






Article

New RED-Type TCP-AQM Algorithms Based on Beta Distribution Drop Functions

Angel Giménez ^{1,2,*} , Miguel A. Murcia ¹ , José M. Amigó ^{1,2} , Oscar Martínez-Bonastre ^{1,2}  and José Valero ^{1,2} 

¹ Departamento de Estadística, Matemáticas e Informática, Universidad Miguel Hernández de Elche, Avenida Universidad s/n, 03202 Elche, Spain

² Centro de Investigación Operativa, Universidad Miguel Hernández de Elche, Avenida de la Universidad s/n, 03202 Elche, Spain

* Correspondence: a.gimenez@umh.es

Abstract: In recent years, Active Queue Management (AQM) mechanisms to improve the performance of TCP/IP networks have acquired a relevant role. In this paper, we present a simple and robust RED-type algorithm together with a couple of dynamical variants with the ability to adapt to the specific characteristics of different network environments, as well as to the user's needs. We first present a basic version called Beta RED (BetaRED), where the parameters can be tuned according to the specific network conditions. The aim is to introduce control parameters that are easy to interpret and provide a good performance over a wide range of values. Secondly, BetaRED is used as a framework to design two dynamic algorithms, which we will call Adaptive Beta RED (ABetaRED) and Dynamic Beta RED (DBetaRED). In those new algorithms, certain parameters are dynamically adjusted so that the queue length remains stable around a predetermined reference value and according to changing network traffic conditions. Finally, we present a battery of simulations using the Network Simulator 3 (ns-3) software with a two-fold objective: to guide the user on how to adjust the parameters of the BetaRED mechanism, and to show a performance comparison of ABetaRED and DBetaRED with other representative algorithms that pursue a similar objective.

Keywords: congestion control; active queue management; random early detection; beta distribution; stability



Citation: Giménez, A.; Murcia, M.A.; Amigó, J.M.; Martínez-Bonastre, O.; Valero, J. New RED-Type TCP-AQM Algorithms Based on Beta Distribution Drop Functions. *Appl. Sci.* **2022**, *12*, 11176. <https://doi.org/10.3390/app122111176>

Academic Editors: Francesco Palmieri, Davide Careglio and Mirosław Klinkowski

Received: 24 September 2022

Accepted: 3 November 2022

Published: 4 November 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Over the last few decades, the Internet has become increasingly faster by many orders of magnitude, but at the same time the number of users has increased, along with a huge deployment of Internet-based applications, and with a multitude of bandwidth and latency requirements. This growth in data flow has resulted in increased network traffic congestion, which, if not properly managed, could lead to a considerable decrease in performance, a critical factor for any Internet service. The TCP transport protocol has become the standard for data communications, being the most common type of data flow on the Internet. The most popular network applications (FTP, TELNET, Web Access, etc.) use TCP protocol in their communications. This, together with the massive use of Voice over Internet Protocol (VoIP) applications, online games, financial exchanges, etc., has caused buffers to fill up, resulting in the phenomenon known as bufferbloat [1], characterized by high latency and massive discarding of packets over long periods of time, leading to poor quality of service for the end user.

Congestion control is a critical part of the Transmission Control Protocol (TCP), which directly influences transport performance. TCP is the dominant transport layer protocol on the Internet. In general, there are two congestion control mechanisms: (1) end-to-end congestion controls, which are approaches adopted by TCP and are achieved mainly at the transport layer; and (2) network-assisted congestion controls, which are controls adopted by routers. This mechanism uses router queue size and/or delay to monitor the

congestion state of the network. To meet the demands of Internet users and applications, research both in academic and industrial environments has focused on improving these two control mechanisms to achieve high performance and avoid data traffic collapse. This paper is concerned with the second approach, and aims to design a new congestion control mechanism to be implemented at the router. For those readers interested in the first approach, the surveys [2,3] collect the main transport protocols that have been proposed in recent years.

The main objective of congestion control mechanisms is to keep the network operating fairly close to its nominal capacity, even when faced with extreme overload. This objective is achieved by acting according to two fundamental premises. The first is to prevent network congestion before it occurs and to dissolve congestion if it cannot be avoided. The second is to provide a fairness service to different connections, along with support for various Internet application domains with varying quality of service (QoS) requirements. Designing good congestion control mechanisms is extremely difficult; the specific characteristics of the connection and each individual link can significantly affect performance, often in unpredictable ways.

Active Queue Management (AQM) algorithms are algorithms implemented in routers that act on the buffer queue to control its length so that efficient congestion control of the system is achieved. Such management also allows TCP to do its job of sharing links properly, without which it cannot function as intended. The primary goal of an AQM is to prevent congestion before it occurs and, if it has already occurred, to try to control it. AQM mechanisms act on the length of queues in buffers to achieve lower delay, and try to absorb short-term variations (e.g., bursts), thus playing a crucial role in congestion control on the Internet. Numerous AQM schemes have been proposed in recent years to properly manage queues to avoid undesirable effects such as bufferbloat, link underutilization, large variations in queuing delays, etc. The truth is that there is no one algorithm that has dominated over the others, since network environments are very varied, and network traffic very changeable, so each algorithm has advantages and disadvantages according to those environments.

The Random Early Detection (RED) algorithm, proposed by Floyd and Jacobson [4], works by detecting incipient congestion and notifying the TCP transmission control protocol of the congestion by randomly discarding packets to avoid filling the router. The RED algorithm avoids some of the problems of using the simple Drop Tail [5] (such as blocking, full queues and global synchronization), whose mechanism consists of dropping all incoming packets when the buffer queue is full. In fact, RED has been the most studied AQM to date, and has been the basis for the development of new AQM systems. The reason for this is not only because RED was the first to be developed in the Internet community, but also because of the numerous drawbacks involved in using this algorithm, some of which have not yet been fully resolved. One of the main problems is the difficulty and uncertainty in adjusting its parameters for adequate performance, due to the high sensitivity of the RED parameters to traffic load. Thus, a poor choice of RED parameters can lead to other deficiencies such as forced drops, or link underutilization. Moreover, even if the RED parameters are properly tuned, they are very sensitive to network conditions and can also cause other more complex nonlinear effects such as bifurcation and chaos. In summary, it can happen that a set of parameters works perfectly for a certain network setting but not when those parameters are slightly changed. This is obviously not desirable, as Internet traffic conditions change rapidly. The study of stability and bifurcations of Internet congestion control models involving delay is a central issue that has been intensively studied in recent decades (see [6–14]).

The paper is organized as follows. In Section 2, the current AQM problem is presented, along with the description of the most recent algorithms that have similar objectives to BetaRED. This section also briefly reviews the AQM schemes with which we will compare the proposed new algorithms. Section 3 is devoted to describing the simulation scenarios and the metrics we will use to perform the comparison between the AQM schemes. In the

next two sections, new proposed algorithms are introduced and compared by numerical simulations with ns-3, namely, the main BetaRED algorithm (Section 4), and two new dynamic algorithms ABetaRED and DBetaRED based on the previous one (Section 5). Finally, in Section 6, some final conclusions are presented.

For the convenience of the reader, in Table 1 we list symbols of the most important parameters used in this paper.

Table 1. Notation for the most important parameters and variables used throughout the paper.

	AQM Algorithm	Name	Description
Tunable parameters	ABetaRED, DBetaRED, ARED, CoDel, PIE	T_{target}	Target delay
		T_{update}	Update interval time
		Alpha, Beta	Control parameters with different objectives according to the AQM.
	BetaRED, ABetaRED, DBetaRED	θ	Scale factor determining the standard deviation of the drop probability function
		w	Averaging weight
	BetaRED	q_{target}	Target queue length
		q_{min}	Lower threshold
		q_{max}	Upper threshold
		p_{max}	Maximum packet drop probability
	System parameters	All	B
C			Capacity of the channel (the maximum amount of error-free information that can be transmitted over the channel per unit time)
N			Number of flows in the dumbbell topology
M			Packet size
Variables	All	p	Drop probability
		q_{cur}	Current queue length at Router 1
		q_{avg}	Average queue length at Router 1

2. Description of the Problem and Related Works

Numerous AQM algorithms with different approaches have been proposed over the past two decades to handle the queuing delay problem. However, the conditions of a network and the specific user needs may vary from one scenario to another, so there is no algorithm that can satisfy all demands at the same time. In addition, the optimal configuration of the parameters involved in this type of algorithm is complicated. This is why no single AQM has had a predominant deployment over the others, and the AQM that obtains the best performance in the given scenario is selected. This fact is what has most negatively affected the widespread application of the RED mechanism. There are a large number of publications in the literature aimed at overcoming these difficulties by variations on the RED algorithm. A good summary can be found in [15], which in turn contains a large collection of citations of work based on RED technology. However, the problem is still topical and we can find numerous recent works addressing the issue.

Drop Tail is the simplest algorithm that can be designed: the buffer accepts packets until it is completely full, and when this happens, it discards the last packets received. The main problem with this mechanism is its performance when combined with TCP, which reduces the sending rate when it receives packet loss notifications, and this occurs only after the buffer is completely full. This, in turn, causes TCP to generate a flow with intermittent bursts, which, with full buffers, causes packet loss to increase in each Round

Trip Time (RTT) cycle, eventually ending up in a degraded circular dynamics with high latency and low throughput. The problem is further aggravated when the number of flows connected in the same link increases, each regulated by its own congestion window. It has been found that, in this situation, the slow start of each of the flows tends to synchronize, causing the phenomenon known as global synchronization [4], which is characterized by very low overall network throughput.

Active queue management attempts to provide a solution to problems that appear in Drop Tail (see [5,16]) by notifying congestion initiation signals before the buffer fills up. Notification can be done via packet drop or with the Explicit Congestion Notification (ECN) flag. The large amount of research on the design and study of AQM algorithms has resulted in significant improvement in performance metrics: latency, link utilization, throughput, jitter, etc., reducing burst losses and solving the global synchronization problem.

In the RED scheme, the probability function p of packet dropping at an instant is dependent on the average size of the queue length q_{avg} and its expression is given by

$$p(q_{avg}) = \begin{cases} 0 & \text{if } q_{avg} < q_{min}, \\ 1 & \text{if } q_{avg} > q_{max}, \\ p_{max} \cdot z(q_{avg}) & \text{otherwise.} \end{cases} \quad \text{where } z(q_{avg}) = \frac{q_{avg} - q_{min}}{q_{max} - q_{min}} \quad (1)$$

The values q_{min} and q_{max} are the lower and upper thresholds for the average queue length q_{avg} where below q_{min} all packets are accepted, and above q_{max} all packets are rejected. The value p_{max} is the maximum value for the packet drop probability function when the average queue length is between q_{min} and q_{max} , which is reached at the point $q_{avg} = q_{max}$. The average queue length is updated upon packet arrival according to the Exponential Weighted Moving Average (EWMA),

$$q_{avg}^{new} = (1 - w) \cdot q_{avg}^{old} + w \cdot q_{cur} \quad (2)$$

between the previous average queue length q_{avg}^{old} and the current queue length q_{cur} , where $0 < w < 1$ is the averaging weight. The higher w , the faster the RED mechanism reacts to the actual buffer occupancy.

Although RED was able to eliminate some shortcomings of Drop Tail such as blocking, full queues and global synchronization, other shortcomings remained. One of the main problems of RED is that its parameters must be adjusted according to different Internet traffic load states. Experiments and numerical simulations have been the main tool used to adjust the parameters of many of the algorithms studied. In the literature we find many conclusions based solely on simulation results, but they lack a theoretical foundation that guarantees the results. The combination of end-to-end TCP congestion control and active RED queue management can be modeled as a discrete-time dynamic system and this system exhibits a variety of irregular behaviors, such as bifurcation and chaos. Recently, in [11,17], a generalized RED-based model was proposed in which two new control parameters are introduced by means of a nonlinear packet dropping probability function, namely, the normalized incomplete beta function. In addition, in [12,18] a theoretical analysis of the global stability was performed, and the results were used to find robust ranges of the new control parameters. This theoretical study has helped to design the new AQM algorithms proposed here for queue management, since they showed the existence of a wide range of parameters for which a higher stability than RED is achieved. One of the advantages of the algorithms proposed here is that the parameters are easily and intuitively adjusted with guarantees of achieving a good balance between stability and performance, and supported by the theoretical analysis made in [12,18].

The idea of using a nonlinear packet drop probability function also appears in several papers. In [19], the authors designed an algorithm called Three-section Random Early Detection (TRED), where the probability function is divided into three sections in order to achieve a trade-off between delay and throughput by distinguishing between light,

moderate and high loads. Following a similar idea, an AQM scheme called Flexible Random Early Detection (FXRED) is proposed in [20] where different variations of the packet drop probability function are applied according to the state of the network traffic load. Moreover, in [21], a new probability model with a variation of the packet drop function with respect to RED is proposed, obtaining an increase in throughput and a reduction in the expected end-to-end delay.

There is a huge amount of research on the design of new dynamical AQM algorithms whose goal is to maintain a stable average queue length around a predetermined target under changing network traffic conditions. In fact, we can find recent work addressing the issue that show that the problem is still open. In [22–25], the RED algorithm is modified in order to improve its performance and stability for various network states, obtaining better results compared to their predecessors. In [26], the Weight Queue Active Queue Management (WQDAQM) scheme, based on dynamic monitoring and reaction depending on the traffic load, is proposed. This algorithm aims to maintain the average queue length between two dynamically predetermined thresholds to prevent the buffer from exceeding the latter and overflowing.

For our numerical comparison in Sections 4.3 and 5.3, we have chosen some representative AQM algorithms that are implemented in ns-3, such as Adaptive RED (ARED), Control Delay (CoDel) and Proportional Integral Controller Enhanced (PIE). We add a brief summary of the most important features of these algorithms as follows.

ARED, proposed by Feng et al. [27] and subsequently improved by Floyd et al. in [28], was intended to adjust the RED parameters adaptively to achieve a queue length around a prefixed target queue length. The main parameter that is tuned is the maximum packet drop probability, for which it uses an Additive-Increase-Multiplicative-Decrease (AIMD) approach. However, the performance of ARED is inferior to that of RED when faced with complex network environments.

CoDel [29,30] manages congestion control through the time that packets are in a given buffer (sojourn time), or the time a given packet spends in the queue of a buffer. Thus, CoDel distinguishes between a “good queue”, one that does not show bufferbloat, maintaining an adequate delay in the face of bursts of traffic, and a “bad queue”, one that, on the contrary, has a high buffer delay in the face of low utilization.

PIE (see [31–33]) uses an estimate of the buffer queue delay as an indicator of congestion, marking with this estimated time each packet at the buffer entrance. When queuing a packet, a random discard is performed with a probability p obtained as a function of the latency calculated as an estimate of the delay and the trend that this value develops. PIE adopts the model Proportional Integral (PI) [34] to maintain the queuing delay at a specified target value. Furthermore, the algorithm self-adjusts the control parameters as a function of the level of congestion, directly reflecting this measure in the current discard probability, this being updated at regular intervals.

The CoDel and PIE algorithms (see [32,35], in which a comparative analysis is performed) are designed with the main objective of minimizing queue latency while maintaining high link utilization. They represent solutions to the problem raised in [36] of the Internet Engineering Task Force (IETF), where a call is made for the design of new methods to control network latency. For dumbbell topologies, both algorithms have performed well and mainly serve the purpose for which they were designed, which is to allow packet bursts to fill the buffer queue, preventing the queue from stalling under a persistent packet load.

3. Scenarios of Simulation and Metrics

The complexity of the network means that the mathematical models of the protocols are not completely realistic and often the theoretically optimal algorithms do not perform as expected in real networks. This inevitably leads to the use of numerical simulations to obtain reliable predictions about the behavior of AQM algorithms in a real environment. Furthermore, in most cases (e.g., for TCP), the congestion control algorithm for a given transport protocol is implemented in the same code base as the core of that protocol and is

therefore the same for each end-to-end connection. Therefore, it is not possible to customize the response of the congestion algorithm to the characteristics of each connection.

3.1. Scenarios of Simulation

For testing the AQM algorithms, the topology used for the simulation (Sections 4.3 and 5.3) is a simple dumbbell topology (see Figure 1), where N is the number of long-lived TCP connections sharing a single bottleneck. Traffic is generated from the left side to the right side, specifically for each $i \in \{1, 2, \dots, N\}$, S_i and D_i denote the source and destination of the TCP flow i . The router R_1 on the left is where the bottleneck is actually located and the control AQM model will be installed. All other nodes, by default, have the Drop Tail queue manager installed. The edge links between the TCP sources and the router R_1 , and the router R_2 and the TCP sinks have a capacity of 100 Mbps with a mean of 1 ms propagation delay. Router R_1 is connected to R_2 through a capacity C of 50 Mbps and 10 ms propagation delay. The maximum buffer size B of each router is set to 1000 packets of a size M of 1000 bytes each. The TCP transport agent will be TCP Cubic.

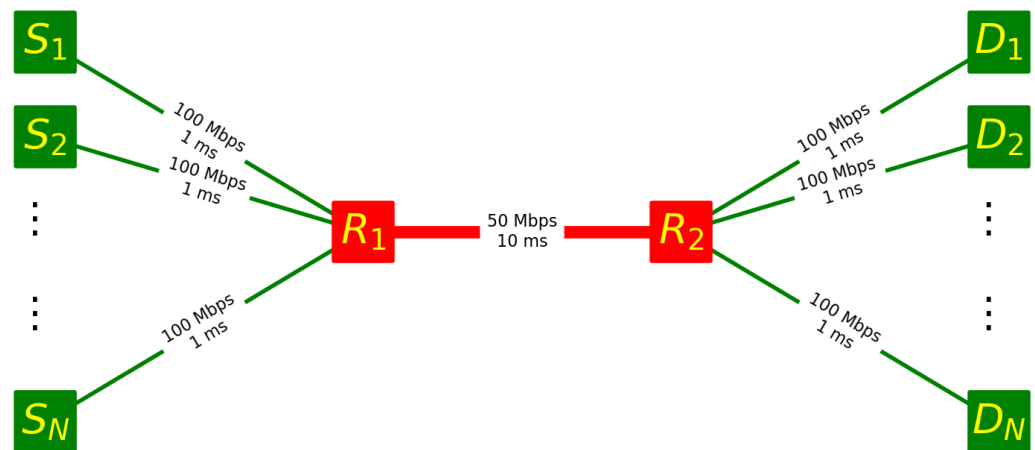


Figure 1. Network topology.

The level of congestion of the simulation is set according to the number N of long-lived TCP connections sharing the bottleneck. In that sense, we distinguish two different scenarios:

Scenario 1: Constant congestion level. In this case the number N of active flows is constant throughout the simulation time, which is of 250 s.

Scenario 2: Changing congestion level. In this case the number N of active flows varies as a function of time according to the following distribution:

- Between 0 and 50 s, the number of active flows is $N = 100$.
- Between 50 and 100 s, the number of active flows is $N = 200$.
- Between 100 and 150 s, the number of active flows is $N = N_{\max}$.
- Between 150 and 200 s, the number of active flows is $N = 200$.
- Between 200 and 250 s, the number of active flows is $N = 100$.

The congestion level in this case will be determined by the maximum number of active flows N_{\max} in the simulation.

3.2. Performance Metrics

The ultimate goal of TCP/AQM protocols is to improve the performance of end-user metrics. However, the final performance is strongly influenced by the performance of router-based metrics. Thus, to evaluate the performance of the proposed algorithms, we have chosen to use two router-based metrics and three end-user metrics. The following is a brief description of the most important characteristics of the metrics used for performance evaluation:

- Average queue length (AQL). The queue size indicates the number of packets pending transmission in the buffer queue. An unstable system is usually characterized by a synchronization in the TCP queues, accompanied by strong oscillations. Under suitable settings, the average queue length tends to stabilize at a value that we refer to as the equilibrium point. One of the main objectives of AQM algorithms is to stabilize the buffer queue size and, in this sense, we shall refer to the stability of an AQM algorithm as the ability to maintain the average queue length (or the average queuing delay) around a certain target value. The stability is an important performance characteristic of TCP/IP networks.
- The packet drop rate. It measures the ratio of the number of packets dropped by an AQM to the total number of packets in queue. In this count, packets dropped by the link or channel at the physical layer are not counted, considering only the drop rate at the network layer. The main objective of an AQM is to maintain a stable queue size with as low packet drop rate as possible. This increases the performance, since dropped packets are an early signal of congestion to the TCP, causing a decrease in its send rate.
- End-to-end throughput. This is a performance measure obtained between two interlocutors (server–host). It measures the actual transmission of the total data propagated with respect to the simulation time (from the time the data is sent to the time it is received). It is defined as the number of bits received correctly per unit of time. Specifically, the calculation of this metric is obtained from the ratio between the number of bits received by the server/host and the time elapsed between the reception of the first segment and the last one. To calculate the throughput in the dumbbell type topology, we have averaged the ratio between the number of bits received by the left and right hosts, and the sum of elapsed time between the reception of the first segment and the last one at each of these nodes.
- End-to-end delay (latency). It is one of the most significant metrics in a communication system and, in general terms, it is the time required to transmit a segment along its entire length, end-to-end. Specifically, it is calculated using the equation:

$$\text{Latency} = \text{propagation time} + \text{transmission time} + \text{queuing time} + \text{processing delay}$$

For its calculation in the dumbbell topology, the end-to-end delay times of all the segments sent between the left and right hosts are summed, divided by the number of segments received on both hosts.

- Jitter. Jitter in a flow is defined as the variation in delay of arriving packets over time. A very high jitter can cause packet loss due to buffer overflow. In the dumbbell topology, an average value is calculated by summing the time variations between the relative packets of all flows, and dividing by the number of variations.

All simulations presented in this paper were performed with ns-3 [37]. This is a discrete event network simulator for Internet systems, primarily intended for research and educational use. It is free software, licensed under GNU GPLv2, and is publicly available for research, development and use.

4. The Beta RED Scheme

The Beta RED algorithm that we introduce is inspired by the classical RED, where the packet drop probability function is dependent on two new parameters. The idea is to adapt the parameters in order to improve the performance and stability of the system according to the characteristics of the congestion scenarios, such as traffic load, available bandwidth and desired delay, etc. At first it might be thought that introducing two new parameters complicates the already intricate parameter tuning that the RED algorithm undergoes since its inception, but we will show that it is actually an advantage, since it is possible to easily and intuitively adapt these parameters to stabilize the average queue length as close as possible to a preset reference value. We will now briefly describe the biparametric family

of probability beta functions and the properties on which we will rely in the design of the new algorithm.

4.1. Normalized Incomplete Beta Function

The beta distribution function (or normalized incomplete beta function) $I(\alpha, \beta)$ is given by the expression

$$I_z(\alpha, \beta) = \frac{\mathfrak{B}(z; \alpha, \beta)}{\mathfrak{B}(1; \alpha, \beta)}, \quad \mathfrak{B}(z; \alpha, \beta) = \int_0^z t^{\alpha-1}(1-t)^{\beta-1} dt, \quad (3)$$

with $\alpha, \beta > 0$ and $z \in [0, 1]$. By definition, $I_z(\alpha, \beta)$ is strictly increasing, and is hence invertible. Its inverse, $I_z^{-1}(\alpha, \beta)$, is also strictly increasing.

Although usually the family of probability beta functions are described according to the parameters α and β , here it will be much more convenient to describe them according to their mean μ and standard deviation σ , as we will show later. It is well known that the expected value and the variance of the beta distribution is given by:

$$\mu = E[I(\alpha, \beta)] = \frac{\alpha}{\alpha + \beta}, \quad \sigma^2 = Var[I(\alpha, \beta)] = \frac{\alpha\beta}{(\alpha + \beta)^2(\alpha + \beta + 1)}. \quad (4)$$

If we solve α and β as a function of μ and σ^2 in the above equations we obtain:

$$\alpha(\mu, \sigma) = \mu \left(\frac{\mu(1-\mu)}{\sigma^2} - 1 \right), \quad \beta(\mu, \sigma) = (1-\mu) \left(\frac{\mu(1-\mu)}{\sigma^2} - 1 \right). \quad (5)$$

Thus, given values $\mu \in (0, 1)$ and $\sigma < \sqrt{\mu(1-\mu)}$, we have univocally determined parameters α and β such that the beta distribution $I(\alpha, \beta)$ has mean μ and standard deviation σ . Henceforth we will consider the beta distribution $\tilde{I}(\mu, \sigma)$ with respect to the parameters μ and σ , specifically:

$$\tilde{I}_z(\mu, \sigma) = I_z(\alpha(\mu, \sigma), \beta(\mu, \sigma)) \quad (6)$$

4.2. Beta RED Algorithm

In [11,17] the authors generalize the RED scheme by replacing the linear packet drop probability function $p_{\max} \cdot z(q_{\text{avg}})$ in (1) by the nonlinear function $p_{\max} \cdot I_{z(q_{\text{avg}})}(\alpha, \beta)$, where $0 \leq z \leq 1$ and $\alpha, \beta > 0$. Since $I_z(1, 1) = z$, we recover the classical RED scheme for $\alpha = \beta = 1$. The purpose of this generalization was to improve the stability properties by introducing the additional control parameters α and β . Following the analysis of Ranjan [38], in [12,18] the authors perform a detailed theoretical study of chaos, bifurcation diagrams, Lyapunov exponents and global stability robustness for different control parameters and fixed system parameters. Bifurcation diagrams are discussed for specific values of α and β in different scenarios, as well as biparametric sweeps of these parameters in which it was found that there are parameter regions in which the system performs very successfully in terms of stability and robustness.

However, for the design of our algorithms it is much more convenient to consider the biparametric family of packet drop probability functions with respect to the parameters μ and σ , since they have an intuitive meaning in statistical terms, namely, the mean and the standard deviation. In this case the packet drop probability function takes the expression

$$p(q_{\text{avg}}) = \begin{cases} 0 & \text{if } q_{\text{avg}} < q_{\min}, \\ 1 & \text{if } q_{\text{avg}} > q_{\max}, \\ p_{\max} \cdot \tilde{I}_{z(q_{\text{avg}})}(\mu, \sigma) & \text{otherwise.} \end{cases} \quad (7)$$

where $\mu \in (0, 1)$ and $\sigma < \sqrt{\mu(1-\mu)}$. From Equation (4), the classical RED ($\alpha = \beta = 1$) is recovered when $\mu = \frac{1}{2} = 0.5$ and $\sigma = \frac{1}{2\sqrt{3}} \approx 0.2886$.

As in the RED algorithm, the basic idea of BetaRED is to maintain the average queue length q_{avg} (calculated through the EWMA algorithm) within minimum and maximum thresholds q_{min} and q_{max} , but as close as possible to a predetermined target queue length that we denote by q_{target} . With this objective in mind, the next step is to provide concrete values of μ and σ for which a good balance between stability and performance of the algorithm is obtained. From this point of view, our selection of the mean is given as a function of the target queue length q_{target} and will be completely determined by means of the expression:

$$\mu = \frac{q_{target} - q_{min}}{q_{max} - q_{min}}. \quad (8)$$

With this choice, we match the mean of the packet drop probability function to the q_{target} value. Moreover, the standard deviation σ is a spread measure of the values of the distribution around μ . This means that the smaller σ is, the more concentrated the packet drop probability mass is around its mean μ (see Figure 2). Therefore, when the buffer starts to fill up and the average length of the buffer queue q_{avg} approaches q_{target} , the probability of packet drop will increase faster the smaller σ is. Moreover, the smaller σ is, the closer to q_{target} the system will stabilize. Consequently, the value of the standard deviation is a parameter that has to be adjusted in our BetaRED scheme. However, the fact that the value of the standard deviation verifies $0 < \sigma < \sqrt{\mu(1-\mu)} = \sigma_{max}$ causes a slight inconvenience since the user has to calculate the maximum value σ_{max} before selecting the value of σ . To avoid this inconvenience, the σ value is selected by mean of a scale factor θ that verifies $0 < \theta < 1$, being $\sigma = \theta \cdot \sqrt{\mu(1-\mu)}$, which facilitates the task to the user. Summarizing, in BetaRED, once the target queue length q_{target} is set, we have to adjust the parameters q_{min} , q_{max} , p_{max} , w and θ . The rest of the algorithm works exactly the same as RED. For the sake of completeness, we present an outline of the pseudo-code for BetaRED in Algorithm 1.

Algorithm 1 Pseudo-code outline for the Beta RED algorithm.

```

1: set tunable parameters:  $q_{target}, q_{min}, q_{max}, w, p_{max}, \theta$ 
2:  $\mu = \frac{q_{target} - q_{min}}{q_{max} - q_{min}}; \quad \sigma = \theta \cdot \sqrt{\mu(1-\mu)}$ ;
3: for each arriving packet do
4:   calculate new  $q_{avg} = (1-w) \cdot q_{avg} + w \cdot q_{cur}$ 
5:   if  $q_{avg} \leq q_{min}$  then
6:      $p = 0$ 
7:   else if  $q_{avg} \geq q_{max}$  then
8:      $p = 1$ 
9:   else
10:     $p = p_{max} \cdot \tilde{I}_{z(q_{avg})}(\mu, \sigma)$ 
11:   end if
12:   with probability  $p$ , drop arriving packet
13: end for

```

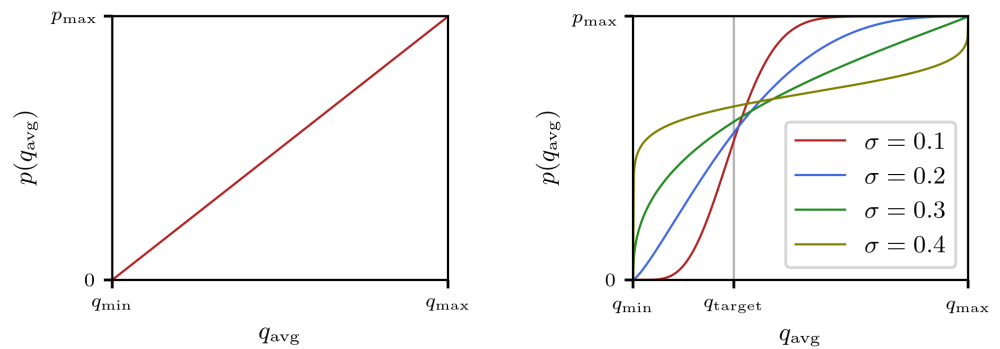


Figure 2. The lower the standard deviation σ , the higher the concentration of the packet drop probability mass around q_{target} .

4.3. Simulations for the BetaRED Algorithm

In this section we present some numerical simulations with the purpose of analyzing the average queue length and the performance of the BetaRED algorithm for some values of the control parameters. However, space limitation conditions us to present only a selection of a larger set of the simulations that have been carried out. The analysis of the results obtained will help us to design two new dynamic algorithms based on BetaRED, which will be described in Section 5.

In the first set of simulations (see Figure 3) we vary the parameter p_{max} for different numbers of nodes in the dumbbell topology shown in Figure 1, and analyze the average queue length in the last 125 s (half of the total simulation time). We contemplate only the second half of the total simulation considering the first half as a transition period, and thus obtain a better estimate of the equilibrium point. Two different scenarios are setting according to the parameters q_{min} y q_{max} . The first one is when $q_{min} = 0$ and $q_{max} = 1000$ (maximum buffer capacity). In the second scenario, we consider the minimum and maximum thresholds close to the target queue length $q_{target} = 250$, namely, $q_{min} = 200$ and $q_{max} = 400$.

We note that, to achieve a higher robustness in terms of the stability of the average queue length, it is convenient to select the value $p_{max} = 1$. This way, the average queue length stays closer to q_{target} . However, the performance may decrease for increasing values of the parameter p_{max} , which means that we should look for the lowest possible p_{max} parameter value that guarantees stability and increases performance. It is well known that the ARED algorithm [28] is based precisely on adjusting p_{max} to obtain a predetermined target average queue length. In Section 5 we will follow the same idea of ARED adapted to BetaRED.

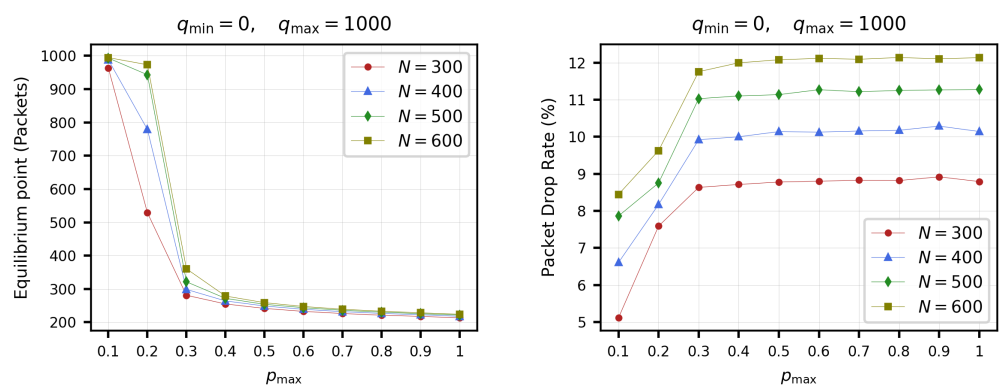


Figure 3. Cont.

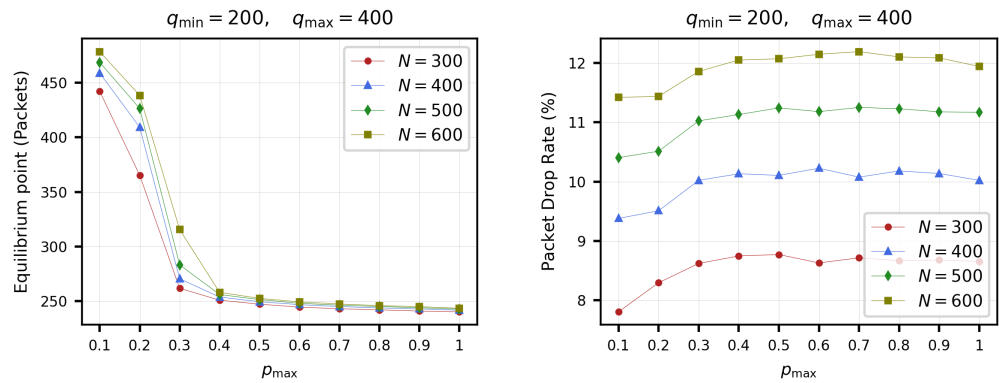


Figure 3. Comparison of the BetaRED algorithm according to the number N of FTP active flows and different values of the maximum probability threshold p_{max} . Different thresholds q_{min} and q_{max} are also considered. Other tunable parameters: $q_{target} = 250$ packets, $w = 0.1, \theta = 0.1$. Scenario 1 on the dumbbell topology (described in Section 3.1) is used. As p_{max} and/or the level of congestion increase, the equilibrium point of the average queue length gets closer to the prefixed q_{target} . On the other hand, if minimum and maximum thresholds q_{min} and q_{max} are closer to the target queue length q_{target} , this also results in the average queue length equilibrium point being closer to q_{target} .

In a second set of simulations (see Figure 4), we vary the θ parameter for different numbers of nodes in the dumbbell topology presented in Figure 1. Again, to estimate the equilibrium point, we calculate the average queue length in the last 125 s, i.e., in the second half of the total simulation time. As in the first set of simulations, we also consider the same two scenarios according to the minimum and maximum thresholds q_{min} and q_{max} . As expected (taking into account the interpretation of the standard deviation σ), better stability results are obtained as the value of θ (and hence σ) decreases. In the following, the default value of $\theta = 0.1$ will be used for simulations.

The discussion of the results when we vary the value of the weight parameter w is more intricate. Although we do not show concrete numerical simulations since the behavior is less predictable, it was observed that, as a general rule, increasing its value is accompanied by better performance, but stability gets worse. Nevertheless, we shall show that for the DBetaRED algorithm (described in Section 5.3) both high performance and good stability are obtained for a wide range of w values.

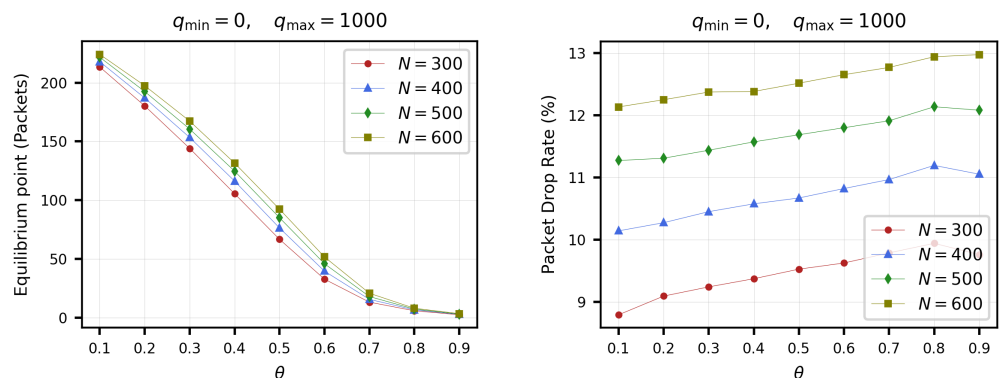


Figure 4. Cont.

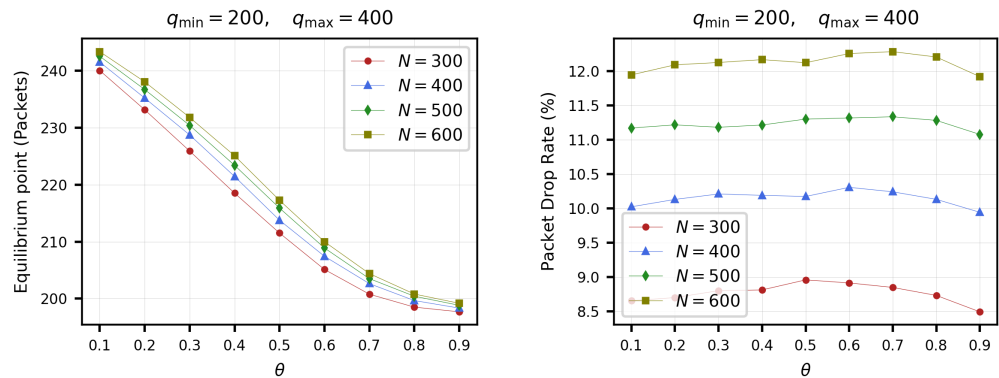


Figure 4. Comparison of the BetaRED algorithm according to the number N of FTP active flows and different values of θ . Different thresholds q_{\min} and q_{\max} are also considered. Other tunable parameters: $q_{\text{target}} = 250$ packets, $w = 0.1$, $p_{\max} = 1$. Scenario 1 on the dumbbell topology (described in Section 3.1) is used. As the standard deviation σ decreases and the level of congestion increases, the equilibrium point of the average queue length gets closer to the prefixed q_{target} . Similarly, if the minimum and maximum thresholds q_{\min} and q_{\max} are closer to the target queue length q_{target} , then the average queue length equilibrium point is closer to q_{target} .

5. Dynamic Algorithms Based on BetaRED

In the previous section, we designed a generic AQM based on the nonlinear beta probability distribution function. The major difficulty was, as in the case of RED and so many other schemes, the selection of the parameters to obtain good performance and stability around the target queue length. We found that the BetaRED algorithm tries to keep the average queue length close to the target queue length, but the level of congestion, the particular network conditions and the choice of appropriate values for the control parameters q_{\min} , q_{\max} , p_{\max} , w and the θ , can cause that the average queue length at equilibrium point undergoes substantial deviations from the desired q_{target} , as shown in Figures 3 and 4.

In this section, we propose two dynamical schemes based on BetaRED that are more accurate in reaching the target queue length and adapt satisfactorily to network traffic changes. Specifically, the objective is to perform dynamic parameter adjustments that correct the deviation of the equilibrium value of q_{avg} from the q_{target} value. The possibilities of acting on the parameters to correct these deviations are diverse. We propose two: the first one follows the same approach as the ARED algorithm [28], which is based on varying the value of p_{\max} to correct the difference from the equilibrium point of q_{avg} to q_{target} , whereas the second is to modify dynamically the target queue length q_{target} to correct the deviations.

5.1. The Adaptive Beta RED Algorithm (ABetaRED)

ABetaRED is inspired by the ARED algorithm (which in turn is based on RED), although there are some relevant changes due to the differences between the RED and BetaRED algorithms. For our ABetaRED algorithm, the tuning of the following two parameters is straightforward:

$$q_{\min} = 0, \quad q_{\max} = B. \tag{9}$$

The choices for q_{\min} and q_{\max} are based primarily on the numerical simulations for BetaRED in Section 4.3. It is observed that with the choices (9) an acceptable performance is achieved for a very wide range of parameters in various scenarios, while no substantial improvement in performance was obtained with the selection of other values. The control parameters Alpha and Beta determine the size of the increase and decrease steps of the maximum probability, respectively, with the range of admissible values being $0 < \text{Alpha}, \text{Beta} \leq 1$.

On the other hand, the tunable parameters to be set by the user are the target delay T_{target} , the scale factor θ and the averaging weight w . As in ARED, once the target delay is set, the target queue length can be estimated by

$$q_{\text{target}} = C \cdot T_{\text{target}} \quad (10)$$

where, as in Equation (9), C is the link capacity measured in packets per second, and T_{target} is measured in seconds. A pseudo-code for ABetaRED is outlined in Algorithm 2.

Algorithm 2 Pseudo-code outline for the Adaptive Beta RED algorithm. By default, Alpha = Beta = 1 and $T_{\text{update}} = 0.5$ s, were set in all simulations.

```

1: set control parameters:  $T_{\text{target}}, w, \theta$ 
2:  $q_{\text{min}} = 0; \quad q_{\text{max}} = B; \quad p_{\text{max}} = 0.5; \quad q_{\text{target}} = C \cdot T_{\text{target}};$ 
3:  $\mu = \frac{q_{\text{target}} - q_{\text{min}}}{q_{\text{max}} - q_{\text{min}}}; \quad \sigma = \theta \cdot \sqrt{\mu(1 - \mu)};$ 
4: for each arriving packet do
5:   calculate new  $q_{\text{avg}} = (1 - w) \cdot q_{\text{avg}} + w \cdot q_{\text{cur}}$ 
6:   for every interval time  $T_{\text{update}}$  do
7:     if  $q_{\text{avg}} < q_{\text{target}}$  then
8:       decrease maximum probability:
9:        $p_{\text{max}} = \max \left[ 0.01, p_{\text{max}} \cdot \text{Alpha} \cdot \left( 1 - \frac{q_{\text{target}} - q_{\text{avg}}}{q_{\text{max}} - q_{\text{min}}} \right) \right]$ 
10:     else if  $q_{\text{avg}} > q_{\text{target}}$  then
11:       increase maximum probability:
12:        $p_{\text{max}} = \min \left[ 0.99, p_{\text{max}} + \text{Beta} \cdot p_{\text{max}} \cdot (1 - p_{\text{max}}) \cdot \frac{q_{\text{avg}} - q_{\text{target}}}{q_{\text{max}} - q_{\text{min}}} \right]$ 
13:     end if
14:   end for
15:   update  $p = p_{\text{max}} \cdot \tilde{I}_z(q_{\text{avg}})(\mu, \sigma)$ 
16:   with probability  $p$ , drop arriving packet
17: end for

```

5.2. The Dynamic Beta RED Algorithm (DBetaRED)

The idea of the DBetaRED algorithm is, once the target queue delay T_{target} is fixed and the target queue length is estimated as in ABetaRED by Equation (10), we introduce another dynamic parameter called virtual target queue length $\tilde{q}_{\text{target}}$ to correct for deviations between the average queue length q_{avg} and the actual target queue length q_{target} . According to the value of $\tilde{q}_{\text{target}}$, the value of the mean μ (and thus the standard deviation $\sigma = \theta \cdot \sqrt{\mu(1 - \mu)}$) of the drop probability function are updated dynamically for each time interval T_{update} previously established. An outline of the pseudo-code of this new algorithm is given in Algorithm 3.

Algorithm 3 Pseudo-code outline for the Dynamic Beta RED algorithm. By default, Alpha = Beta = 1 and $T_{\text{update}} = 0.5$ s, were set in all simulations.

```

1: set control parameters:  $T_{\text{target}}, w, \theta$ 
2:  $q_{\text{min}} = 0; \quad q_{\text{max}} = B; \quad p_{\text{max}} = 1;$ 
3:  $q_{\text{target}} = C \cdot T_{\text{target}}$ 
4:  $\tilde{q}_{\text{target}} = q_{\text{target}}; \quad \mu = \frac{\tilde{q}_{\text{target}} - q_{\text{min}}}{q_{\text{max}} - q_{\text{min}}}; \quad \sigma = \theta \cdot \sqrt{\mu(1 - \mu)};$ 
5: for each arriving packet do
6:   calculate new  $q_{\text{avg}} = (1 - w) \cdot q_{\text{avg}} + w \cdot q_{\text{cur}}$ 
7:   for every interval time  $T_{\text{update}}$  do
8:     calculate  $\delta = \mu \cdot (1 - \mu) \cdot (q_{\text{target}} - q_{\text{avg}})$ 
9:     if  $q_{\text{avg}} < q_{\text{target}}$  then
10:      increase virtual target queue length:  $\tilde{q}_{\text{target}} = \min[q_{\text{max}} - 1, \tilde{q}_{\text{target}} + \text{Alpha} \cdot$ 
11:       $\delta]$ 
12:     else if  $q_{\text{avg}} > q_{\text{target}}$  then
13:      decrease virtual target queue length:  $\tilde{q}_{\text{target}} = \max[q_{\text{min}} + 1, \tilde{q}_{\text{target}} + \text{Beta} \cdot \delta]$ 
14:     end if
15:     update  $\mu = \frac{\tilde{q}_{\text{target}} - q_{\text{min}}}{q_{\text{max}} - q_{\text{min}}}; \quad \sigma = \theta \cdot \sqrt{\mu(1 - \mu)};$ 
16:   end for
17:   update  $p = p_{\text{max}} \cdot \tilde{I}_z(q_{\text{avg}})(\mu, \sigma)$ 
18:   with probability  $p$ , drop arriving packet
19: end for

```

In this case we make a straightforward selection of the following parameters:

$$q_{\text{min}} = 0, \quad q_{\text{max}} = B, \quad p_{\text{max}} = 1.$$

The choice of the minimum and maximum thresholds is based on the same reason as we have stated for the ABetaRED algorithm. Regarding p_{max} , it is well known that the choice of p_{max} in RED is linked to the traffic load in the network, where the higher the traffic load the higher the value of p_{max} should be. One of the advantages of BetaRED over RED is that we can control the increase of the drop probability function around the target queue length (in a smoothly or sharply way, but always continuously) by means of the θ parameter. Thus, the choice of $p_{\text{max}} = 1$ is the most reasonable in order to avoid discontinuities in the drop probability function. Moreover, the numerical results of Figure 3 show that a higher value of p_{max} implies better performance. The control parameters Alpha and Beta determine the size of the increase and decrease steps of the virtual target queue length, respectively, with the range of admissible values being $0 < \text{Alpha}, \text{Beta} \leq 1$. The tunable parameters to be set by the user in this case are the target delay T_{target} , the scalar factor θ and the averaging weight w .

5.3. Simulations

This section presents the results and discussion of the numerical simulations performed to compare ABetaRED (Section 5.1) and DBetaRED (Section 5.2), as well as these algorithms with the selection of the other AQM algorithms described in Section 2. The simulations are performed in several scenarios according to different parameters and levels of congestion, including slight, moderate and abrupt variations in the number of nodes in order to verify the robustness of the proposed AQM algorithms. However, space limitation conditions us to present only a selection of the most important features regarding the behavior of the above algorithms.

The first comparison we carried out was between the ABetaRED and DBetaRED algorithms (see Figures 5 and 6) when we progressively increase the parameter θ (and hence the standard deviation σ). Figure 5 shows the result of the simulations for a constant level of congestion (Scenario 1 described in Section 3.1), while Figure 6 shows the result for a changing level of congestion (Scenario 2 described in Section 3.1). It is observed that, in both algorithms, the throughput is quite unpredictable for different θ parameters. However, the jitter is lower the smaller the value of the θ parameter, i.e., for smaller σ values.

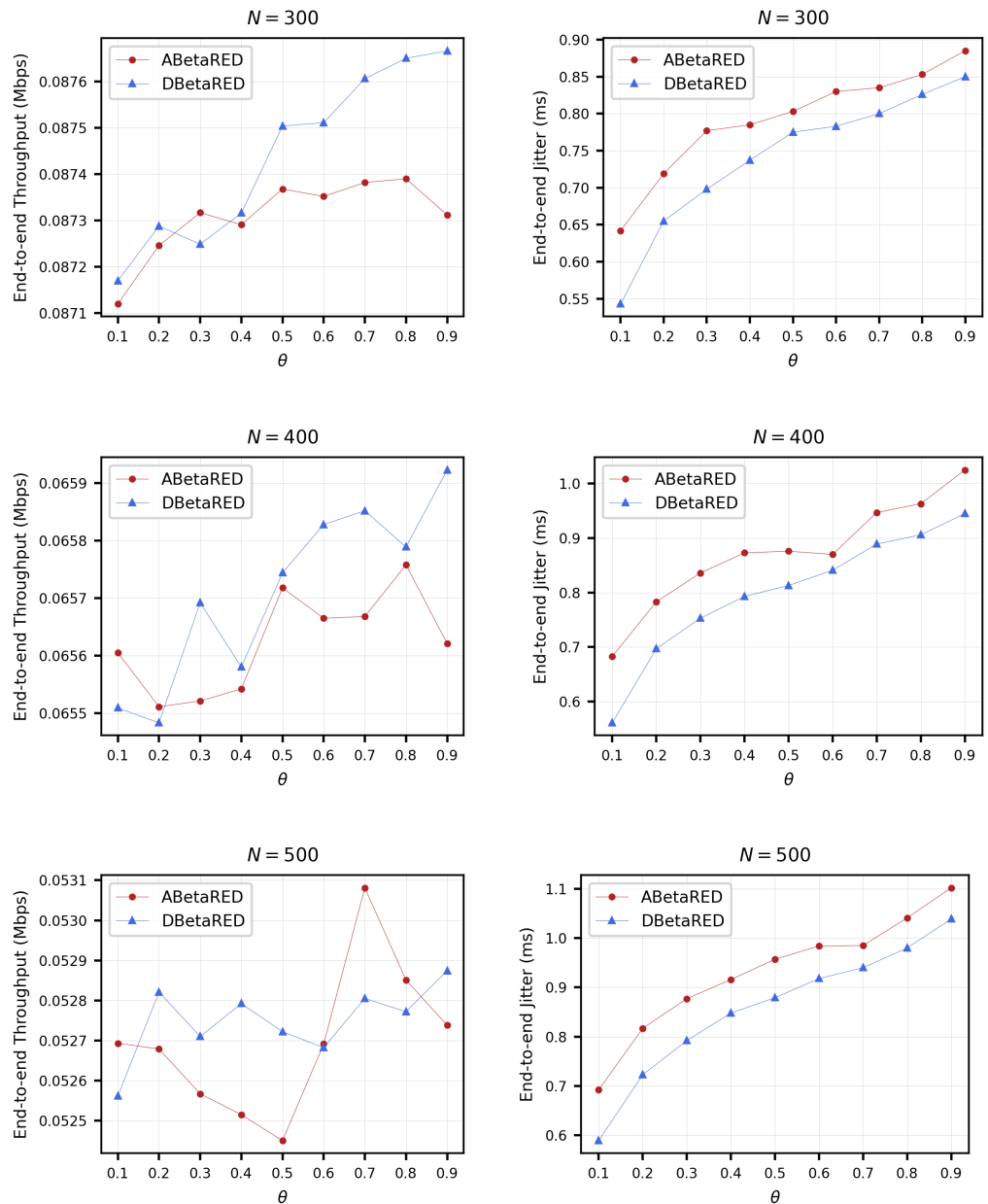


Figure 5. Comparison between ABetaRED and DBetaRED algorithms according to the standard deviation σ . Different levels of congestion are also considered by means of a constant number N of active flows. Other tuning parameters: $T_{\text{target}} = 40$ ms ($q_{\text{target}} = 250$ packets) and $w = 0.1$. Scenario 1 on the dumbbell topology (described in Section 3.1) is used. Overall, the DBetaRED algorithm exhibits better performance, especially for the end-to-end jitter metric.

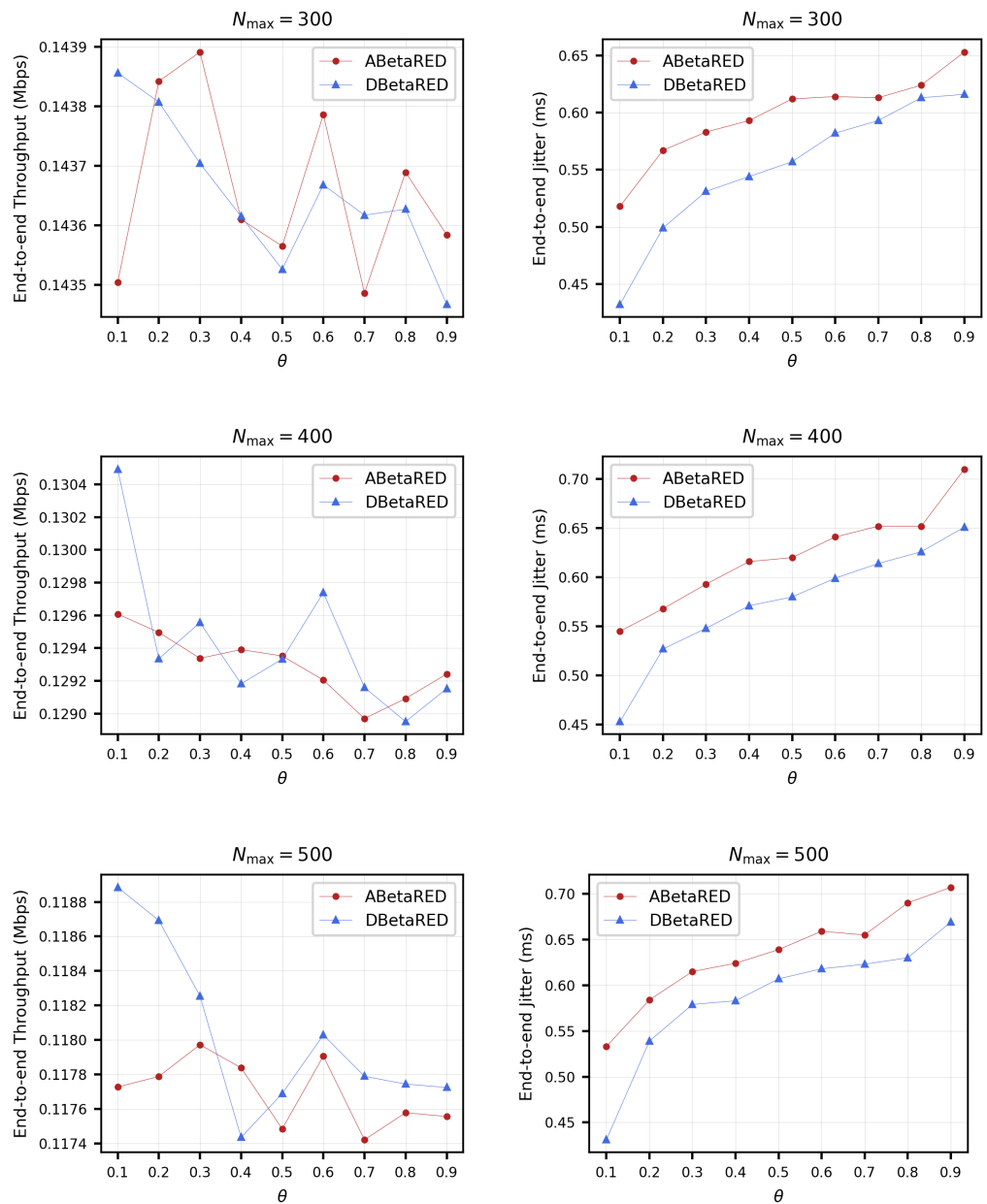


Figure 6. Comparison between the ABetaRED and DBetaRED algorithms according to the parameter θ . Different levels of congestion are also considered, but, unlike in Figure 5, the number N of active flows varies dynamically with time. Other tuning parameters: $T_{target} = 40$ ms ($q_{target} = 250$ packets) and $w = 0.1$. Scenario 2 on the dumbbell topology (described in Section 3.1) is used. As in the non-dynamic scenario (Figure 5), the DBetaRED algorithm performs better than the ABetaRED algorithm.

Figure 7 shows the performance of DBetaRED for different levels of congestion according to different values of the weight parameter w . It can be seen that whatever the level of congestion, the choice of a sufficiently high w parameter guarantees a satisfactory performance. However, the most appropriate value of w according to the scenario is difficult to estimate, as it may vary depending on the network topology, the level of congestion, etc. In any case, it was observed that the qualitative behavior does not change from one scenario to another.

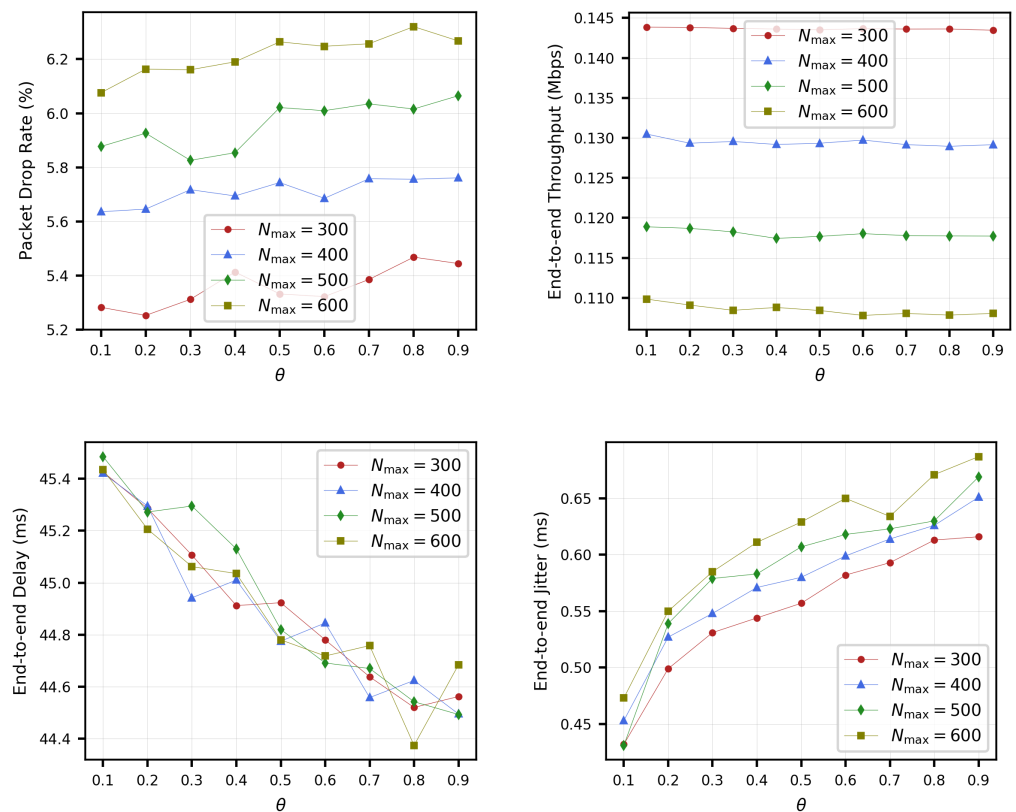


Figure 7. Performance comparison of the Dynamic Beta RED algorithm according to a number N of active flows that vary dynamically over time and different values of the weight parameter w . Other tunable parameters: $T_{target} = 40$ ms ($q_{target} = 250$ packets) and $\theta = 0.1$. Scenario 2 on the dumbbell topology (described in Section 3.1) is used. We observe that the trend is for the performance of the DBetaRED algorithm to increase as the value of w increases.

It is of special interest when a comparison is made with other related AQM schemes. The common denominator of all the selected dynamic algorithms to be compared with ABetaRED and DBetaRED is the need to set the target delay parameter T_{target} . However, the mechanism of action of each of the AQM algorithms according to this parameter is different. CoDel starts dropping packets when the queue delay remains above the target delay for a certain time. PIE continuously updates its probability of dropping packets according to the difference between the current queue delay and the target delay. ARED does not act directly as a function of the target delay, but rather as a function of a target queue length estimated via the target delay. Both ABetaRED and DBetaRED algorithms act in the same way as ARED, namely, to achieve a given predetermined T_{target} , a target queue length given by $q_{target} = C \cdot T_{target}$ is estimated, where C is measured in packets per second.

As can be seen in Figure 8, when fluctuations and congestion level grow, all performance metrics of the DBetaRED algorithm outperform all other AQM algorithms. Moreover, in Figure 9 a greater stability around the target value of the average queue length is also observed.

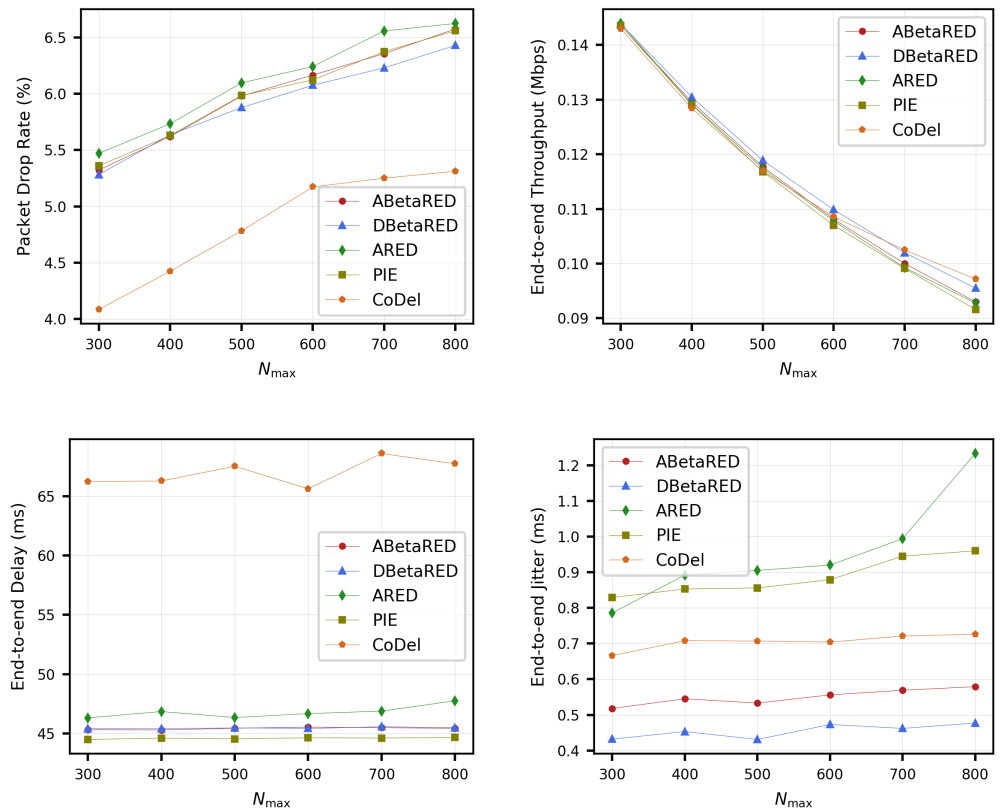


Figure 8. Performance comparison between related AQM algorithms when considering different levels of congestion by varying dynamically the number N of active flows. Other tuning parameters: $T_{target} = 40$ ms for all AQM algorithms; $\theta = 0.1$ and $w = 0.1$ for ABetaRED and DBetaRED. Scenario 2 on the dumbbell topology (described in Section 3.1) is used. The performance of the DBetaRED algorithm outperforms the other algorithms as the congestion level increases.

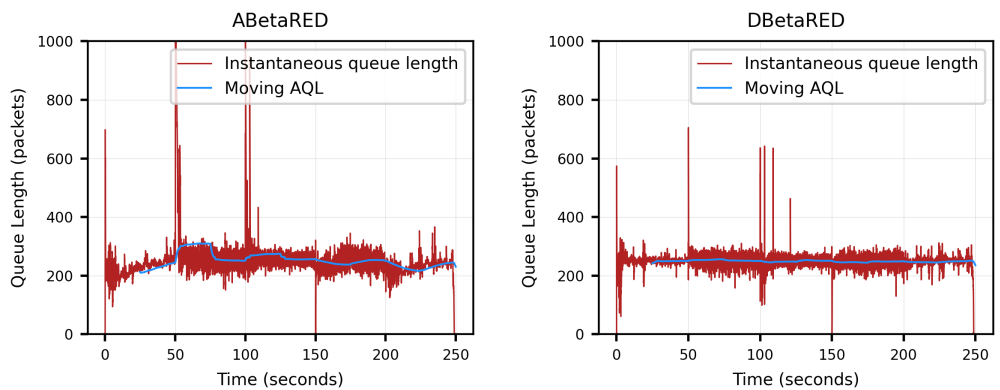


Figure 9. Cont.

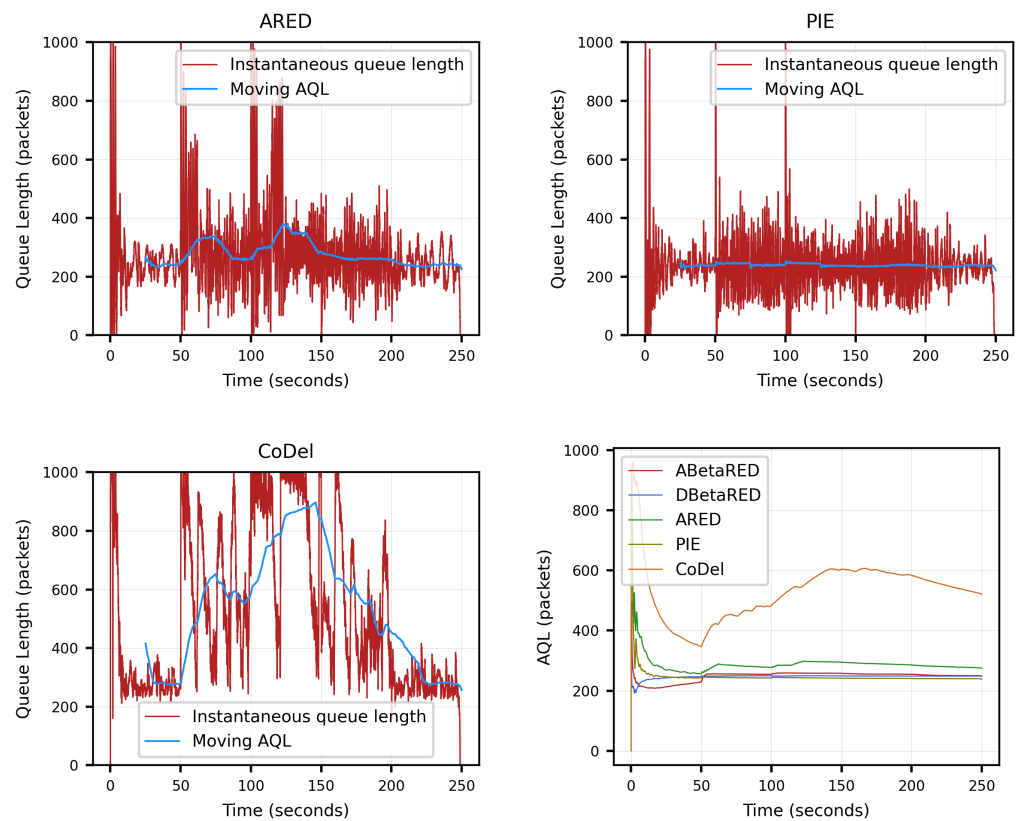


Figure 9. Stability comparison between related AQM algorithms when considering different levels of congestion by varying dynamically the number N of active flows with $N_{\max} = 800$. Other tuning parameters: $T_{\text{target}} = 40$ ms for all AQM algorithms; $\theta = 0.1$ and $w = 0.1$ for ABetaRED and DBetaRED. Scenario 2 on the dumbbell topology (described in Section 3.1) is used. The bottom right panel shows a comparison of the average queue length (for the total simulation time) of all selected AQM algorithms. The other panels illustrate the instantaneous queue length pattern, together with its moving average queue length (of the last 25 s) for each of the AQM algorithms. Among all the algorithms, DBetaRED exhibits the highest stability.

6. Conclusions

In this work we propose new RED-type AQM algorithms, which we call BetaRED, ABetaRED and DBetaRED. Using a simple dumbbell topology and applying different levels of congestion (both constant and variable), we have analyzed and evaluated its performance comparing it with the ARED, CoDel and PIE algorithms through different performance metrics such as the average queue length, the packet queue rate, the end-to-end throughput, the end-to-end delay (latency) and the delay jitter.

We have shown that the new BetaRED algorithm is a simple, flexible and robust mechanism, which provides good stability and performance over a very wide range of parameters. However, BetaRED needs parameter tuning according to the network characteristics and congestion scenario. In order to reduce the number of tuning parameters, the dynamic algorithms ABetaRED and DBetaRED (based on BetaRED) have been proposed and compared with benchmark algorithms such as ARED, CoDel and PIE, obtaining comparable results and even outperforming them in certain scenarios. DBetaRED stabilizes queueing length, further improves throughput and reduces packet drops, compared to other representative AQM algorithms, most notably in high-load congestion scenarios. Although the simulations carried out to obtain these conclusions have been numerous, the testing possibilities are enormous, and so there is still a lot of work to be done in this regard. Additionally, we share the view of [39] that the choice of the TCP-AQM couple to

adopt is crucial, i.e., the most effective congestion control would occur when AQM and TCP work together in a shared bottleneck. Consequently, we continue studying heterogeneous scenarios of our AQM algorithms with variants of TCP different to those studied in this paper. Specially, it would be interesting to evaluate the behavior of the proposed AQM algorithms with the bottleneck bandwidth and round-trip time (BBR) congestion control algorithm published by Google in 2016. Unlike packet loss-based TCP variants (such as Cubic and Reno), BBR is designed to obtain a high performance coupled with low bottleneck buffer occupancy, reducing packet loss and minimizing delay by means of its end-to-end action. Thus, the effect of the proposed AQM algorithms is likely to be negligible in this case, or even negatively affect overall performance.

Last but not least, the proposed BetaRED-type algorithms are not claimed to provide a general optimal solution, since the optimality of the parameters will depend on the objective set and the characteristics of the scenarios. Therefore, one of our future research topics will be the search for optimal settings of BetaRED parameters in different concrete network scenarios, for which an in-depth analysis of mathematical models derived from the BetaRED algorithm will be necessary.

Author Contributions: Conceptualization, A.G., M.A.M., J.M.A., O.M.-B. and J.V.; methodology, A.G., M.A.M. and O.M.-B.; software, A.G. and M.A.M.; writing—review, A.G., M.A.M., J.M.A., O.M.-B. and J.V.; content discussion and final version, A.G., M.A.M., J.M.A., O.M.-B. and J.V. All authors have read and agreed to the published version of the manuscript.

Funding: This research work was financially supported by Agencia Estatal de Investigación, Spain, grant PID2019-108654GB-I00/AEI/10.13039/501100011033. A. Giménez, J.M. Amigó and J. Valero have also been partially supported by Generalitat Valenciana, Spain, grant PROMETEO/2021/063.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Gettys, J. Bufferbloat: Dark Buffers in the Internet. *IEEE Internet Comput.* **2011**, *15*, 96. [[CrossRef](#)]
2. Al-Saadi, R.; Armitage, G.; But, J.; Branch, P. A Survey of Delay-Based and Hybrid TCP Congestion Control Algorithms. *IEEE Commun. Surv. Tuts.* **2019**, *21*, 3609–3638. [[CrossRef](#)]
3. Polese, M.; Chiariotti, F.; Bonetto, E.; Rigotto, F.; Zanella, A.; Zorzi, M. A Survey on Recent Advances in Transport Layer Protocols. *IEEE Commun. Surv. Tuts.* **2019**, *21*, 3584–3608. [[CrossRef](#)]
4. Floyd, S.; Jacobson, V. Random early detection gateways for congestion avoidance. *IEEE/ACM Trans. Netw.* **1993**, *1*, 397–413. [[CrossRef](#)]
5. Shorten, R.; King, C.; Wirth, F.; Leith, D. Modelling TCP congestion control dynamics in drop-tail environments. *Automatica* **2007**, *43*, 441–449. [[CrossRef](#)]
6. Chen, L.; Wang, X.; Han, Z. Controlling chaos in Internet congestion control model. *Chaos Solitons Fractals* **2004**, *21*, 81–91. [[CrossRef](#)]
7. Liu, F.; Guan, Z.H.; Wang, H.O. Controlling bifurcations and chaos in TCP-UDP-RED. *Nonlinear Anal. Real World Appl.* **2010**, *11*, 1491–1501. [[CrossRef](#)]
8. Xiao, M.; Zheng, W.X.; Cao, J. Bifurcation control of a congestion model via state feedback. *Int. J. Bifurc. Chaos* **2013**, *23*, 1330018. [[CrossRef](#)]
9. Zhang, S.; Xu, J.; Chung, K.w. On the stability and multi-stability of a TCP/RED congestion control model with state-dependent delay and discontinuous marking function. *Commun. Nonlinear Sci. Numer. Simul.* **2015**, *22*, 269–284. [[CrossRef](#)]
10. Pei, L.; Wang, S. Dynamics and the periodic solutions of the delayed non-smooth Internet TCP-RED congestion control system via HB-AFT. *Appl. Math. Comput.* **2019**, *361*, 689–702. [[CrossRef](#)]
11. Duran, G.; Valero, J.; Amigó, J.M.; Giménez, A.; Martínez Bonastre, O. Bifurcation analysis for the Internet congestion. In Proceedings of the IEEE INFOCOM 2019—IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), Paris, France, 29 April–2 May 2019; pp. 1073–1074. [[CrossRef](#)]
12. M. Amigó, J.; Duran, G.; Giménez, A.; Martínez-Bonastre, O.; Valero, J. Generalized TCP-RED dynamical model for Internet congestion control. *Commun. Nonlinear Sci. Numer. Simul.* **2020**, *82*, 105075. [[CrossRef](#)]

13. Amigó, J.M.; Duran, G.; Giménez, A.; Valero, J.; Martínez Bonastre, O. Modeling a New AQM Model for Internet Chaotic Behavior Using Petri Nets. *Appl. Sci.* **2021**, *11*, 5877. [CrossRef]
14. Pei, L.; Wu, F. Periodic solutions, chaos and bi-stability in the state-dependent delayed homogeneous Additive Increase and Multiplicative Decrease/Random Early Detection congestion control systems. *Math. Comput. Simul.* **2021**, *182*, 871–887. [CrossRef]
15. Adams, R. Active Queue Management: A Survey. *IEEE Commun. Surv. Tuts.* **2013**, *15*, 1425–1476. [CrossRef]
16. Brandauer, C.; Iannaccone, G.; Diot, C.; Ziegler, T.; Fdida, S.; May, M. Comparison of tail drop and active queue management performance for bulk-data and Web-like Internet traffic. In Proceedings of the Sixth IEEE Symposium on Computers and Communications, Hammamet, Tunisia, 5 July 2001; pp. 122–129. [CrossRef]
17. Duran, G.; Valero, J.; Amigó, J.M.; Giménez, A.; Martínez Bonastre, O. Stabilizing Chaotic Behavior of RED. In Proceedings of the 2018 IEEE 26th International Conference on Network Protocols (ICNP), Cambridge, UK, 25–27 September 2018; pp. 241–242. [CrossRef]
18. Amigó, J.M.; Giménez, A.; Martínez Bonastre, O.; Valero, J. Internet congestion control: From stochastic to dynamical models. *Stochastics Dyn.* **2021**, *21*, 2140009. [CrossRef]
19. Feng, C.W.; Huang, L.F.; Xu, C.; Chang, Y.C. Congestion Control Scheme Performance Analysis Based on Nonlinear RED. *IEEE Syst. J.* **2017**, *11*, 2247–2254. [CrossRef]
20. Adamu, A.; Shorgin, V.; Melnikov, S.; Gaidamaka, Y. Flexible Random Early Detection Algorithm for Queue Management in Routers. In *Distributed Computer and Communication Networks*; Vishnevskiy, V.M., Samouylov, K.E., Kozyrev, D.V., Eds.; Springer International Publishing: Cham, Switzerland, 2020; Volume 12563, pp. 196–208. [CrossRef]
21. Patel, S.; Karmeshu. A New Modified Dropping Function for Congested AQM Networks. *Wirel. Pers. Commun.* **2019**, *104*, 37–55. [CrossRef]
22. Xiong, N.; Yang, L.; Yang, Y.; Défago, X.; He, Y. A novel numerical algorithm based on self-tuning controller to support TCP flows. *Math. Comput. Simul.* **2008**, *79*, 1178–1188. [CrossRef]
23. Bhatnagar, S.; Patel, S.; Karmeshu. A stochastic approximation approach to active queue management. *Telecommun. Syst.* **2018**, *68*, 89–104. [CrossRef]
24. Sharma, N.; Rajput, S.S.; Dwivedi, A.K.; Shrimali, M. P-RED: Probability Based Random Early Detection Algorithm for Queue Management in MANET. In *Advances in Computer and Computational Sciences*; Bhatia, S.K., Mishra, K.K., Tiwari, S., Singh, V.K., Eds.; Springer: Singapore, 2018; pp. 637–643. [CrossRef]
25. Abu-Shareha, A.A. Enhanced Random Early Detection using Responsive Congestion Indicators. *Int. J. Adv. Comput. Sci. Appl.* **2019**, *10*. [CrossRef]
26. Baklizi, M. Weight Queue Dynamic Active Queue Management Algorithm. *Symmetry* **2020**, *12*, 2077. [CrossRef]
27. Feng, W.C.; Kandlur, D.; Saha, D.; Shin, K. A self-configuring RED gateway. In Proceedings of the IEEE INFOCOM '99. Conference on Computer Communications, Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No. 99CH36320), New York, NY, USA, 21–25 March 1999; Volume 3, pp. 1320–1328. ISSN: 0743-166X. [CrossRef]
28. Floyd, S.; Gummadi, R.; Shenker, S. Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management; Technical Report. 2001. Available online: <http://www.icir.org/floyd/papers/adaptiveRed.pdf> (accessed on 24 July 2021).
29. Nichols, K.; Jacobson, V. Controlling queue delay. *Commun. ACM* **2012**, *55*, 42–50. [CrossRef]
30. Nichols, K.; Jacobson, V.; McGregor, E.A.; Iyengar, E.J. Controlled Delay Active Queue Management; RFC 8289; Controlled Delay Active Queue Management. 2018. Available online: <https://www.rfc-editor.org/info/rfc8289> (accessed on 24 July 2021).
31. Pan, R.; Natarajan, P.; Piglione, C.; Prabhu, M.S.; Subramanian, V.; Baker, F.; VerSteeg, B. PIE: A lightweight control scheme to address the bufferbloat problem. In Proceedings of the 2013 IEEE 14th International Conference on High Performance Switching and Routing (HPSR), Taipei, Taiwan, 8–11 July 2013; pp. 148–155. ISSN: 2325-5560. [CrossRef]
32. Schwarzkopf, F.; Veith, S.; Menth, M. Performance Analysis of CoDel and PIE for Saturated TCP Sources. In Proceedings of the 2016 28th International Teletraffic Congress (ITC 28), Würzburg, Germany, 12–16 September 2016; pp. 175–183. [CrossRef]
33. Pan, R.; Natarajan, P.; Baker, F.; White, G. Proportional Integral Controller Enhanced (PIE): A Lightweight Control Scheme to Address the Bufferbloat Problem; Technical Report RFC8033; RFC Editor. 2017. Available online: <https://www.rfc-editor.org/info/rfc8033> (accessed on 24 July 2021).
34. Hollot, C.; Misra, V.; Towsley, D.; Gong, W.-B. On designing improved controllers for AQM routers supporting TCP flows. In Proceedings of the IEEE INFOCOM 2001, Conference on Computer Communications, Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No. 01CH37213), Anchorage, AK, USA, 22–26 April 2001; Volume 3, pp. 1726–1734. [CrossRef]
35. Kuhn, N.; Ros, D.; Bagayoko, A.B.; Kulatunga, C.; Fairhurst, G.; Khademi, N. Operating ranges, tunability and performance of CoDel and PIE. *Comput. Commun.* **2017**, *103*, 74–82. [CrossRef]
36. Baker, E.F.; Fairhurst, E.G. IETF Recommendations Regarding Active Queue Management. 2015. Available online: <https://www.rfc-editor.org/info/rfc7567> (accessed on 24 July 2021).
37. nsnam. Network Simulator 3. 2011. Available online: <https://www.nsnam.org/> (accessed on 24 July 2021).

38. Ranjan, P.; Abed, E.H.; La, R.J. Nonlinear Instabilities in TCP-RED. *IEEE/ACM Trans. Netw.* **2004**, *12*, 1079–1092. [[CrossRef](#)]
39. Grazia, C.A.; Patriciello, N.; Klapez, M.; Casoni, M. A cross-comparison between TCP and AQM algorithms: Which is the best couple for congestion control? In Proceedings of the 2017 14th International Conference on Telecommunications (ConTEL), Zagreb, Croatia, 28–30 June 2017; pp. 75–82. [[CrossRef](#)]