A Comparison of Load-based and Queue-based Active Queue Management Algorithms

Minseok Kwon and Sonia Fahmy

Dept. of Computer Science, Purdue University
West Lafayette, IN 47906–1398, USA

ABSTRACT

A number of active queue management algorithms have been studied since Random Early Detection (RED) was first introduced in 1993. While analytical and experimental studies have debated whether dropping/marking should be based on average or instantaneous queue length or, alternatively, based on input and output rates (or queue length slope), the merits and drawbacks of the proposed algorithms, and the effect of load-based versus queue-based control have not been adequately examined. In particular, only RED has been tested in realistic configurations and in terms of user metrics, such as response times and average delays. In this paper, we examine active queue management (AQM) that uses both load and queuing delay to determine its packet drop/mark probabilities. This class of algorithms, which we call load/delay controllers (LDC), has the advantage of controlling the queuing delay as well as accurately anticipating incipient congestion. We compare LDC to a number of well-known active queue management algorithms including RED, BLUE, FRED, SRED, and REM in configurations with multiple bottlenecks, round trip times and bursty Web traffic. We evaluate each algorithm in terms of Web response time, delay, packet loss, and throughput, in addition to examining algorithm complexity and ease of configuration. Our results demonstrate that load information, along with queue length, can aid in making more accurate packet drop/mark decisions that reduce the Web response time.

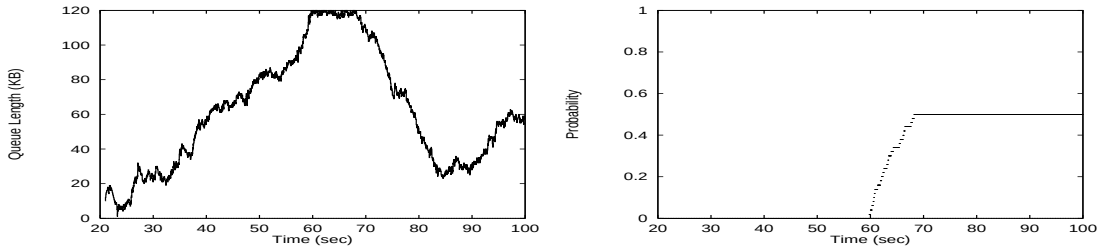
Keywords: B uffer Management, Active Queue Management (AQM), Congestion Control, Random Early Detection (RED)

1. INTRODUCTION

Active queue management in Internet routers can improve application goodput and response times by detecting congestion early and improving fairness among flows. The Internet Research Task Force (IRTF) recommends the deployment of router-based congestion control, especially Random Early Detection (RED), in RFC 2309.[1] The main goal of RED[2] is to drop/mark packets before buffer overflow in order to give early warning to sources, avoid TCP synchronization and bias against bursty connections, and punish misbehaving sources.

Since RED development in 1993, a number of RED variants have appeared including Flow RED (FRED),[3] Stabilized RED (SRED)[4] and BLUE.[5] Although RED and its variants improve performance over simple drop tail queues, they exhibit two main weaknesses. First, it is difficult to configure their parameters. Second, they are insensitive to current queue arrival and drain rates, and rely on long term queue length averages. Recently, a number of algorithms including REM,[6–8] AVQ[9, 10] and the PI controller[11] have been proposed to remedy the insensitivity to load problem. While these algorithms are effective from a control-theoretic perspective, they have not been carefully tested in realistic Internet-like configurations with bursty traffic and variable delays, and especially without Explicit Congestion Notification (ECN) marking[12] which they recommend or require. Some are difficult to configure, and some are complex to implement. While many studies have debated whether dropping/marking should be based on queue length or on input and output rates (or alternatively the queue length slope over time), the merits and drawbacks of various algorithms and the effect of load-based versus queue-based control has not been adequately examined. No thorough comparisons have been made of all algorithms in terms of user-perceived metrics such Web response time (as is done with RED in[13]).

This research has been sponsored in part by the Purdue Research Foundation and the Schlumberger Foundation technical merit award. The authors can be reached through e-mails: {kwonm,fahmy}@cs.purdue.edu, telephone:+1-765-494-6183, fax: +1-765-494-0739



(a) Queue length in BLUE

(b) Packet drop probability in BLUE

Figure 1. BLUE drop relies on queue overflow/underflow history

This paper has two main contributions. First, we examine a class of active queue management algorithms which incorporate both load and queuing delay metrics (we call this class of algorithms Load/Delay Controllers (LDC)). LDC uses a load factor, computed as the ratio of the queue arrival and drain rates, in addition to the queuing delay to determine the drop/mark probability (similar to our ERICA ATM-ABR algorithm[14, 15]). The objective of LDC is to keep the average queuing delay under a specified target, thus reducing Web response time, without significantly affecting application throughput and link utilization.

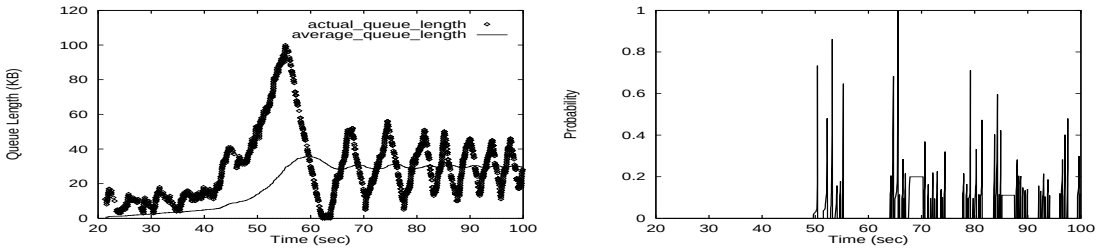
Our second contribution is to compare LDC with a number of well-known active queue management algorithms including RED, BLUE, FRED, SRED, and REM in realistic network configurations with bursty Web traffic. We evaluate each algorithm in terms of response time, packet delay, packet loss rate and throughput, in addition to examining its complexity and ease of configuration. Our simulation results, especially in terms of Web response time demonstrate the importance of the load factor along with the queue length as congestion indicators. We classify the schemes as primarily load-based or primarily queue-based.

We focus on drop and drop probability in this paper, though "drop probability" throughout the paper (except for results) can be replaced by "marking probability." We defer the study of ECN-capable networks to future work, since some Internet firewalls are not yet ECN-aware.[16] The remainder of this paper is organized as follows. Section 2 discusses related work. Sections 3 and 4 examine load, queue length and queuing delay as congestion indicators, and discuss methods for combining them. Section 5 analyzes the performance of the different algorithms in a number of network configurations. Finally, section 6 summarizes our conclusions and discusses future work.

2. BACKGROUND AND RELATED WORK

Active queue management has been extensively researched since the early 1990s. RED[2] detects congestion at network routers in order to promptly notify sources to reduce their rates, punish misbehaving flows without being biased against bursty traffic, and avoid global synchronization. RED maintains a long term average of the queue length (buffer occupancy) of the router, and randomly drops packets in proportion to this buffer occupancy value. RED, however, is difficult to configure since it relies on the buffer thresholds max_{th}, min_{th}, the weight used in averaging, and the maximum drop probability max_p as parameters. RED performance is extremely sensitive to these parameters. It is difficult to tune the parameters to reflect whether short queuing delay is more important than link utilization or vice versa.

Other weaknesses of RED have been reported and several approaches to overcoming them have later been proposed. Lin and Morris[3] proposed Flow RED (FRED) to overcome problems observed when RED is evaluated over three different traffic types categorized according to their response behaviors to congestion. When a mix of these traffic types share a link, RED allows unfair bandwidth sharing since RED imposes the same loss rate on all flows regardless of their bandwidths. To remedy this weakness, FRED uses per-active-flow accounting to impose on each flow a loss rate depending on the flow buffer occupancy. Stabilized RED (SRED)[4] is another variant of RED where the packet marking probability depends on the instantaneous buffer occupancy as well



(a) Queue length in RED

(b) Packet drop probability in RED

Figure 2. RED drop relies solely on average buffer occupancy

as on the estimated number of active flows. This feature helps SRED stabilize its buffer occupancy at a level independent of the number of active connections.

BLUE[5] uses packet loss and link utilization rather than the average queue length. It increments the packet drop probability when packet loss occurs and decrements the packet drop probability if the link is idle. BLUE may thus lose the *early* warning advantage of RED. Figure 1 shows the actual queue length and the packet drop probability of BLUE in a simple Web configuration (figure 7(a)). The BLUE drop probability does not increase until the queue becomes full, and does not decrease until the queue becomes idle.

Queue management has also been studied in the context of Kelly's utility framework.[6–11, 17–19] Lapsley et al.[6–8, 17] argue that RED parameters can be tuned to either provide high utilization or low queuing delay but not both. They propose Random Early Marking (REM) to achieve high utilization with negligible delays and loss, regardless of the number of sources. This is accomplished through decoupling congestion measures (price) and performance measures such as loss or delay. The congestion measure is computed by each link and fed back to the sources through packet dropping or marking. Thus, sources reach a globally optimal equilibrium. Kunniyur et al.[9, 10] consider a penalty function formulation instead of the dual formulation in REM, for designing congestion controllers. A decentralized design of ECN marking rates at each node is used to ensure global loss-free operation of a fluid model of the network. This class of algorithms, which also includes the recent PI controller,[11] is mainly designed for marking (ECN) and has not been adequately studied without ECN.

Finally, Firoiu and Borden[20] show how RED can induce network instability if not properly configured. RED performance is analytically studied in,[21] and selection of RED parameters and their effect on web traffic is studied in.[13]

3. QUEUE-BASED VERSUS LOAD-BASED ACTIVE QUEUE MANAGEMENT (AQM)

Is queue length (as used in RED) a sufficient congestion indicator? Is the load information sufficient? What about queuing delay versus queue length? This topic has been recently debated in a number of forums, such as the IRTF end-to-end group. We classify active queue management (AQM) algorithms into two classes based on the *primary* metric used for detecting congestion. The first class, queue-based AQM, uses only queue length as an indicator. RED and FRED[2–4] belong to this class. The second class, load-based AQM, uses load information for congestion detection. REM and AVQ[6–10, 17] belong to the second class.

3.1. Queue-based AQM

RED,[2] FRED[3] and SRED[4] belong to the queue-based AQM class for the following reason. RED uses a long-term average of the queue length to absorb temporary burstiness. This is the sole congestion indicator in RED, which assumes queue length indirectly reflects network load and queue drain. Figure 2 illustrates one situation where this may be problematic. The figure shows the actual queue length and packet drop probability of RED

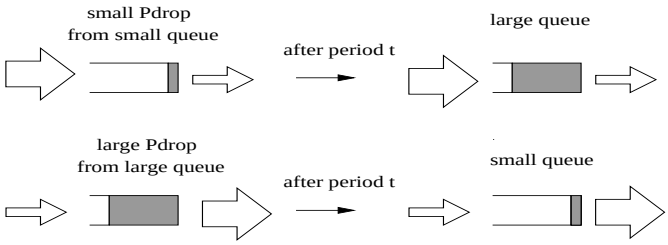
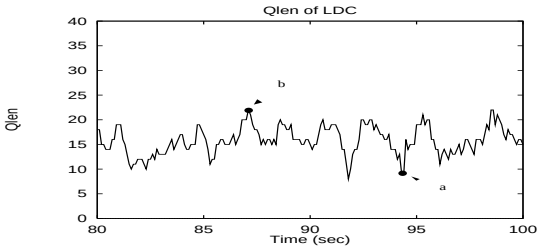
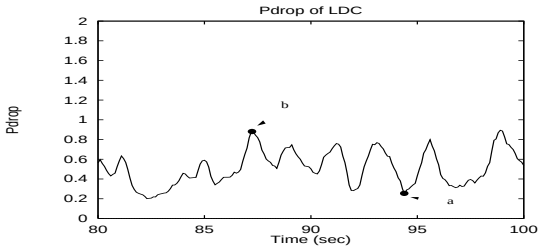


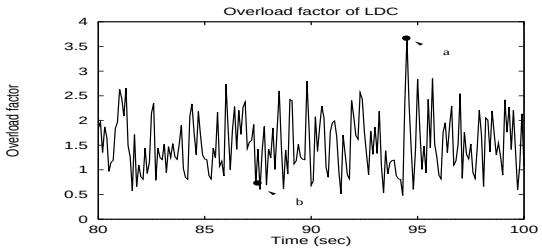
Figure 3. Queue dynamics with queue length as sole congestion indicator



(a) Queue length



(b) Drop probability based on queue length only



(c) Load factor

Figure 4. Queue length as the sole congestion indicator

with Web traffic. (We use the simple Web configuration illustrated in figure 7(a)). Between time 60 and 70 seconds, the actual queue length approaches zero, but the packet drop probability stays high. The fact that the queue may be large but draining rapidly, or small but building up rapidly is not reflected in the drop probability.

Figure 3 explains this situation. In the upper half of the figure, although the queue arrival rate is greater than the drain rate (indicated by the size of the arrow), the router uses a small packet drop probability due to the short queue length (since traffic is bursty). The lower part of the figure depicts the opposite scenario. A queue is drained rapidly since the queue service (drain) rate is higher than the arrival rate. Nevertheless, P_{drop} is high due to an initially large queue length, resulting in excessive drops and subsequent underutilization. A similar argument can be made when the drain rate is variable (small queue but small drain rate, large queue but high drain rate). The load insensitivity problem can be observed in figure 4. We run the simple Web configuration (described in section 5.1) for 100 seconds. The figure shows the actual queue length, drop probability if based on queue length *only*, and load information from 80 to 100 seconds. The point labeled "a" illustrates the upper case of figure 3. There, the load is high but P_{drop} is low since Q_{avg} is low. Q_{avg} grows afterwards, causing P_{drop} to increase. At point "b" between 85 and 90 seconds, P_{drop} is high due to high Q_{avg}. However, the load at that point is not high. P_{drop} need not be as high since the queue will be depleted shortly. Thus load information is a good congestion indicator.

3.2. Load-based AQM

Load information is used in REM, AVQ and the PI controller.[6–11, 17] The load information speeds up the response, but must be carefully used in order to avoid instability. The advantages of using the load information are studied in.[11, 18, 19] The load information in such schemes is measured in various forms, such as the queue length slope, or the difference between input rate and output rate. A simple load indicator that we will use is the load factor defined as the ratio of periodically measured arrival rate (input) and service rate (output):

$$Load\_Factor = \frac{Input\_Rate}{Output\_Rate}. \tag{1}$$

Output_Rate is the link capacity for this particular queue (i.e., not being used by higher priority queues) in a quality of service (QoS) network. Thus, *Output_Rate* is the remaining capacity after subtracting higher priority traffic from output link capacity. When only one class exists (e.g., in a best effort network), *Output_Rate* is the same as output link capacity. A high load factor value indicates congestion and a low value indicates link underutilization. Both cases are undesirable. The router should maintain the load factor less than (but close to) 1 in steady state.

Is queue length information still important even when the load factor is used? Clearly, queue length is a function of load. However, queue length gives a more stable (slower) congestion indication. In addition, queue length information may be helpful when making dropping decisions, in order to allow drainage of queue backlogs. This ensures that queuing delay is low and remaining buffer capacity can be used to absorb packet bursts.

What about using queue length versus queuing delay? The queuing delay has two benefits over the queue length. First, a target queuing delay is a meaningful measure for users (unlike target queue length, which depends on buffer sizes and link speeds). Second, queuing delay is independent of output link rate, in contrast to queue length. A target queue size of 100 packets produces entirely different results when the drain rate is 1.5 Mbps from when it is 10 Mbps. The user-specified target queuing delay is converted to a target queue length for a certain drain rate as follows:

$$Q_{target} = D_{target} \times Output\_Rate. \tag{2}$$

where Q_{target} is target queue length, D_{target} is target queuing delay, and *Output_Rate* is the output link rate.

4. LOAD/DELAY CONTROL (LDC)

In this section, we describe a generic class of active queue management algorithms: LDC (Load/Delay Controllers). We maintain the misbehaving (but not bursty) flow punishment and global synchronization avoidance benefits of RED, but the algorithm provides better load (input/output rate) response, and more intuitive parameters. LDC does not assume a particular (different from TCP) source behavior. Explicit congestion notification (ECN) can improve performance, but it is not necessary for LDC operation. Throughout the remainder of this paper, we use the term packet drop probability, though this can be replaced by packet mark probability if ECN-compliant end systems are used.

The goals of LDC algorithms are (1) to maintain queuing delay below a target value, and (2) to maintain queue arrival rate at or below queue service rate. Given the arrival rate, service rate, queue length and target queuing delay, we design an example LDC algorithm in this section.

4.1. Measuring the Load Factor and Queue Length

RED computes the average queue size with every packet arrival as an Exponential Weighted Moving Average (EWMA) with w_q (the weight for the current sample) of 0.002. The primary goal of the *average* queue size in RED is to avoid bias against bursty traffic. LDC calculates queuing delay at fixed time intervals, τ, not with the arrival of each packet. Thus computation of the queuing delay is synchronized to that of the load factor. Should *average* input/drain rates or *instantaneous* rates be used in load factor computation? The same question also applies to queue sizes. In our experiments, we use 0.002 as the weight for the *current* queue size sample, and 0.002 as the weight for *previous* input/drain rates for the load factor. Averaging for both is necessary to

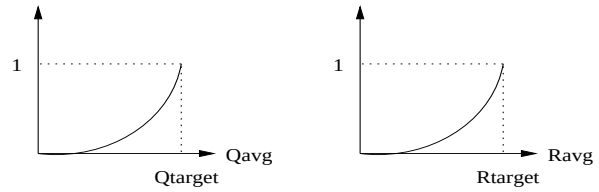


Figure 5. P_{drop} versus average queue length and load factor

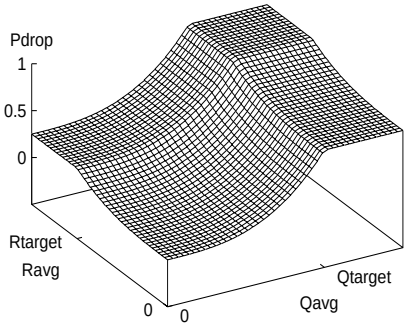


Figure 6. Packet drop probability of LDC

avoid sensitivity to the time interval τ. The queue size average places significant weight on queue size *history* to avoid bias against bursty traffic. The input and drain rate averages place more emphasis on the current sample to increase *responsiveness* of the algorithm to current traffic load.

4.2. The Packet Drop Probability Function

LDC employs concave probability functions as shown in figure 5. The concave function eliminates the need for a minimum threshold and avoids *discontinuities* (e.g., as seen in non-gentle versions of RED). The potential drawback of the concave function is higher computational complexity. This problem can be avoided by using small exponent values (less than 4) to limit multiplication operations. LDC computes the ratio of the load factor and its desired value, and the ratio of the queue length and its target value. A ratio of 1 means that the congestion indicator is at its target value. Using a ratio rather than a difference simplifies the design of the concave P_{drop} function (figure 5). The two ratios are combined additively to calculate the packet drop probability. Addition clearly divides the contribution of the two congestion indicators and simplifies configuring LDC to give more emphasis to one indicator over the others. One example of an LDC function is:

$$P_{drop} = \frac{3}{4} min(1, (\frac{Q_{avg}}{Q_{target}})^n) + \frac{1}{4} min(1, (\frac{R_{avg}}{R_{target}})^n). \quad (3)$$

The contribution of each of the load factor and the average queue length is upper-bound by 1. Multiplicative factors of $\frac{3}{4}$ and $\frac{1}{4}$ are used with the queuing delay and load respectively to normalize the sum, guaranteeing that P_{drop} is upper-bound by 1. Other factors (instead of $\frac{3}{4}$ and $\frac{1}{4}$) can be used, as long as they add up to one. This example packet drop probability function is illustrated in figure 6.

5. PERFORMANCE EVALUATION

In this section, we compare the performance of LDC, RED, SRED, FRED, BLUE, and REM. For RED and REM, we use the *ns* (ns-2.1b7[22]) implementations provided by the authors of the algorithm, while we implement SRED, FRED and BLUE (in addition to LDC) according to the pseudo-code given in the papers where they first appeared. We have validated our results by replicating the experiments given in the original papers.

Table 1. Performance metrics

Metric	*Description*
Response Time	The (mean and maximum) time elapsed (in seconds) between the time at which the request of a web client is triggered and the time at which the last requested page from a server arrives at this client.
Delay	The (mean and maximum) time elapsed (in seconds) between when a sender sends a packet and when a receiver receives that packet.
Throughput	The total amount of data received *at the receiver application layer* during the entire simulation time, divided by the entire simulation time. We give goodput values in Mbps.
Packet Drop Ratio	The ratio of dropped packets to the total number of packets sent during the simulation time.

Table 2. Simulation parameters

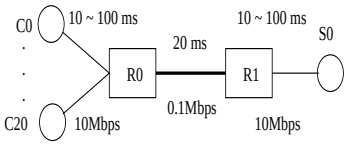
Algorithm	*Parameter Settings*
RED	$th_{min} = \frac{1}{12}\times$ buffer, $th_{max} = \frac{1}{4}\times$ buffer, $(3 \times th_{min})$, $w_q = 0.002$, $max_p = 0.1$ (as recommended by RED designers[23]), gentle version used
SRED	Full SRED is used, $max_p = 0.15$, $M = 1000$, $p = 0.25$
FRED	$min_q = 4$, others same as RED
BLUE	$decrement = 0.02$, $increment = 0.2$, freeze_time = 0.1 s
REM	$\alpha = 0.1$, $\gamma = 0.005$, $\phi = 1.001$, $w_q = 0.002$, time interval $\tau = 0.002$ s
LDC	$D_{target} = 4.0$ s, ($w_q = 0.002$, $w_r = 0.998$, $n = 3$, $R_{target} = 0.95$, time interval $\tau = 0.002$ s, weights of $\frac{3}{4}$ and $\frac{1}{4}$ for the delay and load components)

5.1. Simulation Model

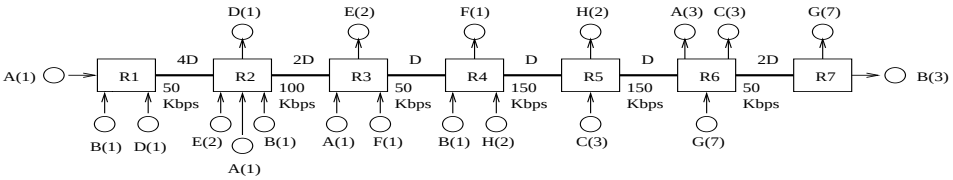
Today's Internet has two important characteristics. First, web traffic dominates the Internet (more than 80%).[24] Web traffic is short-lived and bursty, not long-lived and steady like large FTP transfers that are used in most active queue management simulations. Second, streaming and real-time audio/video applications are slowly emerging in the Internet. Some of these applications do not use TCP or TCP-friendly mechanisms like,[25] thus maintaining a fixed rate and not reducing their rates on packet loss.

We use 3 simulation models. The first model, shown in figure 7(a), only uses one web server and 10 clients, for simplicity. All connections share one bottleneck link, with a link capacity of 0.1 Mbps. 20 HTTP-TCP connections are used, in addition to one CBR-UDP connection sending at 50 kbps, which is half of the bottleneck link capacity. The web traffic is generated by a Poisson Process where the inter-object, inter-page and inter-session times are defined by an exponential distribution. We fix the random number generator seed for each run to obtain comparable results when we compare different algorithms. We refer to this dumbbell configuration as "The simple web configuration." The total simulation time for the simple web configuration is 900 seconds.

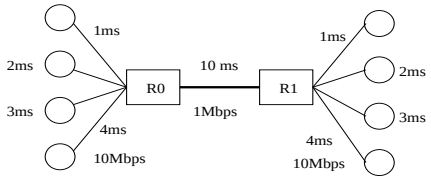
The purpose of our second model is to compare LDC to algorithms proposed in the literature in a more realistic setup. We use the Generic Fairness Configuration 2 (GFC-2) shown in figure 7(b). The configuration has multiple bottleneck links and connections with different link capacities and different round trip times. We use D = 5 ms. 22 HTTP sessions created as discussed above, in addition to one 12 kbps CBR-UDP flow, and one unlimited FTP (on TCP-Reno) flow. Servers reside at the receivers while clients are at the senders. CBR-UDP runs on node F and unlimited FTP on node H. The HTTP sessions are created on each sender/receiver pair. The CBR-UDP flow shares the bottleneck link between R3 and R4 while the FTP flow shares the link from R4 to R5 with several HTTP sessions. Total running time is 900 seconds. We run 5 simulations with different random number generator seeds, and average the results.



(a) The simple Web configuration



(b) Generic Fairness Configuration-2



(c) The simple FTP configuration

Figure 7. Three Simulation Configurations

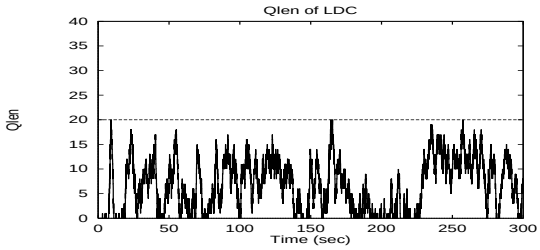


Figure 8. Target delay guarantee of LDC algorithms

The third model is a simple configuration with large FTP transfers, and its purpose is to simplify examining and comparing LDC parameter values. Figure 7(c) shows 4 sender-receiver pairs with one bottleneck link. 100 FTP connections are running on TCP Reno– 25 on each sending node. D_{target} is set to 0.2 seconds for this experiment only (outgoing link capacity is 1 Mbps). To evaluate each algorithm, we use the performance metrics described in table 1. The parameters for each algorithm used in the simulations are summarized in table 2. We use a TCP timer granularity of 100 ms and segment size of 1000 bytes. The buffer sizes used are 120 kB.

5.2. Performance Results

In this section, we compare various algorithms by simulations on the configurations discussed above.

5.2.1. LDC Delay Control

Figure 8 shows the delay guarantee property of LDC. We simulate the simple web configuration for 300 seconds with D_{target} of 1.6 seconds. Since the output rate is fixed (0.1 Mbps), the target delay can be converted to a target queue length of 20 kB ($\frac{1.6s \times 0.1Mbps}{8 \times 1k} = 20kB$). The figure shows that, throughout the entire simulation period, the queuing delay never exceeds the target delay.

5.2.2. Multiple-bottleneck Configuration

The results for the GFC-2 configuration are shown in tables 3 and 4. Note that throughput is indicated in Mbps. For web applications, the crucial factor is user response time.[13] From the table, FRED exhibits the shortest response time, and also achieves high throughput. This is because FRED is designed to control misbehaving (CBR-UDP) flows and protect TCP flows, using per-flow accounting. REM achieves low response time but loses some throughput. LDC also exhibits good response time at the expense of some throughput degradation. RED

Table 3. Comparison of queue management algorithms on GFC-2: Mean

Algorithm	HTTP Response Time		UDP Delay		Throughput			Packet Drop Ratio			
	Mean	Max	Mean	Max	HTTP	UDP	FTP	HTTP	UDP	FTP	Total
DropTail	79.070	1094.46	0.950	6.983	0.110	0.012	0.134	3.012	0.956	0.000	1.317
RED	69.358	1138.04	0.465	7.036	0.098	0.012	0.134	6.356	1.808	0.196	2.868
SRED	74.183	1054.85	1.308	7.941	0.100	0.012	0.129	4.005	0.589	0.031	1.752
FRED	52.856	931.835	0.301	4.595	0.132	0.011	0.127	6.378	11.431	1.057	4.282
BLUE	79.029	1177.68	1.409	8.261	0.096	0.012	0.132	3.287	1.623	0.000	1.474
REM	62.342	1008.96	0.406	5.246	0.093	0.012	0.125	7.372	2.437	0.183	3.280
LDC	58.851	1126.74	0.227	2.225	0.088	0.012	0.129	9.444	2.133	0.588	4.273

Table 4. Comparison of queue management algorithms on GFC-2: Standard deviation

Algorithm	HTTP Response Time		UDP Delay		Throughput			Packet Drop Ratio			
	Mean	Max	Mean	Max	HTTP	UDP	FTP	HTTP	UDP	FTP	Total
DropTail	26.729	286.220	0.668	3.010	0.017	0.000	0.001	1.783	1.003	0.000	0.745
RED	24.238	335.954	0.100	1.778	0.014	0.000	0.002	1.316	0.941	0.024	0.574
SRED	24.179	366.155	1.016	2.285	0.009	0.000	0.009	0.946	0.384	0.014	0.399
FRED	21.349	270.126	0.099	0.855	0.036	0.001	0.003	2.599	4.434	0.363	0.897
BLUE	26.950	286.664	0.598	2.167	0.012	0.000	0.002	1.899	1.236	0.000	0.841
REM	23.722	276.992	0.160	0.629	0.014	0.000	0.012	1.947	1.192	0.080	0.817
LDC	19.899	331.829	0.047	0.174	0.006	0.000	0.005	1.037	0.750	0.105	0.506

performs reasonably well. All other algorithms span similar ranges, with somewhat high response time, but high throughput. BLUE and SRED perform well in terms of packet drop ratio. Standard deviation is highest for drop tail, but all the algorithms are comparable.

5.3. Robustness

Although LDC has one main parameter, the target queuing delay, its other parameters, such as averaging weights, can also be modified. Small values of w_q such as 0.002 to 0.2, and large values of w_r such as 0.8 to 0.998 are recommended as the weights for the current sample in Q_{avg} and R_{avg} respectively (as discussed in section 4). We use $n = 3$ in the P_{drop} function. We observe that throughput and delay do not vary with different n values and various time interval τ values (results not shown in the paper). If n is large, however, implementation complexity may be high. The contributions of the queuing delay and load components to the P_{drop} function can also be varied. Table 5 shows results for the FTP configuration when we use (1,0), (0,1), $(\frac{1}{2},\frac{1}{2})$, or $(\frac{3}{4},\frac{1}{4})$ as weights for the queuing delay and load components of LDC, respectivaly. Queuing delay incorporates the drain rate, so queuing delay indirectly incorporates load information. From the table, $(\frac{3}{4},\frac{1}{4})$ exhibits the lowest packet delay and reasonably high throughput.

The time interval τ used is 2 ms, though complexity can be reduced by using a larger interval. Values of 0.9 or 0.95 are recommended for R_{target}, but we have obtained good results with higher values as well, since we always start dropping earlier than the target. The user primarily needs to set the parameter, D_{target}, the target queuing delay. If input traffic is not bursty, a small D_{target} gives good delay without degrading throughput considerably. However, web traffic is bursty and a very low D_{target} is not recommended since it causes timeouts and TCP retransmissions.

Table 5. Load factor versus queuing delay (FTP traffic)

LDC(Q,ρ)	Delay	Throughput	Drop Ratio
LDC ($\frac{3}{4}, \frac{1}{4}$)	0.1335	0.9796	25.01
LDC ($\frac{1}{2}, \frac{1}{2}$)	0.1470	0.9765	23.12
LDC $(1, 0)$	0.1470	0.9821	25.55
LDC $(0, 1)$	0.7086	0.9976	3.88

5.4. Merits and Drawbacks of Various Approaches

Table 6 summarizes the advantages and disadvantages of the active queue management algorithms we have investigated. From the table, it is clear that complexity of the algorithms depends on quantities they measure, their sampling frequency, the algorithm employed to determine drop, and the state they maintain. An important question is whether active queue management can achieve low delay and high throughput simultaneously. A non-zero queue size ensures high utilization/throughput with bursty traffic. But a non-zero queue size leads to queuing delay, increasing end-to-end delay. The class of algorithms based on Kelly's framework, such as REM, combines active queue management with ECN and end system behavior to keep user utilities high. LDC algorithms allow control over the desired operating point through the target queuing delay parameter value.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we have compared a number of active queue management algorithms with a class of generic load and queue based algorithms. Queue-based algorithms like RED, and especially FRED, performed reasonably well but were difficult to configure. Load based algorithms like REM reduced queuing delay, but exhibited low throughput for Web traffic when used without ECN. Using both load and queuing delay as congestion indicators in LDC reduces response time and delay, but sometimes adversely affects throughput, in both single and multiple bottleneck cases, and with web and FTP traffic and UDP background. Moreover, LDC keeps actual queuing delay under the target queuing delay. LDC gives the user the freedom to control the queuing delay versus goodput tradeoff.

A number of issues remain unresolved. We need to study the effect of ECN in the context of LDC. ECN allows active queue management at the router and TCP at the sender to cooperate, reducing queuing delay and packet loss. This will allow us to include schemes such as AVQ and the PI controller in our comparisons. Parameter values of LDC algorithms need further study as well. Finally, we need to study the effect of variable output rate in the context of multiple queues. This study may be extended to multiple service classes such as a differentiated services network with TCP-friendly traffic conditioners, as in.[26]

Acknowledgements

We would like to thank Srinivas R. Avasarala (former student of the second author (Sonia Fahmy); now with Andiamo Systems) for providing us with his FRED and SRED ns-2 implementations; and Sanjeewa Athuraliya and Victor Li (with the California Institute of Technology) for providing us with their REM ns-2 implementation.

REFERENCES

1. B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang, "Recommendations on queue management and congestion avoidance in the internet." RFC 2309, April 1998. http://info.internet.isi.edu:80/in-notes/rfc/files/rfc2309.txt.
2. S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking* **1**, pp. 397–413, August 1993. ftp://ftp.ee.lbl.gov/papers/early.ps.gz.
3. D. Lin and R. Morris, "Dynamics of random early dectection," in *Proceedings of the ACM SIGCOMM*, **27**, pp. 127–136, September 1997.

Table 6. Advantages and disadvantages of active queue management algorithms

Algorithm	*Advantages*	*Disadvantages*	*Complexity*
RED	–Early congestion detection –Bias against bursty traffic mitigated –Global synchronization mitigated	–Difficulty in parameter setting –Insensitivity to traffic load and drain rate	–High Q sampling frequency –Computational Complexity: $+$, $-$, $\times$, $\div$, comparison, RNG (random number generator)
SRED	–Stabilized queue occupancy –Protection from misbehaving flows	–Some per-flow state (zombie list) –RED disadvantages	–High Q sampling frequency –O(M) space complexity –Computational Complexity: $+$, $\times$, $\div$, comparison, RNG, hashing
FRED	–Good protection from misbehaving flows	–Per-flow state –RED disadvantages	–High Q sampling frequency –O(number of active flows) space complexity –Computational Complexity: $+$, $-$, $\times$, $\div$, comparison, RNG
BLUE	–Easy to understand –High throughput	–No early congestion detection (P_{drop} updated only with queue overflow or link idle) –Slow response and dependency on history	–High Q sampling frequency –Computational Complexity: $+$, $-$, comparison, RNG, timer
REM	–Low delay and small queues –Independence of the number of users	–Some complexity due to parameters –Low throughput for Web traffic –Inconsistency with TCP sender mechanism; works best with ECN	–Low Q sampling frequency but measures load –Computational Complexity: $+$, $-$, $\times$, $\div$, exponent, RNG, timer
LDC	–Sensitivity to traffic load and drain rate –Low delay and some losses of throuput –Target delay achieved –Intuitive parameters, meaningful to user (target delay)	–Some complexity due to parameters	–Low Q sampling frequency but measures load –Computational Complexity: $+$, $-$, $\times$, $\div$, exponent, RNG, timer

4. T. J. Ott, T. V. Lakshman, and L. H. Wong, "SRED: Stabilized RED," in *Proceedings of the IEEE INFOCOM*, March 1999.
5. W. Feng, D. Kandlur, D. Saha, and K. Shin, "BLUE: A new class of active queue management algorithms." U. Michigan CSE-TR-387-99, April 1999. http://www.eecs.umich.edu/~wuchang/blue/. Also appears in NOSSDAV 2001.
6. D. E. Lapsley and S. H. Low, "Random Early Marking for Internet Congestion Control," in *Proceedings of the IEEE GLOBECOM*, December 1999.
7. S. H. Low and D. E. Lapsley, "Optimization flow control, I: basic algorithm and convergence," *IEEE/ACM Transaction on Networking* , December 1999. http://www.ee.mu.oz.au/staff/slow/research/.
8. S. Athuraliya, V. H. Li, S. H. Low, and Q. Yin, "REM: Active Queue Management," *IEEE Network* , May/June 2001. http://netlab.caltech.edu/netlab-pub/.
9. S. Kunniyur and R. Srikant, "End-To-End Congestion Control: Utility Functions, Random Losses and ECN Marks," in *Proceedings of the IEEE INFOCOM'2000*, March 2000.
10. S. Kunniyur and R. Srikant, "A Time-Scale Decomposition Approach to Decentralized ECN Marking," in *Proceedings of the IEEE INFOCOM'2001*, April 2001.
11. C. V. Hollot, V. Misra, D. Towsley, and W.-B. Gong, "On Designing Improved Controllers for AQM Routers Supporting TCP Flows," in *Proceedings of the IEEE INFOCOM'2001*, April 2001. http://www.ieee-infocom.org/2001/.
12. K. Ramakrishnan and S. Floyd, "A proposal to add explicit congestion notification (ECN) to IP." RFC 2481, January 1999.
13. M. Christiansen, K. Jeffay, D. Ott, and F. D. Smith, "Tuning RED for web traffic," in *Proceedings of the ACM SIGCOMM*, August 2000.
14. S. Kalyanaraman, R. Jain, S. Fahmy, R. Goyal, and B. Vandalore, "The ERICA Switch Algorithm for ABR Traffic Management in ATM Networks," *IEEE/ACM Transactions on Networking* **8**, pp. 87–98, February 2000. http://www.cis.ohio-state.edu/~jain/papers/erica.htm.
15. B. Vandalore, R. Jain, R. Goyal, and S. Fahmy, "Dynamic queue control functions for ATM ABR switch schemes: Design and analysis," *Journal of Computer Networks* **31**, pp. 1935–1949, August 1999. http://www.cis.ohio-state.edu/~jain/papers/cnis_qctrl.htm.
16. "ECN reachability tests." http://www.aciri.org/tbit/ecn.html, September 2000.
17. S. Athuraliya, D. E. Lapsley, and S. H. Low, "An Enhanced Random Early Marking Algorithm for Internet Flow Control," in *Proceedings of the IEEE INFOCOM*, March 2000.
18. C. V. Hollot, V. Misra, D. Towsley, and W.-B. Gong, "A Control Theoretic Analysis of RED," in *Proceedings of the IEEE INFOCOM'2001*, April 2001. http://www.ieee-infocom.org/2001/.
19. F. Kelly, P. Key, and S. Zachary, "Distributed admission control," *IEEE Journal on Selected Areas in Communications* **18**, 2000.
20. V. Firoiu and M. Borden, "A study of active queue management for congestion control," in *Proceedings of the IEEE INFOCOM*, March 2000. http://www.ieee-infocom.org/2000/papers/405.pdf.
21. T. Bonald, M. May, and J.-C. Bolot, "Analytic evaluation of RED performance," in *Proceedings of the IEEE INFOCOM*, March 2000. http://www.ieee-infocom.org/2000/papers/369.ps.
22. UCB/LBNL/VINT groups, "UCB/LBNL/VINT Network Simulator." http://www.isi.edu/nsnam/ns/, May 2001.
23. S. Floyd, "RED web page." http://www.aciri.org/floyd/red.html, August 2001.
24. CAIDA, "The CAIDA Web Site." http://www.caida.org/.
25. S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-based congestion control for unicast applications," in *Proceedings of the ACM SIGCOMM*, August 2000. http://www.aciri.org/tfrc/.
26. A. Habib, S. Fahmy, and B. Bhargava, "Design and evaluation of an adaptive traffic conditioner for differentiated services networks," in *Proceedings of IEEE ICCCN*, pp. 90–95, October 2001.