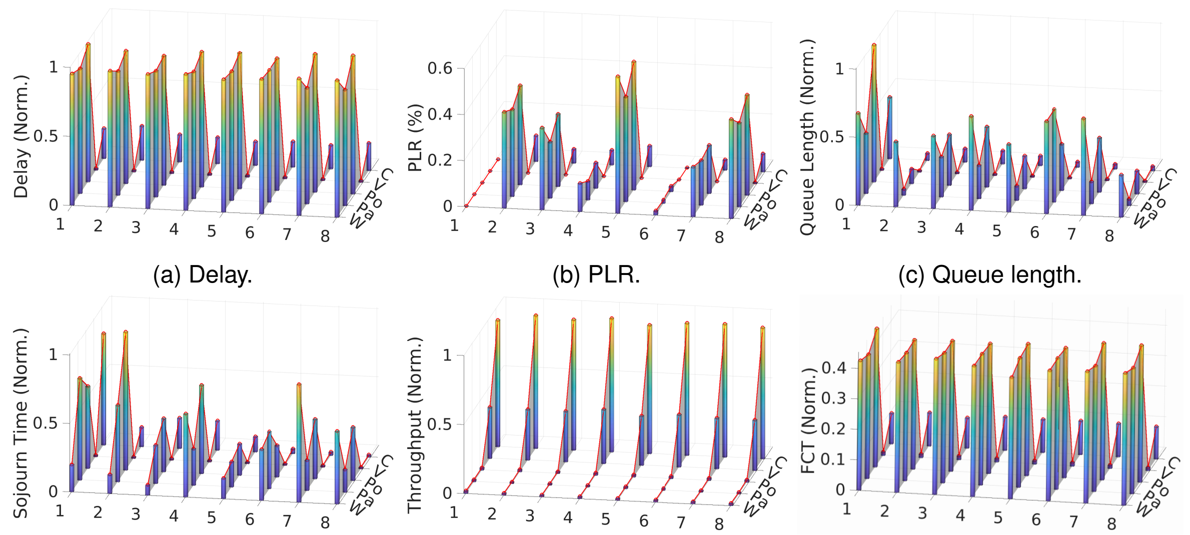
Fig. 9a - Fig. 9f present the performance of various traffic distribution models for HTTP, Fig. 10a - Fig. 10f present the results for Streaming. The data have been normalized to the highest value within each model. For visualization, the data sets are plotted in 3D, derived from the five traffic models tested across 8 AQMs. This format allows for easier comparison.

Various Distribution Models for HTTP Traffic
Analysis of Fig. 9a - Fig. 9f :

Since HTTP is using TCP and it is short flow, the delay, throughput, and FCT performance align among all AQMs. Therefore, we would only discuss the performance difference of PLR, queue length, and sojourn time.

dAQM$_{c1}$: has the lowest PLR, comparable to RED. It has a relatively low sojourn time, on par with CoDel, slightly higher than COBALT, but outperforms RED, PIE, and FQ-CoDel. It has a queue length lower than RED, but higher than PIE and COBALT, roughly equivalent to the queue lengths of CoDel and FQ-CoDel.

dAQM$_{c2}$: has a relatively low queue length, comparable to CoDel and FQ-CoDel, outperformed RED, but slightly worse than PIE and COBALT. It has a relatively high sojourn time, comparable to FQ-CoDel, lower than RED and PIE, but higher than CoDel and COBALT. It has a relatively average PLR, lower than PIE, CoDel, and COBALT, but higher than RED and FQ-CODel.



(a) Delay. (b) PLR. (c) Queue length.

(d) Sojourn time. (e) Throughput. (f) FCT.

Figure 9: Various traffic distribution models for HTTP. Performance for: 1. RED; 2. PIE; 3. CoDel; 4. FQ-CoDel; 5. COBALT; 6. dAQM$_{c1}$; 7. dAQM$_{c2}$; 8. dAQM$_{c3}$

dAQM$_{c3}$: has the lowest queue length. It has a relatively low sojourn time, on par with CoDel, higher than COBALT, but surpassing RED, PIE, and FQ-CoDel. It has a relatively high PLR, comparable to CoDel, and slightly below COBALT.

RED: has the lowest PLR, dAQM$_{c1}$ is comparable to RED in obtaining the lowest PLR. However, RED has the highest queue length, especially in Poisson-distributed flow, about 80% higher than dAQM$_{c3}$, and it has the highest sojourn time under CBR traffic and Pareto-distributed traffic.

PIE: has a relatively high PLR, on par with dAQM$_{c3}$, but its queue length is higher than dAQM$_{c3}$, especially in Weibull-distributed traffic, and the sojourn time is the highest in Poisson-distributed traffic.

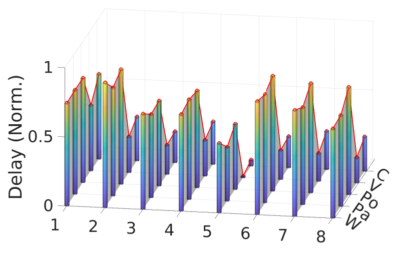
CoDel: has a relatively high PLR, higher than dAQM$_{c1}$ and dAQM$_{c2}$ but lower than dAQM$_{c3}$. The sojourn time is comparable to dAQM$_{c3}$, however, its queue length is a lot higher than dAQM$_{c3}$ across various distribution models.

FQ-CoDel: has a relatively low PLR, although not as low as RED and dAQM$_{c1}$. It has a relatively high queue length and sojourn time among all traffic distribution models.

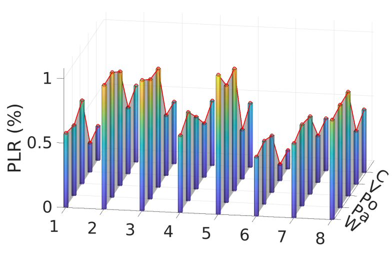
COBALT: although, has the lowest sojourn time, and a relatively low queue length. dAQM$_{c3}$ provides lower queue length under Weibull, Poisson, and CBR traffic. COBALT also has the worst PLR among all distributions, about 60% higher than dAQM$_{c1}$.

The analysis shows the configurability of dAQM to provide low PLR, or low queue length and sojourn time, under various traffic distribution models for short flow.

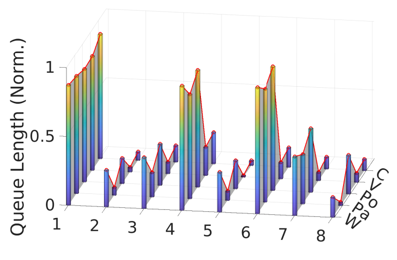
Various Distribution Models for Streaming Traffic
Analysis of Fig. 10a - Fig. 10f :



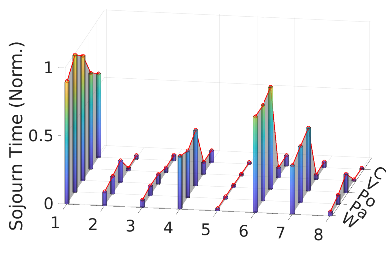
(a) Delay.



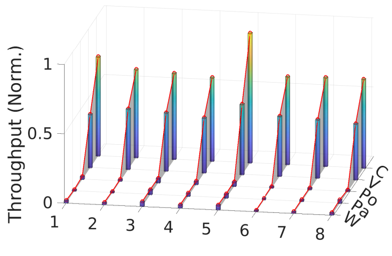
(b) PLR.



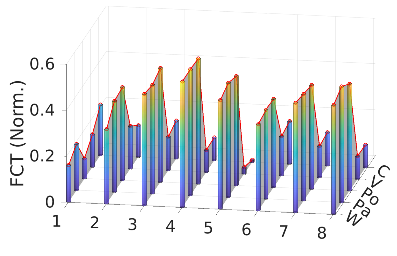
(c) Queue length.



(d) Sojourn time.



(e) Throughput.



(f) FCT.

Figure 10: Various traffic distribution models for Streaming. Performance for: 1. RED; 2. PIE; 3. CoDel; 4. FQ-CoDel; 5. COBALT; 6. dAQM$_{c1}$; 7. dAQM$_{c2}$; 8. dAQM$_{c3}$

dAQM$_{c1}$: has the lowest PLR and outperforms all other state-of-the-art AQMs among all distributions. It has a relatively low FCT, which is higher than RED, on par with PIE, but outperforms CoDel, FQ-CoDel, and COBALT. Its throughput is comparable to other state-of-the-art AQMs, except COBALT. It has a relatively high queue length, it matches FQ-CoDel in Weibull, Pareto, and Poisson traffic but shows better performance under CBR and VBR traffic, and it fully outperforms RED. It has a relatively high sojourn time, although lower than RED. It has a higher delay than CoDel, FQ-CoDel, and COBALT, comparable delay to RED and PIE.

dAQM$_{c2}$: does not stand out in any metric compared to dAQM$_{c1}$ or dAQM$_{c3}$. However, compared to the state-of-the-art AQMs, it has a relatively low PLR, comparable to RED in Weibull, Prato, and Poisson distributed traffics, and outperforms PIE, CoDel, FQ-CoDel and COBALT (COBALT is only slightly worse than dAQM$_{c2}$) among all distributions. It has a relatively low queue length and outperforms RED and FQ-CoDel, slightly worse than CoDel. It has a relatively high sojourn time, comparable to FQ-CoDel, but outperforms RED. Its delay is relatively high, comparable to RED, PIE, and outperformed by CoDel, FQ-CoDel, and COBALT. It has a relatively high FCT lower than CoDel, FQ-CoDel, and COBALT, but higher than RED and PIE. The throughput aligns with other AQMs except for COBALT.

dAQM$_{c3}$: has the smallest queue length under Weibull and Pareto distributions, outperforms all other AQMs, and, still a very low queue length under Poisson, CBR and VBR traffic, although not as good as PIE and COBALT. It has a relatively low sojourn time, on par with PIE, CoDel, higher than COBALT, but surpassing RED, and FQ-CoDel. It has a relatively low PLR, higher than RED and FQ-CoDel, but outperforms PIE, CoDel, and COBALT. Its throughput is comparable to other state-of-the-art AQMs except for COBALT. Has FCT, higher than RED and PIE, outperforms CoDel, FQ-CoDel, and COBALT. Has a higher delay than CoDel, FQ-CoDel, and COBALT.

RED: RED has the highest queue length among all distributions. It is more than 70% worse than dAQM$_{c3}$ under the Weibull distribution, about 85% worse under the Pareto distribution and VBR, and about 80% worst in CBR traffic.

PIE: has a relatively low FCT, comparable to dAQM$_{c1}$. It has the lowest queue length under Poisson distributed traffic. It has a very low sojourn time but dAQM$_{c3}$ has a lower. It has a relatively high delay, similar to the dAQMs and RED. However, it has the highest PLR together with CoDel and COBALT, and the throughput is aligned with other AQMs except for COBALT.

CoDel: has a low queue length but not as low as dAQM$_{c3}$. It has a low sojourn time, comparable to dAQM$_{c3}$. It has a relatively lower delay. However, it has a relatively high FCT, higher than the dAQMs. It has the highest PLR together with PIE and COBALT, and its throughput aligns with other AQMs besides COBALT.

FQ-CoDel: has a relatively low PLR, comparable to dAQM$_{c2}$, but dAQM$_{c1}$ has much lower. It has a very high queue length, more than 50% higher than dAQM$_{c2}$ and 80% higher than dAQM$_{c3}$ under Weibull and Pareto distributions, although it is comparable to dAQM$_{c1}$ under Weibull, Pareto and Poisson distributions, dAQM$_{c1}$ has much lower queue length under CBR and VBR. Its sojourn time is lower than dAQM$_{c1}$, comparable to dAQM$_{c2}$ but much higher than dAQM$_{c3}$.

COBALT: has the maximum throughput, especially under CBR traffic. It has a relatively low queue length, it is the lowest under VBR and CBR traffic, and it has the lowest sojourn time. It also has a relatively lower delay. However, it has a relatively high FCT, higher than dAQM$_{c1}$, and it has the highest PLR together with PIE and CoDel, across various distributions.

Discussion:

In this discussion, we aim to answer the research question: *How does the performance of dAQM algorithm compare to traditional AQMs in handling variable network flows?*

The analysis of Fig. 9d shows, dAQM$_{c3}$ outperformed all other state-of-the-art AQMs in achieving the lowest sojourn time for CBR HTTP traffic. However, it fell behind RED, PIE, CoDel, and COBALT in managing Weibull-distributed HTTP traffic. Interestingly, dAQM$_{c3}$ surpassed RED, PIE, CoDel, and FQ-CoDel in Pareto distributed HTTP traffic, with a sojourn time aligning closely to COBALT's. For Poisson-distributed HTTP traffic, as demonstrated in our earlier figures, dAQM$_{c3}$ surpassed the performance of RED, PIE, and FQ-CoDel. It was on par with CoDel, but less efficient than COBALT. With VBR HTTP traffic, where congestion isn't a significant factor, the performance metrics for all AQMs were roughly similar. However, in Fig. 10d, dAQM$_{c3}$ stands out, almost leading the way across all traffic models for the shortest sojourn time. It is comparable to PIE and CoDel, slightly inferior to COBALT, but clearly surpasses RED and FQ-CoDel.

The analysis shows that while both HTTP and Streaming were connected via the TCP protocol, the AQMs give different performances under the same traffic distributions. The performances of AQMs are also different when applied to different traffic distribution models for the same flow type. Their behaviors are significantly influenced by traffic patterns. This partially reflects the analysis outcome we found for Poisson distributed traffic (in subsection 6.2.1). Recall that we found the use of the transportation layer protocols (TCP & UDP) on the performance of dAQM mainly affects PLR, sojourn time, and queue length. While the impact on the throughput, delay, or FCT, are more related to the traffic patterns (short or long flow, light or heavy load). In this subsection, we found that the throughput, delay, and FCT, are also highly affected by the flow types or distribution models.

COBALT was the only one whose throughput was affected by flow type, in HTTP (short and light flow) its performance was comparable to the others, but in streaming (long and heavy flow) it stands out. The delay of all AQMs was roughly the same in HTTP (short and light flow), but RED gives the worst performance in Streaming (long and heavy flow), especially under CBR and VBR flow, while CoDel gives lower delay and better performance, then COBALT gives the best performance. This difference is due to how each AQM mechanism reacts to congestion. For example, RED focuses on managing the average queue size, so it will start dropping packets with a probability when the queue size exceeds some threshold. Since RED's PLR is low, it means that for most of the time, the queue size was within its threshold, thus dropping fewer packets leads to less retransmission, allowing it to obtain the lowest FCT. However, when fewer packets are dropped, they stay longer in the queue, increasing the sojourn time, and leading to higher end-to-end delay. While CoDel focused on keeping the minimal sojourn time, it was proactively dropping more packets. Thus, it had higher PLR but lower delay, and at the same time, more retransmission of dropped packets leads to higher FCT.

Traffic models, in essence, mirror real-world scenarios, revealing the adaptability and resilience of these AQMs. The ability of an AQM to efficiently manage traffic and deliver optimized sojourn times is inherently tied to the type of traffic flow it is subjected to. For instance, in Weibull distributed HTTP traffic, certain AQMs like RED, PIE, CoDel, and COBALT exhibited better adaptability compared to dAQM. Conversely, under Pareto distribution, which can be associated with 'bursty' traffic patterns, dAQM3 showcased its superiority, indicating its efficiency in managing occasional high-traffic bursts. As highlighted from our observations, the dAQM3 demonstrates proficiency with specific traffic distributions but encounters challenges with the Weibull distribution. Yet, when it comes to the flow type streaming, the landscape shifts. The consistent performance of dAQM3 across various traffic models for streaming, particularly in terms of minimal sojourn time, underscores its robustness. However, the shift in its rank when transitioning from one flow to another, and from one model to another signifies the need for AQMs to be contextually deployed based on anticipated traffic patterns. Under idle-like HTTP VBR traffic, the choices of AQMs may not be apparent. Yet, as the traffic flow intensity increases, it becomes more unpredictable, and the performance gains of the AQMs would be more apparent. This emphasized the importance of selecting the appropriate AQM for the specific traffic environment. Implementing dAQM with rich traffic features, high programmability, and configurability would be very beneficial for maintaining network stability and congestion control for variable network conditions.

6.3 Short and Long Flows under Varying Network Parameters

When we look at different types of network traffic, like short flows (HTTP) and long flows (FTP), the AQMs behave differently under various conditions. The differential nature of these flows suggests that certain network parameters may influence the AQMs differently. This section aims to identify the optimal parameter configurations of dAQM and understand how variation in the parameters influences the performance of dAQM.

•**Varying dAQM drop rate**: varying the drop rates of dAQM from a minimal 1% to a substantial 99%.

•**Varying Packet Size**: varying the packet size from a small 50 B to a more substantial 1400 B.

•**Varying (Offered) Load**: varying the transmission rate per client from 5 kbps to 4 Mbps for HTTP; and from 50 kbps to 10 Mbps for FTP.

All displayed figures are normalized against the highest value within their respective group and are presented as percentages. For instance, each y value will be divided by its y_{max} in its kind, and multiplied by 100 (%). Meaning, that if m's delay is 40%, and n's is 100% (the maximum), the delay of m is 40% of n's. The figures present results plotted using straight lines connecting the original data points, and their performance statistics at their maximum values are given in the form of tables in Appendix A. Furthermore, we aim to answer the other half of the third research question in the subsection, *How would the dAQM perform under different flow types (short flow vs long flow), and variations in network parameters (dAQM drop rate, load, packet size)?*

6.3.1 Drop Rate Variation

Fig. 11a to Fig. 11f and Fig. 12a to Fig. 12f illustrate the variations in performance resulting from changes in the dAQM drop rate variable, ‘d_r’, while all other settings remain unchanged. Since the drop rate variable is specific to the dAQM algorithm, only the dAQM will undergo changes. Consequently, the state-of-the-art AQMs will appear as constant lines in the figures. Specifically, Fig. 11a to Fig. 11f represent HTTP CBR traffic (for short flows), while Fig. 12a to Fig. 12f represent FTP CBR traffic (for long flows).

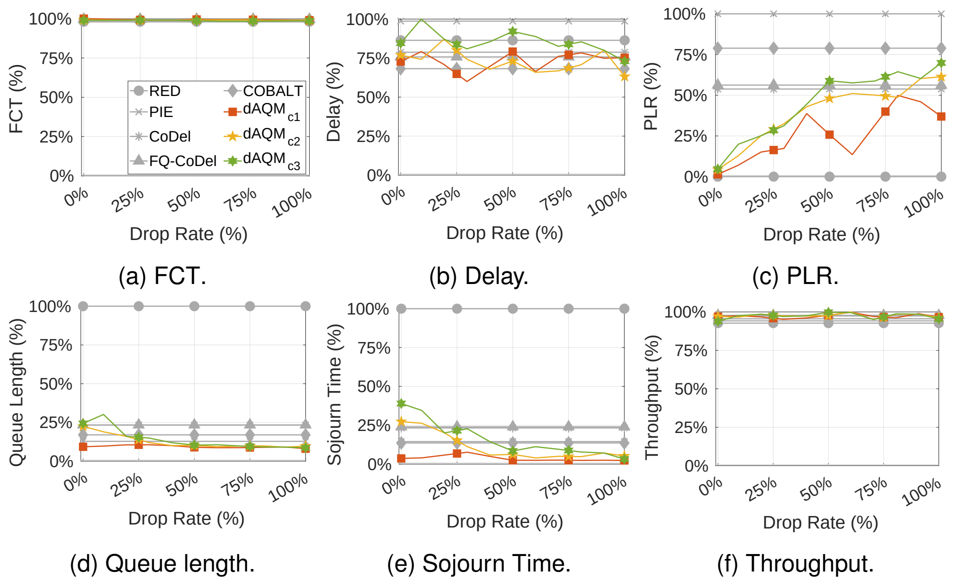


Figure 11: Performance of dAQM by increasing the drop rate for HTTP.

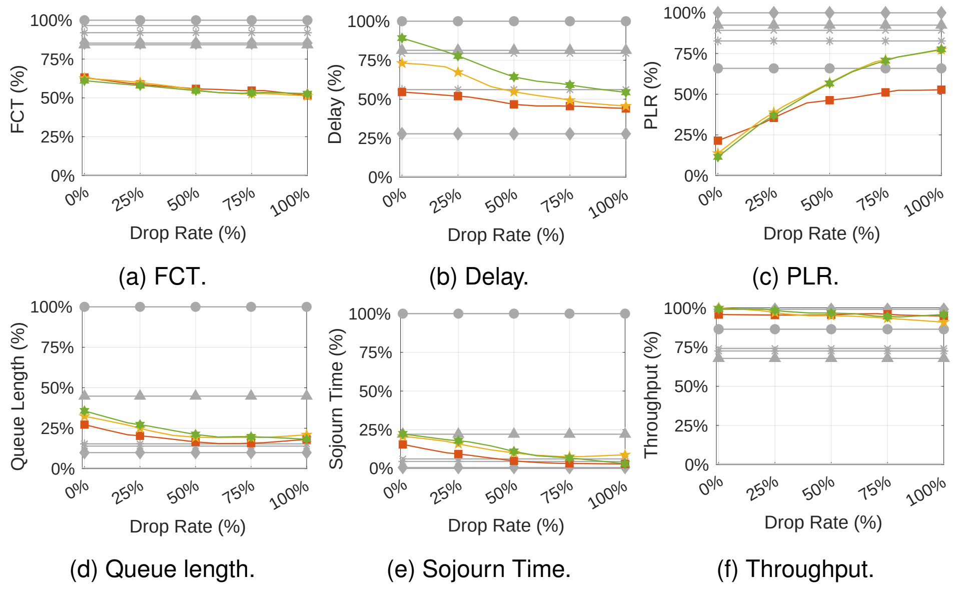


Figure 12: Performance of dAQM by increasing the drop rate for FTP.

Analysis and Discussion:

• **PLR, Queue Length, Sojourn Time**: As the drop rate variable 'd_r' increases, the PDP correspondingly rises, leading to an increase in the PLR. Concurrently, both queue length and sojourn time decrease.

In terms of PLR, dAQM$_{c1}$ is the lowest in both scenarios, over 60% better than the worst performance given by PIE in the short flows; and more than 45% better than the worst performance given by COBALT in the long flow. The performance of dAQM$_{c2}$, and dAQM$_{c3}$ are better than PIE and COBALT, while worse than RED, CoDel, and FQ-CoDel; however, dAQM$_{c2}$, and dAQM$_{c3}$ are still over the average in the long flows, outperforming COBALT, FQ-CoDel, CoDel and PIE. RED is an exception in the short flows because RED only starts dropping packets when the average queue length exceeds its threshold. The PLR of RED is zero in the short flows scenario indicating that no packets are dropped, its average queue length did not exceed the threshold.

The queue length of dAQM$_{c1}$, dAQM$_{c2}$, and dAQM$_{c3}$ are the lowest in short flows, over 90% better than the worst performance given by RED. However, in the long flows, although their queue length trend decreases with increasing dAQM drop rate, dAQMs were still slightly outperformed by COBALT, PIE, and CoDel. A similar outcome was found early in CBR streaming (long flow) traffic, the queue length of dAQMs is higher than PIE, CoDel, and COBALT. This finding might be specific to the traffic type applied, since for Streaming traffic under the Pareto distribution, dAQM$_{c3}$ obtained the minimal queue length.

The sojourn time of dAQM$_{c1}$, dAQM$_{c2}$, and dAQM$_{c3}$ are the lowest in short flows as well as in the long flows. They are about 95% and 90% to 97% better than the worst performance given by RED for the short and long flows, accordingly.

• **Throughput**: The throughput of dAQMs is maximized in the short flows, all AQMs have obtained the same level of throughput. However, in the long flows, dAQMs had a reduced throughput in its trend with increasing dAQM's drop rate. At its maximum drop rate, dAQMs have outperformed RED, PIE, CoDel, and FQ-CoDel, with dAQM$_{c1}$ and dAQM$_{c3}$ being over 40% better than the worst performance given by FQ-CoDel, however, slightly outperformed COBALT. A similar outcome was found early with streaming CBR traffic (in Section 6.2.2). COBALT is the only technique whose throughput performance is affected by the traffic patterns, it has an increased throughput with long flow traffics compared to other AQMs.

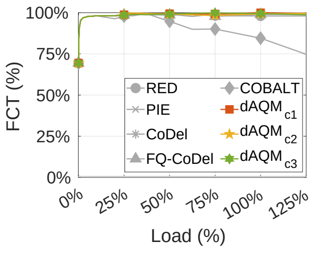
• **FCT**: In short flows, the FCTs for all AQMs are roughly equivalent. This finding corresponds to the outcome of the previous analysis on Poisson Distributed Traffic (in Section 6.2.1) and HTTP CBR traffic (in Section 6.2.2). Where the FCTs are roughly the same in the short flows, for Poisson distributed Gaming, VoIP, and HTTP traffic, and for HTTP flow under various traffic models (CBR, VBR, Poisson, Pareto, Weibull).

However, in the long flows, dAQMs obtained the lowest FCT across all AQMs, giving performance about 50% better than the worst AQM, RED. This discovery adds a new dimension to our previous findings. Previously, in the analysis for streaming (long flow) under various traffic models. We found that dAQM$_{c3}$ gives the highest queue length and sojourn time for Poisson distributed streaming among the many distribution models while comparing to itself. Therefore, we may assume, dAQM is less adaptive to exponentially increasing traffic as to other traffic categories. Thus, it is explainable that although dAQM$_{c3}$ wasn't giving the lowest FCT for Poisson distributed FTP, it is still convincing that dAQM$_{c3}$ can provide the best performance in obtaining the lowest FCT under CBR long flow traffic. Since, in the streaming CBR, it was also having a relatively low FCT. Even though, COBALT had a lower FCT for streaming CBR, it has a much higher PLR, and FTP is more heavily loaded than streaming, therefore, we found COBALT had a larger difference in the PLR compared to dAQM$_{c3}$ for FTP. Higher PLR leads to more retransmissions and thus higher FCT. The heavier the load in long flow for CBR, the greater the difference between dAQM obtaining the lowest FCT compared to the other AQMs.

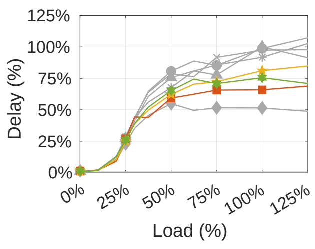
• **Delay**: The trend of the delay is decreasing while dAQM's drop rate is increasing. This is reasonable, given that increasing the PDP leads to lower sojourn time and queue length, hence lowering the overall delay. This observation can be seen in both short and long flows. In the short flows, dAQM$_{c2}$ has obtained the lowest delay together with COBALT, being more than 35% effective than the worst AQM PIE. In the long flows, the best performance was given by COBALT, though followed by dAQM$_{c1}$, being about 55% effective than the worst performance given by RED.

6.3.2 Offered Load Variation

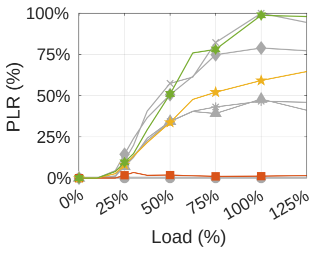
In this subsection, we focus on comparing the changes in dAQM performance against other state-of-the-art AQMs when modifying the offered load (client transmission rate), with all other network conditions remaining unchanged. Specifically, Fig. 13a to Fig. 13f represents HTTP CBR traffic (for short flows), while Fig. 14a to Fig. 14f represents FTP CBR traffic (for long flows).



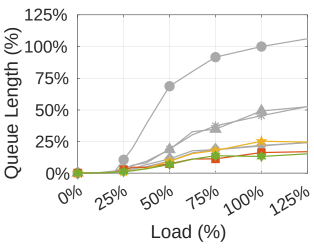
(a) FCT.



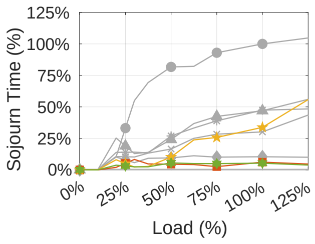
(b) Delay.



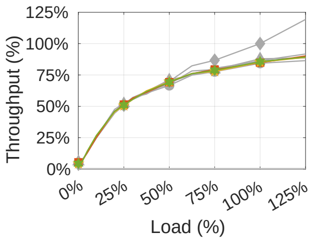
(c) PLR.



(d) Queue length.

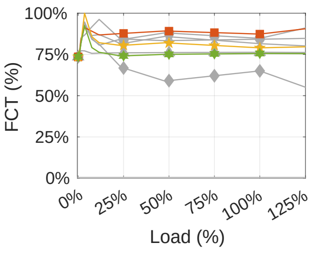


(e) Sojourn Time.

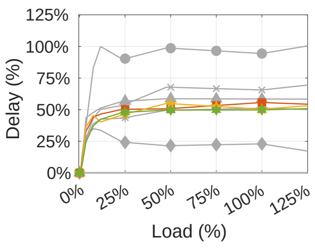


(f) Throughput.

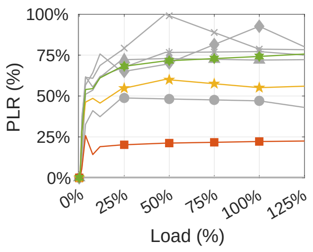
Figure 13: Performance of dAQM by increasing the load for HTTP.



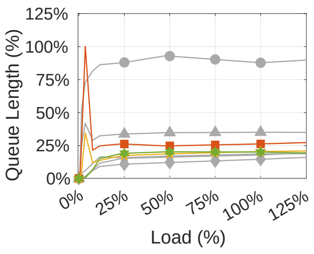
(a) FCT.



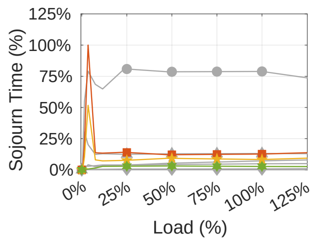
(b) Delay.



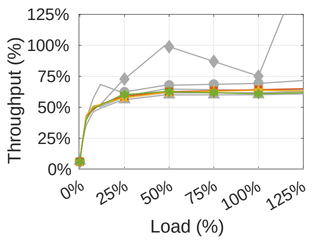
(c) PLR.



(d) Queue length.



(e) Sojourn Time.



(f) Throughput.

Figure 14: Performance of dAQM by increasing the load for FTP.

Analysis and Discussion:

Before delving into a comparison of the results presented in the two sets of figures, it's essential to address the *observed peaks within the initial 10% range* of the FTP load. Several factors can account for this observation.

There are peaks for all the AQMs in FTP, however, the most obvious peaks are with the dAQM algorithms for dAQM$_{c1}$ and dAQM$_{c2}$, especially with queue length (Fig. 14d) and sojourn time (Fig. 14e). This is related to both the TCP protocol used, as well as dAQM's higher-order derivatives configurations. TCP congestion control mechanism uses a scheme called Congestion Window (CWMD). It controls how much data can be sent by the sender to avoid overloading the link. The CWMD grows (exponentially at TCP's slow start phase, additive at TCP's congestion avoidance phase) based on the acknowledgment received, until it finds the equilibrium at which rate the sender should be sending data and the network can handle it. At a low transmission rate, new packets are coming to the network at a low rate, it takes a longer time to receive new acknowledgments, and thus, the CWMD would grow more slowly because TCP is more conservative in increasing its congestion window. At a high transmission rate, the CWMD would grow more quickly, because the acknowledgments are returned faster increasing the rate for the window to grow. At lower loads, the network is more slowly utilized, and thus also spends more time in the slow start phase, allowing windows to grow large enough to fill up the queues before it enters the congestion avoidance phase. Thus, increasing the queue length and sojourn time.

Additionally, the clients were not started simultaneously in the CBR traffic, as in the Poisson, Pareto, or Weibull distributed traffic. Therefore, peaks were not observed for HTTP traffic even at low loads, because for its 10s simulation, fewer clients were started, as for 60s FTP traffic there are six times more clients than HTTP traffic by the end of the simulation. The effect of TCP slow start and congestion avoidance therefore brought a more pronounced effect (peak) to the simulation.

We would discuss the specifics of the high peaks provided by dAQM$_{c1}$ and dAQM$_{c2}$ while evaluating their performance.

• **Throughput**: the throughput of dAQM is roughly the same across all AQMs in both scenarios, except that COBALT gives a slightly higher throughput.
The plot (Fig. 14f) shows there is a peak at COBALT throughput at 50% load. Then, it slowly decreases after 50% load. The maximum load applied is 10 Mbps, the half of it is 5 Mbps, which corresponds to the load parameter we set for the streaming. This finding recalls (Fig. 10e) our previous finding, that COBALT has increased performance in throughput when transitioning from HTTP (short flow) to Streaming (long flow). From the analysis (Fig. 14f), our understanding at that time was not comprehensive. COBALT does increase in performance as traffic intensity gets increases, but after achieving half the load (5 Mbps), its performance will slowly drop until it ends up almost on par with other AQM algorithms (10 Mbps). Therefore, we may conclude that COBALT might be more suitable for network conditions similar to our streaming configuration, however, at heavier load long flows traffic, dAQM remains to be very competitive.

• **FCT**: For short flows, the FCTs of all AQMs are approximately the same, with the exception of COBALT, which demonstrates a performance that is more than 15% better (at 100% load) in achieving a lower FCT compared to other AQMs. In the long flows, dAQM$_{c3}$ gives the second-lowest FCT, on par with FQ-CoDel. It outperforms RED by approximately 10%, whereas COBALT surpasses dAQM$_{c3}$ by more than 10%.

• **Queue Length, Sojourn Time**: Across both scenarios, dAQMs have effectively achieved the lowest queue lengths and sojourn times. Specifically, in short flow, dAQM$_{c1}$ and dAQM$_{c3}$ perform about 85% better in terms of obtaining the lowest queue length, and about 95% better in obtaining the lowest sojourn time, compared to the poorest performance given by RED. In long flow, dAQMs and several other state-of-the-art AQMs produced queue lengths within a similar range, with COBALT obtaining slightly lower queue length and sojourn time. dAQM$_{c2}$ and dAQM$_{c3}$ gives more than 75% smaller queue length, and dAQM$_{c3}$ gives more than 95% smaller sojourn time than the worst performance given by RED.

• **PLR**: In both scenarios, dAQM$_{c1}$ gives the lowest PLR. Specifically, in short flows, it outperforms the least effective AQM, PIE, by about 99%. In long flows, dAQM$_{c1}$ performs more than 75% better than the worst performance given by COBALT. In contrast, dAQM$_{c3}$ had the highest PLR, on par with PIE in the short flows and comparable to those AQMs outperformed by dAQM$_{c1}$ in long flows.

• **Delay**: The delay given by the dAQMs are relatively good, surpassing all other AQMs with the exception of COBALT. Specifically, in the short flow, COBALT outperforms dAQM$_{c1}$ by approximately 20%, whereas dAQM$_{c1}$ exhibits a performance that is around 35% superior to the highest delay given by RED, and FQ-CoDel. In long flows, COBALT is about 48% more efficient than dAQM$_{c3}$, while dAQM$_{c3}$ is about 50% better than RED.

Based on the analysis, we gain a better understanding of peaked queue length and sojourn time for dAQMs. Fig. a shows the queue length reaches a peak for dAQM$_{c1}$, followed by FQ-CoDel, and then dAQM$_{c2}$, while AQMs like RED, PIE, CoDel, COBALT, and dAQM$_{c3}$ do not have such a noticeable peak in the queue length. Therefore, we may assume that peaks in dAQM$_{c1}$ and dAQM$_{c2}$ are caused by the parameter configurations. The three configurations are set up with different focuses to understand the optimization of dAQM parameters configuration, in queue management and congestion control. For example, dAQM$_{c3}$ was set up with the aim of minimal queue length and sojourn time, and dAQM$_{c1}$ was aiming at minimal PLR. The drop rate of dAQM$_{c3}$ is set up to 98% upon detection of congestion, whereas the drop rate for dAQM$_{c2}$ is 50%, and for dAQM$_{c1}$ is 5%. The threshold parameters of dAQM$_{c3}$ are also set smaller for the eight traffic features, allowing it to trigger the drop mechanism at a much earlier stage. For these reasons, it is much more effective in queue management. We also suggest that the peaks may have become pronounced by the use of TCP. As previously, discussed the reasons for peak occurrences are related to TCP's slow start. The slow start phase is longer with lower loads, allowing the queue to grow, like in the case for dAQM$_{c1}$. dAQM$_{c1}$ has higher thresholds, making it harder to satisfy the drop conditions. Even at its dropping phase, it has only 5% chance of dropping the packets, making it less efficient in maintaining the queue. However, dAQM$_{c3}$ is efficient enough to handle the congestion by itself, while maintaining a high level of throughput and a relatively low delay, without relying on the drop mechanism from TCP's algorithms.

6.3.3 Packet Size Variation

In this subsection, we focus on comparing the changes in dAQM performance against other state-of-the-art AQMs when varying the packet sizes, the other network conditions are unchanged. Specifically, Fig. 15a to 15f represent HTTP CBR traffic (for short flows), and Fig. 16a to Fig. 16f represent FTP CBR traffic (for long flows).

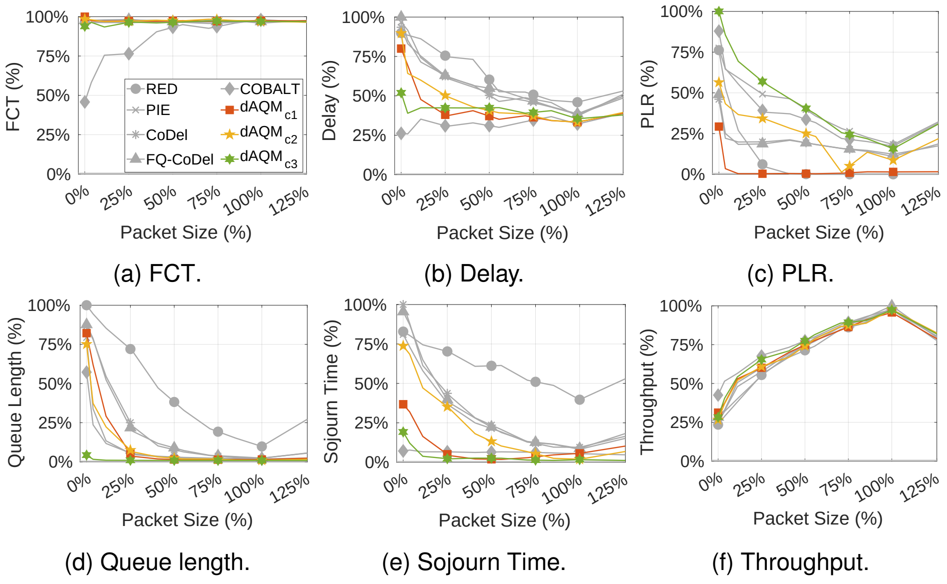


Figure 15: Performance of dAQM by increasing the packet size for HTTP.

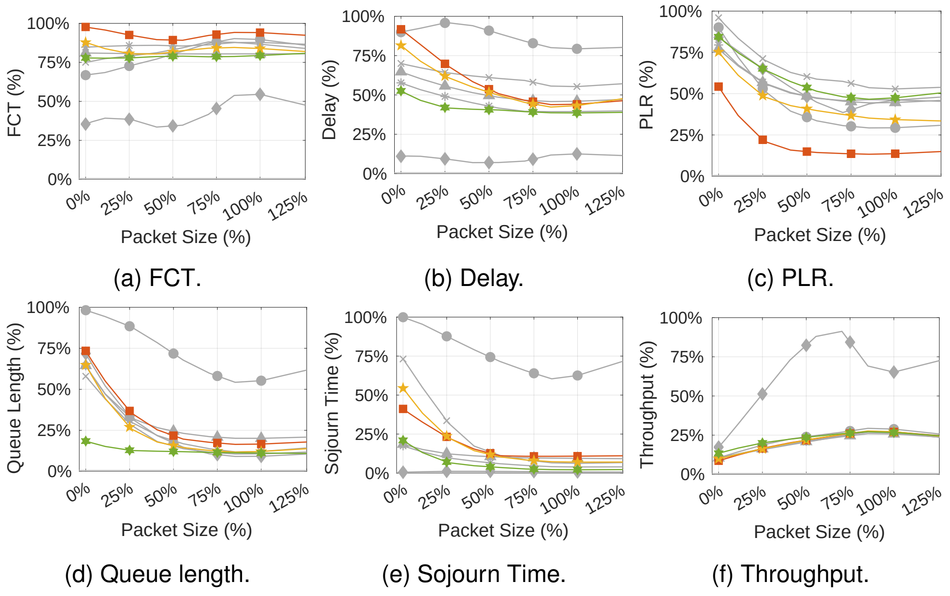


Figure 16: Performance of dAQM by increasing the packet size for FTP.

Analysis and Discussion:

Since we are using a fixed queue length of 2000 packets (as opposed to a fixed physical memory size measured in bytes) for the simulations, scenarios with smaller packet sizes coupled with high transmission rates can lead to rapid queue saturation. This may lead to congestion. As a result, with smaller packet sizes, metrics such as queue length, sojourn time, PLR, and delay are notably high due to this congestion, while throughput is low. As the packet sizes increase, those former metrics would decrease, while the throughput would increase.

• **FCT**: In the short flows, the FCT of all AQMs are approximately the same, except that COBALT was initially performing the best with smaller packet sizes, and its performance slowly align to the others with increasing packet sizes. In the long flows, COBALT is leading in giving the lowest FCT, which demonstrates a performance that is approximately 14% better than dAQM$_{c3}$, where dAQM$_{c3}$ is the second best option being at least 10% better other AQMs.

• **Delay**: In the short flow, three dAQMs give the lowest delay at large packet size together with COBALT, and outperform RED, PIE, CoDel, and FQ-CoDel. However, COBALT was already obtaining the lowest delay at a small packet size. Followed by dAQM$_{c3}$, whose delay is about 50% lower than FQ-CoDel, PIE, CoDel, and RED at very small packet sizes. In the long flow, dAQM$_{c3}$ gives the second lowest delay, worse than COBALT, but outperforms RED by about 47%. In the long flow, dAQM$_{c3}$ remained the second best option adaptive to both small and large packet sizes.

• **PLR**: dAQM$_{c1}$ gives the lowest PLR in both scenarios. In the short flows, its performance is more than 90% better than PIE, tying with RED for the lowest PLR. In the long flows, dAQM$_{c1}$ leads all the AQMs, outperforming RED (second lowest) by about 55%. Notably, dAQM$_{c1}$ consistently delivers the lowest PLR across all packet sizes, from small to large, even under heavy congestion.

• **Queue length**: The queue lengths of dAQMs are the lowest for both short and long flows at large packet sizes. With small packets in the short flow, dAQM$_{c3}$ is about 95% better than RED (worst) and 92% better than COBALT (second best) in obtaining the lowest queue length. While it is around 80% better than RED and 65% than PIE (second best) in the long flows. With large packets, the other algorithms slowly adapt to the network condition, thus decreasing their queue length until it aligns (or close) with dAQM$_{c3}$ performance. Notably, dAQM$_{c3}$ consistently maintains the lowest queue length starting from small packet sizes to large sizes. Underscoring its high configurability and robustness for queue management.

• **Sojourn Time**: For large packet size, the sojourn times for dAQM$_{c1}$, dAQM$_{c2}$, and dAQM$_{c3}$ consistently rank among the lowest in both short and long flows. Although dAQM$_{c3}$ (best) is about 70%, and dAQM$_{c2}$ (second best) is about 65% better than COBALT (third best) in the short flow. COBLAT became about 70% better than dAQM$_{c3}$ (second best) in the long flows. It has more than 95% improved performance than given by the worst case RED, in both the short, and long flows. From the analysis, dAQM$_{c3}$ is very preferred for its superior performance in both flow categories and a wide range of packet sizes.

• **Throughput**: For short flows, all AQMs achieve a similar, maximized throughput level. In the long flows, while dAQMs perform on par with RED, PIE, CoDel, and FQ-CoDel, they are outperformed by COBALT by approximately 8% to 23%. Furthermore, analysis shows, that there is an increasing throughput in the long flows, for loads up to roughly 65%, then it slowly reduces.

These correspond to our previous findings (Fig. 8e, 9e, 10e). For Poisson-distributed HTTP and VoIP (slow flows), the throughput across all AQMs was roughly equivalent (except CoDel has a slightly reduced throughput for HTTP), while Poisson-distributed FTP and Streaming (long flows), had pronounced performance differences (COBALT and CoDel have higher throughput). This is also supported by the analysis of various traffic distribution models for HTTP. The throughput was roughly the same among all AQMs and traffic distribution models. Whereas only COBALT had increased throughput for CBR streaming. Furthermore, we have also gained more analytical feedback on long flows under CBR traffic with COBALT increased throughput in variation to load (Fig. 16f). The throughput of COBALT increases up to 50% load, and then slowly decreases to align with the others. Increasing load, or increasing packet size, essentially both accumulate the data flowing on the network link. Thus, the similar discovery of increased throughput is justifiable. Therefore, we could suggest, that although the simulations were done only up to 125% packet size, if we keep increasing the packet size, COBALT eventually would also align its throughput with other AQMs at the end.

6.3.4 Section Discussion

Analysis shows the performance of dAQM, across varying network parameters and conditions, exhibits significant success. When considering both short and long flows, dAQM consistently ranks among the top performers in key metrics such as PLR, queue length, sojourn time, and delay, often surpassing other state-of-the-art AQMs.

Specifically, in short flows, dAQM effectively manages to achieve the lowest queue lengths, sojourn times, PLR, and maximize throughput. For long flows, the performance remains impressive. dAQM$_{c3}$ is frequently being highlighted for obtaining the lowest queue length and sojourn time under varying packet sizes and heavy load. dAQM$_{c1}$, on the other hand, frequently stands out in obtaining the minimal PLR. COBALT, however, emerges as a strong competitor, at times outperforming dAQM, especially in terms of delay and throughput.

Performance analysis suggests that dAQM adapts well across various scenarios, highlighting its robustness and potential based on consistently pronounced performance. It showcases how its programmability and configurability aid its ability to handle varied traffic patterns and network conditions.

7 Conclusion and Future Work

In the thesis, we proposed a new adaptive network management technique, the dAQM algorithm. It uses eight advanced traffic features to precisely calculate the PDP, including constant and higher-order derivatives of sojourn time and buffer size. It allows early detection of congestion, and proactively controls packet drop with a pre-defined probability and duration.

7.1 Key Research Findings

• **TCP vs. UDP Protocols**: analysis shows dAQM has proven to be particularly efficient when employed with the TCP protocol due to its reliance on sojourn time and queue length variations. Greater variations lead to higher derivative values and better management of the dAQM drop mechanism.

• **Traffic Distribution and Flow Type**: Analysis showed the behaviors of AQM techniques, are highly influenced by the underlying traffic characteristics. dAQM has shown superior effectiveness under Pareto distribution, for both short and long flows, in obtaining the minimal queue length and sojourn time.

• **Network Parameters Variation**: Analysis shows dAQM scales well to scenarios with varying network parameters like offered load, packet size, and dAQM drop rate. Under certain conditions, the network was very congested (e.g., at long flows with low loads or small packet sizes), however, dAQM$_{c3}$ (min. queue length and sojourn time), and dAQM$_{c1}$ (min. PLR) adapt well without the need for tuning or reconfiguration of its initial parameters. It gives consistently low queue length and sojourn time.

The performance analysis of various traffic models (CBR, VBR, Poisson, Pareto and Weibull) and flow types (short flows: Gaming, VoIP, HTTP, and long flows: FTP, Streaming), demonstrates that dAQM is compatible with various network traffic scenarios. Different configurations yield substantial differences in their performance under the same network conditions. The performance of the dAQM highly depends on the expected traffic patterns. Its high programmability and configurability allow it to be tuned and tailored according to the expected traffic patterns and performance objectives. Where minimal queue length allows for less physical memory resources, and small sojourn time provides a fast and responsive network connection. Alternatively, dAQM can move its focus to obtain the lowest PLR. Keeping a low PLR, allows for a more stable network, and less disruption or loss of packets transmitted over the network link.

7.2 Future Work

At the moment, the dAQM algorithm was designed based on constant packet arrival time. In the future, we aim to improve the approach by implementing dynamic packet arrival time in the higher-order derivatives calculation. By this transition, we could enhance the accuracy of the variations in current network congestion, leading to more efficient and adaptive network management.

Furthermore, we would like to focus on improving queue management techniques using deep learning technologies [66]. Deep learning, a subset of machine learning, uses artificial neural networks to understand vast data and make precise predictions. Deep learning models through data analysis, can understand and predict future traffic patterns in real-time. Therefore, improving the network performance by adaptively switching between different AQM algorithms or different sets of AQM configurations.

Acknowledgments

Saad Saleh conceived the proposed algorithm. Sunny Shu and Saad Saleh jointly contributed to the design and analysis of the approach through network simulations. Boris Koldehofe and Saad Saleh jointly contributed to the approach's supervision, discussion, and performance analysis. The research paper (under review) was written by Saad Saleh, jointly with Boris Koldehofe. The bachelor's thesis was written by Sunny Shu. Boris Koldehofe has been involved in the review of both the research paper and the bachelor's thesis.

Submitted Research Paper

Part of this thesis is included in the paper:

Saad Saleh, Sunny Shu, Boris Koldehofe,“Adaptive In-Network Queue Management using Derivatives of Sojourn Time and Buffer Size”, 10 pages, 2023, Submitted.

Bibliography

[1] ITU, “Number of internet users worldwide from 2005 to 2022.”

[2] CybersecurityVentures, “ Humans On The Internet Will Triple From 2015 To 2022 And Hit 6 Billion.”

[3] TransformInsights, “ Number of Internet of Things (IoT) connected devices worldwide from 2019 to 2021, with forecasts from 2022 to 2030.”

[4] S. Saleh, Z. Shah, and A. Baig, “Capacity analysis of combined IPTV and VoIP over IEEE 802.11n,” in *Proceedings of the 38th Annual IEEE Conference on Local Computer Networks*, pp. 785–792, 2013. doi:10.1109/LCN.2013.6761333.

[5] S. Saleh, Z. Shah, and A. Baig, “IPTV Capacity Analysis Using DCCP over IEEE 802.11n,” in *Proceedings of the IEEE 78th Vehicular Technology Conference (VTC Fall)*, pp. 1–5, 2013. doi:10.1109/VTCFall.2013.6692252.

[6] A. Iqbal, U. Javed, S. Saleh, J. Kim, J. S. Alowibdi, and M. U. Ilyas, “Analytical Modeling of End-to-End Delay in OpenFlow Based Networks,” *IEEE Access*, vol. 5, pp. 6859–6871, 2017. doi:10.1109/ACCESS.2016.2636247.

[7] J. Gettys, “Bufferbloat: Dark buffers in the Internet,” *IEEE Internet Computing*, vol. 15, no. 3, pp. 96–96, 2011. doi:10.1109/mic.2011.56.

[8] I. C. Paschalidis and S. Vassilaras, “Importance sampling for the estimation of buffer overflow probabilities via trace-driven simulations,” *IEEE/ACM Transactions on Networking*, vol. 12, no. 5, pp. 907–919, 2004. doi:10.1109/TNET.2004.836139.

[9] X. Lan, Q. Chen, L. Cai, and L. Fan, “Buffer-Aided Adaptive Wireless Powered Communication Network With Finite Energy Storage and Data Buffer,” *IEEE Transactions on Wireless Communications*, vol. 18, no. 12, pp. 5764–5779, 2019. doi:10.1109/TWC.2019.2938958.

[10] M. Andrews, K. Kumaran, K. Ramanan, A. Stolyar, P. Whiting, and R. Vijayakumar, “Providing quality of service over a shared wireless link,” *IEEE Communications Magazine*, vol. 39, no. 2, pp. 150–154, 2001. doi:10.1109/35.900644.

[11] S. Saleh, Z. Shah, and A. Baig, “Improving QoS of IPTV and VoIP over IEEE 802.11n,” *Computers & Electrical Engineering*, vol. 43, pp. 92–111, 2015. doi:10.1016/j.compeleceng.2014.10.017.

[12] K. Shafique, B. A. Khawaja, F. Sabir, S. Qazi, and M. Mustaqim, “Internet of Things (IoT) for Next-Generation Smart Systems: A Review of Current Challenges, Future Trends and Prospects for Emerging 5G-IoT Scenarios,” *IEEE Access*, vol. 8, pp. 23022–23040, 2020. doi:10.1109/ACCESS.2020.2970118.

[13] K. Nichols and V. Jacobson, “Controlling queue delay,” *Communications of the ACM*, vol. 55, no. 7, pp. 42–50, 2012. doi:10.1145/2208917.2209336.

[14] W. chang Feng, K. Shin, D. Kandlur, and D. Saha, “The BLUE active queue management algorithms,” *IEEE/ACM transactions on networking*, vol. 10, no. 4, pp. 513–528, 2002. doi:10.1109/tnet.2002.801399.

[15] M. M. Hamdi, S. A. Rashid, M. I. M. A. Altahrawi, M. F. Mansor, and M. K. AbuFoul, “Performance Evaluation of Active Queue Management Algorithms in Large Network,” in *Proceedings of the IEEE 4th International Symposium on Telecommunication Technologies (ISTT)*, pp. 1–6, 2018. doi:10.1109/ISTT.2018.8701716.

[16] S. Saleh and B. Koldehofe, “The Future is Analog: Energy-Efficient Cognitive Network Functions over Memristor-Based Analog Computations,” in *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks*, HotNets ’23, (New York, NY, USA), p. 254–262, Association for Computing Machinery, 2023. doi:10.1145/3626111.3628192.

[17] L. Xue, “Simulation of Network Congestion Control Based on RED Technology,” in *Proceedings of International Conference on Computational and Information Sciences*, pp. 1497–1500, 2013. doi:10.1109/ICCIS.2013.394.

[18] 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. doi:10.1109/90.251892.

[19] S. Hanlin, J. Yuehui, C. Yidong, W. Hongbo, and C. Shiduan, “Improving fairness of RED aided by lightweight flow information,” in *Proceedings of the 2nd IEEE International Conference on Broadband Network & Multimedia Technology*, pp. 335–339, 2009. doi:10.1109/ICBNMT.2009.5348505.

[20] T. M. Chen, “Network traffic modeling,” in *The handbook of computer networks*, vol. 3, p. 156, Wiley Hoboken, NJ, 2007. doi:10.1002/9781118256107.ch21.

[21] T. Bonald, M. May, and J.-C. Bolot, “Analytic evaluation of RED performance,” in *Proceedings of the IEEE INFOCOM 2000*, vol. 3, pp. 1415–1424 vol.3, 2000. doi:10.1109/INFCOM.2000.832539.

[22] C.-W. Feng, L.-F. Huang, C. Xu, and Y.-C. Chang, “Congestion Control Scheme Performance Analysis Based on Nonlinear RED,” *IEEE Systems Journal*, vol. 11, no. 4, pp. 2247–2254, 2017. doi:10.1109/JSYST.2014.2375314.

[23] K. Nichols, V. Jacobson, A. McGregor, and J. Iyengar, “Controlled Delay Active Queue Management.” RFC 8289, 2018. Online: `https://www.rfc-editor.org/info/rfc8289`.

[24] J. Kua, P. Branch, and G. Armitage, “Detecting Bottleneck Use of PIE or FQ-CoDel Active Queue Management During DASH-like Content Streaming,” in *Proceedings of the IEEE 45th Conference on Local Computer Networks (LCN)*, pp. 445–448, 2020. doi:10.1109/LCN48667.2020.9314804.

[25] T. Jain, A. B., and M. P. Tahiliani, “Performance Evaluation of CoDel for Active Queue Management in Wired-Cum-Wireless Networks,” in *Proceedings of the Fourth International Conference on Advanced Computing & Communication Technologies*, pp. 381–385, IEEE, 2014. doi:10.1109/ACCT.2014.97.

[26] G. White and J. Padden, “Preliminary study of CoDel AQM in a DOCSIS network,” tech. rep., CableLabs, 2012.

[27] I. Järvinen and M. Kojo, "Evaluating CoDel, PIE, and HRED AQM techniques with load transients," in *Proceedings of the 39th Annual IEEE Conference on Local Computer Networks*, pp. 159–167, IEEE, 2014. doi:10.1109/LCN.2014.6925768.

[28] 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 *Proceedings of the IEEE 14th international conference on high performance switching and routing (HPSR)*, pp. 148–155, IEEE, 2013. doi:10.1109/hpsr.2013.6602305.

[29] 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, 2017. Online: `https://www.rfc-editor.org/info/rfc8033`.

[30] G. F. Franklin, J. D. Powell, A. Emami-Naeini, and J. D. Powell, *Feedback control of dynamic systems*, vol. 4. Pearson, 2nd ed., 2002.

[31] 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, 2018. Online: `https://www.rfc-editor.org/info/rfc8290`.

[32] G. Lee, "Switch fabric technology," in *Cloud Networking*, pp. 37–64, Morgan Kaufmann, 2014. doi:https://doi.org/10.1016/B978-0-12-800728-0.00003-5.

[33] J. Palmei, S. Gupta, P. Imputato, J. Morton, M. P. Tahiliani, S. Avallone, and D. Täht, "Design and Evaluation of COBALT Queue Discipline," in *Proceedings of the IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, pp. 1–6, 2019. doi:10.1109/LANMAN.2019.8847054.

[34] B. Suter, T. Lakshman, D. Stiliadis, and A. Choudhury, "Design considerations for supporting TCP with per-flow queueing," in *Proceedings of the IEEE INFOCOM 1998*, vol. 1, pp. 299–306 vol.1, 1998. doi:10.1109/INFCOM.1998.659666.

[35] T. Høiland-Jørgensen, D. Täht, and J. Morton, "Piece of CAKE: A Comprehensive Queue Management Solution for Home Gateways," in *Proceedings of the IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, pp. 37–42, 2018. doi:10.1109/LANMAN.2018.8475045.

[36] A. Baiocchi, *Network Traffic Engineering: Stochastic Models and Applications*. John Wiley & Sons, 2020. doi:10.1002/9781119632498.

[37] D. Bertsekas and R. Gallager, *Data networks* . Athena Scientific, 2nd ed., 2021.

[38] W. Feller, *An Introduction to Probability Theory and its Applications* , vol. 2. John Wiley & Sons, 2nd ed., 1991.

[39] W. Willinger and V. Paxson, "Where mathematics meets the Internet," *Notices of the AMS*, vol. 45, no. 8, pp. 961–970, 1998.

[40] V. Paxson and S. Floyd, "Wide area traffic: the failure of Poisson modeling," *IEEE/ACM Transactions on Networking*, vol. 3, no. 3, pp. 226–244, 1995. doi:10.1109/90.392383.

[41] W. Leland, M. Taqqu, W. Willinger, and D. Wilson, "On the self-similar nature of Ethernet traffic (extended version)," *IEEE/ACM Transactions on Networking*, vol. 2, no. 1, pp. 1–15, 1994. doi:10.1109/90.282603.

[42] J. Gordon, "Pareto process as a model of self-similar packet traffic," in *Proceedings of GLOBE-COM'95*, vol. 3, pp. 2232–2236, IEEE, 1995. doi:10.1109/GLOCOM.1995.502798.

[43] O. Kutuzov and T. Tatarnikova, "Model of a self-similar traffic generator and evaluation of buffer storage for classical and fractal queuing system," in *Proceedings of Moscow Workshop on Electronic and Networking Technologies, MWENT*, vol. 1, p. 1, 2018. doi:10.1109/mwent.2018.8337306.

[44] M. Rytgaard, "Estimation in the Pareto Distribution," *ASTIN Bulletin: The Journal of the IAA*, vol. 20, no. 2, p. 201–216, 1990. doi:10.2143/AST.20.2.2005443.

[45] K. Park and W. Willinger, "Self-Similar Network Traffic: An Overview," *Self-Similar Network Traffic and Performance Evaluation*, pp. 1–38, 2000. doi:10.1002/047120644x.ch1.

[46] E. Chlebus and G. Divgi, "The Pareto or Truncated Pareto Distribution? Measurement-Based Modeling of Session Traffic for Wi-Fi Wireless Internet Access," in *Proceedings of the IEEE Wireless Communications and Networking Conference*, pp. 3625–3630, 2007. doi:10.1109/WCNC.2007.664.

[47] R. Singhai, S. D. Joshi, and R. K. Bhatt, "Offered-load model for Pareto inter-arrival network traffic," in *Proceedings of the IEEE 34th Conference on Local Computer Networks*, pp. 364–367, IEEE, 2009. doi:10.1109/lcn.2009.5355115.

[48] A. Feldmann, "Characteristics of TCP connection arrivals," *Self-Similar Network Traffic and Performance Evaluation*, pp. 367–399, 2000. doi:10.1002/047120644x.ch15.

[49] M. A. Arfeen, K. Pawlikowski, D. McNickle, and A. Willig, "The role of the Weibull distribution in Internet traffic modeling," in *Proceedings of the 25th International Teletraffic Congress (ITC)*, pp. 1–8, IEEE, 2013. doi:10.1109/itc.2013.6662948.

[50] P. Barford and M. Crovella, "Generating Representative Web Workloads for Network and Server Performance Evaluation," *SIGMETRICS Perform. Eval. Rev.*, vol. 26, p. 151–160, jun 1998. doi:10.1145/277858.277897.

[51] R. B. Abernethy, J. Breneman, C. Medlin, and G. L. Reinman, "Weibull Analysis Handbook," *Pratt & Whitney Aircraft Government Products Division*, 1983. doi:10.21236/ada143100.

[52] X. Li, J. Li, and L. Li, "Performance Analysis of Impaired SWIPT NOMA Relaying Networks Over Imperfect Weibull Channels," *IEEE Systems Journal*, vol. 14, no. 1, pp. 669–672, 2020. doi:10.1109/JSYST.2019.2919654.

[53] A. M. Al-Sharafi and B. A. Alrimi, "Throughput Comparison of AOMDV and OLSR Ad Hoc Routing Protocols Using VBR and CBR Traffic Models," in *Proceedings of the International Conference on Advanced Computer Science Applications and Technologies*, pp. 466–469, IEEE, 2013. doi:10.1109/ACSAT.2013.97.

[54] S. K. Gupta, S. H. Alsamhi, and R. K. Saket, "Comparative performance analysis of AODV for CBR & VBR traffic under influence of ART & DPC," in *Proceedings of the 11th International Conference on Industrial and Information Systems (ICIIS)*, pp. 112–117, IEEE, 2016. doi:10.1109/ICIINFS.2016.8262917.

[55] University of Washington NS-3 Consortium, "ns-3 Network Simulator." Online: `https://www.nsnam.org/`.

[56] PlantUML Developers, "Plantuml." Online: `http://plantuml.com`.

[57] Gnuplot Developers, "Gnuplot." Online: `http://www.gnuplot.info`.

[58] MathWorks Inc., "Matlab." Online: `https://www.mathworks.com/products/matlab.html`.

[59] The GNOME Project, "gedit." Online: `https://help.gnome.org/users/gedit/stable/`.

[60] Sunny Shu, "Derivative-based Active Queue Management Github Repository." Online: `https://github.com/rug-ds-lab/bsc-2023-adaptive-queue-management`.

[61] P. L. Dorlan, *An Introduction to Computer Networks*. Autoedición, 2nd ed., 2020.

[62] J. Zheng, C. Wu, T. Lan, C. Tian, and G. Chen, "Revisiting Weighted AIMD-based Congestion Control: A Comprehensive Perspective," in *Proceedings of the IEEE/ACM 31st International Symposium on Quality of Service (IWQoS)*, pp. 1–10, IEEE, 2023. doi:10.1109/iwqos57198.2023.10188753.

[63] M. Hock, R. Bless, and M. Zitterbart, "Experimental Evaluation of BBR Congestion Control," in *Proceedings of the IEEE 25th International Conference on Network Protocols (ICNP)*, pp. 1–10, IEEE, 2017. doi:10.1109/icnp.2017.8117540.

[64] X. Du, K. Xu, L. Xu, K. Zheng, M. Shen, B. Wu, and T. Li, "R-AQM: Reverse ACK Active Queue Management in Multitenant Data Centers," *IEEE/ACM Transactions on Networking*, vol. 31, no. 2, pp. 526–541, 2023. doi:10.1109/TNET.2022.3197973.

[65] U. Javed, A. Iqbal, S. Saleh, S. A. Haider, and M. U. Ilyas, "A stochastic model for transit latency in OpenFlow SDNs," *Computer Networks*, vol. 113, pp. 218–229, 2017. doi:10.1016/j.comnet.2016.12.015.

[66] S. Saleh and B. Koldehofe, "On Memristors for Enabling Energy Efficient and Enhanced Cognitive Network Functions," *IEEE Access*, vol. 10, pp. 129279–129312, 2022. doi:10.1109/ACCESS.2022.3226447.

Appendices

A Performance Statistics

Table 4: Performance Statistics for HTTP dAQM drop rate at 99%.

P.M	RED	PIE	CoDel	FQ-CoDel	COBALT	dAQM$_{c1}$	dAQM$_{c2}$	dAQM$_{c3}$
FCT	2.808	2.849	2.805	2.850	2.844	2.833	2.849	2.830
Delay	209.262	239.050	190.774	183.466	165.255	182.211	152.962	176.751
PLR	0.000	0.158	0.085	0.089	0.124	0.058	0.097	0.110
Q.L	112.743	14.430	26.231	26.373	19.154	9.079	10.997	9.733
S.J	1550.973	224.706	360.896	372.424	210.935	36.417	80.768	50.501
Th	437.468	459.562	450.983	461.361	444.482	455.725	454.403	450.602

Table 5: Performance Statistics for FTP dAQM drop rate at 99%.

P.M	RED	PIE	CoDel	FQ-CoDel	COBALT	dAQM$_{c1}$	dAQM$_{c2}$	dAQM$_{c3}$
FCT	30.256	29.221	27.829	25.466	25.797	15.426	15.395	15.838
Delay	2403.579	1908.472	1349.269	1955.000	668.221	1082.524	1138.196	1341.050
PLR	0.453	0.614	0.569	0.636	0.688	0.366	0.536	0.531
Q.L	677.576	94.579	103.856	303.726	67.240	122.907	144.558	124.514
S.J	13166.835	809.967	593.505	2911.114	78.786	394.510	1191.862	443.276
Th	263.616	225.977	221.230	206.776	302.307	292.796	278.252	291.550

Table 6: Performance Statistics for HTTP offered load at 4 Mbps.

P.M	RED	PIE	CoDel	FQ-CoDel	COBALT	dAQM$_{c1}$	dAQM$_{c2}$	dAQM$_{c3}$
FCT	2.807	2.864	2.840	2.832	2.421	2.863	2.844	2.848
Delay	351.325	346.713	326.511	355.850	183.078	234.450	288.611	268.654
PLR	0.000	0.358	0.167	0.172	0.283	0.004	0.212	0.354
Q.L	269.563	58.345	122.756	132.708	59.730	44.384	68.412	36.075
S.J	2096.929	632.055	994.035	983.876	218.218	128.296	707.751	111.155
Th	629.221	641.912	644.890	655.699	745.908	632.401	635.772	638.260

Table 7: Performance Statistics for FTP offered load at 10 Mbps.

P.M	RED	PIE	CoDel	FQ-CoDel	COBALT	dAQM$_{c1}$	dAQM$_{c2}$	dAQM$_{c3}$
FCT	17.713	17.042	17.603	15.949	13.590	18.255	16.511	15.800
Delay	2378.042	1652.124	1289.483	1472.566	578.888	1402.792	1269.053	1254.581
PLR	0.330	0.553	0.542	0.506	0.651	0.156	0.387	0.521
Q.L	563.92	1 114.905	120.361	225.680	94.184	168.964	132.039	129.377
S.J	11100.006	1013.155	687.210	1844.623	109.143	1782.286	1165.219	382.455
Th	318.734	292.913	284.268	276.652	345.718	295.467	293.194	280.306

Table 8: Performance Statistics for HTTP packet size at 1400 B.

P.M	RED	PIE	CoDel	FQ-CoDel	COBALT	dAQM$_{c1}$	dAQM$_{c2}$	dAQM$_{c3}$
FCT	2.855	2.799	2.828	2.826	2.863	2.843	2.833	2.830
Delay	248.971	206.807	200.930	209.064	173.834	180.562	180.015	191.975
PLR	0.000	0.136	0.081	0.092	0.128	0.011	0.064	0.117
Q.L	102.201	14.728	23.638	25.388	18.906	15.175	10.910	9.712
S.J	1349.799	279.861	323.248	298.203	185.361	191.932	65.623	56.680
Th	457.526	457.877	458.194	468.508	458.311	448.505	457.457	456.076

Table 9: Performance Statistics for FTP packet size at 1400 B.

P.M	RED	PIE	CoDel	FQ-CoDel	COBALT	dAQM$_{c1}$	dAQM$_{c2}$	dAQM$_{c3}$
FCT	17.713	17.042	17.603	15.949	13.590	18.713	16.795	15.800
Delay	2378.042	1652.124	1289.483	1472.566	578.888	1331.749	1287.087	1254.581
PLR	0.330	0.553	0.542	0.506	0.651	0.151	0.380	0.521
Q.L	563.921	114.905	120.361	225.680	94.184	176.157	125.580	129.377
S.J	11100.006	1013.155	687.210	1844.623	109.143	2020.406	1189.185	382.455
Th	318.734	292.913	284.268	276.652	345.718	290.585	296.261	280.306