

A New Rate-based Active Queue Management: Adaptive Virtual Queue RED

Do J. Byun and John S. Baras, *Fellow, IEEE*

*Department of Computer Science, University of Maryland at College Park
dbyun@hns.com, baras@isr.umd.edu*

Abstract

In an effort to improve performance of congested gateways, a new Active Queue Management (AQM) algorithm, Adaptive Virtual Queue Random Early Detection (AVQRED), was developed by feeding virtual queue size to the RED algorithm. The objective of the new algorithm is to improve overall performance by keeping link utilization high, link utilization stable, queueing delay low and consecutive packet drop rate low. This paper shows the objective is met by comparing AVQRED with six other well known AQM methods in a realistic emulation environment. To provide fair comparisons, the AQM parameters are fine-tuned by exploring many different parameter settings via emulations. The emulation results conclude that AVQRED improves overall performance by 8 to 25%. To provide intuitive properties and validate the emulation results, a mathematical model is proposed and fed to MATLAB, and the results from MATLAB and emulations are compared.

1. Introduction

Due to exponential increases in internet traffic, congestion has been an unavoidable attribute to many Internet service providers for the last decade and improving performance of congested gateways has been an active research interest. This paper contributes an effort to improve overall performance of congested gateways by proposing a new rate-based AQM algorithm, Active Virtual Queue Random Early Detection (AVQRED).

Active Queue Management (AQM) is an algorithm that detects and reacts to congestion to avoid queue overflows. There are generally two ways to react to congestion: signal congestion to traffic sources explicitly by setting Explicit Congestion Notification bits, or signal congestion to traffic sources implicitly by dropping packets. A new AQM algorithm was developed by overlaying a virtual queue on top of the real queue and applying the RED algorithm to the virtual queue instead of the real queue. The objective of the new algorithm is to improve overall performance by keeping link utilization high, link

utilization stable, queueing delay low and consecutive packet drop (mark) rate low.

This paper compares AVQRED with six well-known AQM methods, RED [4], ARED [7], BLUE [8], GKVQ [20], AVQ [18], and PI [17] in a realistic emulation environment. To provide fair comparisons, the parameters for all seven AQM methods are fine-tuned before the comparisons via emulations. The emulation results conclude that AVQRED improves overall performance by 8 to 25%. To provide intuitive properties and validate the emulation results, a mathematical model is proposed and fed to MATLAB, and results from MATLAB and emulations are compared.

This paper is organized as follows: Section 2 provides an overview of the six AQM methods. Section 3 discusses the new AQM algorithm, AVQRED. Section 4 provides the emulation framework and methodologies. Section 5 provides the emulation results. Section 6 provides a mathematical model. Section 7 validates the emulation results by feeding the model to MATLAB and comparing the MATLAB results with the emulation results. Finally, section 8 concludes the paper.

2. Overview of other AQM methods

This section provides high-level understanding of each AQM method which is compared with AVQRED. Refer to the corresponding reference for more details about the algorithm.

2.1. RED

The RED [4] algorithm computes the marking probability when the weighted queue size falls between min_{th} and max_{th} parameters. The marking probability becomes higher as the weighted queue size gets closer to max_{th} (becomes 1 if it's greater than max_{th}), and it also becomes higher as the distance between each marking gets bigger. Parameter tuning is required for w_q and max_p . w_q controls the weighted average queue size which then determines how quickly the algorithm reacts to congestion. Reacting too quickly may result in oscillations in queue size, and reacting too slowly may result in queue overflows. max_p is a scaling factor for the

marking probability which also controls how quickly the algorithm reacts to congestion.

2.2. ARED

The intuition behind the ARED [7] algorithm is to self-adjust max_p according to the target average queue size every half second. Emulation results revealed that the adjustment of max_p does not find the value that best represents the desired packet loss rate of the gateway. i.e. max_p reduction is so aggressive that it always stays at a lower value than the desired packet loss rate resulting in the average queue size to exceed max_{th} . Therefore, the ARED algorithm was slightly modified to have a hysteresis requirement where decreasing max_p by β requires the queue size to stay under the target queue size for n times.

2.3. BLUE

Unlike RED and ARED, BLUE [8] relies on packet loss and idle events of the queue when adjusting its marking probability. It increases or decreases by a configured value, $d1$ or $d2$ respectively, when the queue size exceeds a threshold or it becomes empty. It does so no more than once every configured interval, $freeze_time$.

2.4. GKVQ and AVQ

Gibbens-Kelly Virtual Queue (GKVQ) [20] maintains a virtual queue whose service rate is the desired link utilization. When an incoming packet exceeds the virtual queue limit, it drops or marks the packet. Adaptive Virtual Queue (AVQ) [18] maintains the same virtual queue whose capacity is dynamically adjusted. The virtual capacity is adjusted by adding the number of bytes that could have been serviced between the last and the current packets minus the bytes that were just received. The problems with these two schemes are global synchronization and queue overflows when virtual capacity exceeds the actual processing capacity. Although AVQ seems to solve the latter by adjusting the virtual capacity, there is an assumption that the actual processing capacity is static which could lead to high queuing delays and queue overflows if the processing allocation shifts – a common phenomenon in real networks. However, as long as the virtual capacity does not exceed the actual processing capacity, GKVQ and AVQ yield low queuing delays and stable link utilizations.

2.5. PI

The main idea behind the Proportional Integrator (PI) [17] algorithm is that the queue length slope determines the packet marking probability, and the queue is regulated by the desired queue length. i.e. $p[k+1] = p[k] + a(q[k+1] - qref) - b(q[k] - qref)$. The algorithm computes a new p every T seconds.

3. AVQRED algorithm

The motivations behind AVQRED are to fix the problems of AVQ: global synchronization (consecutive packet drops) and inaccurate virtual capacity which leads to high queuing delays.

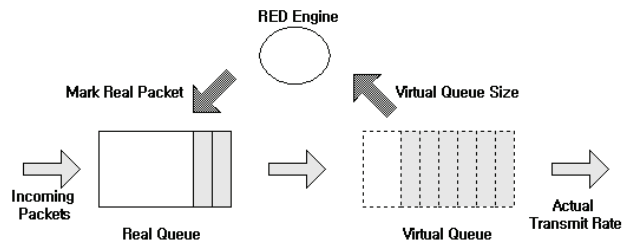


Figure 1. AVQRED flow diagram

The AVQRED algorithm constructs a virtual queue and feeds the virtual queue sizes to the RED algorithm instead of feeding the weighted average queue sizes to it. AVQRED reshapes the incoming traffic according to the desired link utilization because the RED algorithm reacts to the congestion level of the virtual queue which is serviced by the desired link utilization. The *AVQRED Algorithm* highlights the AVQRED parameters in bold. Note that w_q and max_p are no longer in the algorithm because their functionalities are replaced by the desired link utilization in AVQRED. $alpha$ is a low-pass filter for the actual capacity calculation and choosing a small number such as 0.05 is recommended since changes in processing capacity are not drastic. $min_capacity$ and $max_capacity$ define the range of processing capacity. Choosing a big value for $max_capacity$ is recommended if the maximum processing capacity of the gateway is not known and one wishes to maximize the capacity. Choosing a value too low for $min_capacity$ will cause a delay in finding the actual capacity at the beginning but it will stay within the right range once it is found. Note that min_{th} and max_{th} no longer control the real queue length directly. They define the RED region of the virtual queue which indirectly controls the real queue length. One big advantage of applying RED to the virtual queue is that the algorithm works well even with a small queue (because

the RED region for the virtual queue can still be big) whereas having a small queue makes RED resemble a tail-drop queue.

```

Initialization:
count <- -1
last_measure <- curr_time /* Current time in milli-seconds */
prev_tx_bytes <- bytes_transmitted

for each packet arrival
/* Calculate virtual queue size */
delta_time <- curr_time - last_measure
if delta_time > 1
/* Compute actual output rate in bps */
tx_bytes <- bytes_transmitted
output_rate <- (tx_bytes - prev_tx_bytes) * 8000 / delta_time
prev_tx_bytes <- tx_bytes

/* Smoothen virtual capacity */
v_capacity <- alpha * output_rate + (1.0 - alpha) * v_capacity

/* Update virtual capacity */
v_capacity <- MAX(MIN(max_capacity, v_capacity),
min_capacity)

/* Compute the number of bytes that could actually be
* processed and transmitted
*/
serviced_bytes <- v_capacity / 1000 / 8 * delta_time

if VQ > serviced_bytes
VQ <- VQ - serviced_bytes
else
VQ <- 0
q_time <- curr_time

last_measure <- curr_time
q_size <- VQ / 1500

/* Feed virtual queue size to the RED algorithm */
if min_th < q_size < max_th
count <- count + 1
p_b <- (q_size - min_th) / (max_th - min_th)
p_a <- p_b / (1 - count * p_b)
With probability p_a:
Mark the arriving packet
count <- 0
else if max_th <= q_size
Mark the arriving packet
count <- 0
else
count <- -1
VQ <- VQ + b /* b = number of bytes in current packet */

```

AVQRED Algorithm

4. Emulation framework

The actual gateway code from *Hughes Network Systems* was used to evaluate the AQM methods. The software was simplified to have only one queue for receiving packets and one thread that dequeues and

transmits the packets. The seven AQM methods were implemented at the receive queue and “drop” was chosen for the marking method. AQM was implemented only on one queue which is assumed to be the only congestion queue in our network scenarios. To eliminate the unnecessary system complexity, satellite specific features such as TCP spoofing, per-user queuing and output rate limiting were removed. Although the software serves as a gateway in satellite networks, it was modified to be a generic IP Gateway. The emulation environment was constructed using two IP Gateways (one serves as a router to the internet and the other serves as modems for N different users) and a traffic generator called *Spirent*®.

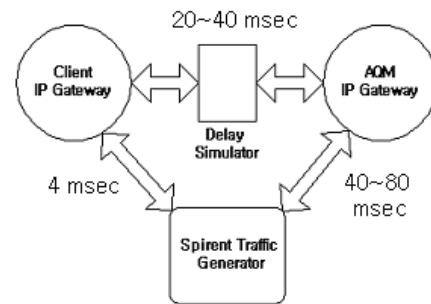


Figure 2. Emulation flow diagram

Figure 2 illustrates the connectivity of the emulation setup. To best emulate latencies, a delay simulator was inserted between the two gateways. The round trip time (RTT) between the client IP Gateway and the *Spirent*® is 4 msec, the RTT between the client IP Gateway and the AQM IP Gateway is 20~40 msec, and the RTT between the AQM IP Gateway and the *Spirent*® is 40~80 msec resulting in an end to end RTT of 104~204 msec. 4000 HTTP connections were generated between 200 clients and 60 servers with the following attributes.

1. At the startup, there are 200 new HTTP connections every 5 seconds with 5 second sleep time between each ramp up until 4000 HTTP connections are established.
2. When a connection is closed, a new connection is created to fill the gap to maintain 4000 HTTP connections.
3. Each web page contains 250Kbytes ~ 550Kbytes of data.
4. The maximum download speed of each TCP connection is 5Mbps.
5. The average birth and death rate of the connections is about 200 connections per second (approximately 5% of the total population).

4.1. Evaluation methodologies

The following performance metrics were used:

1. Link utilization – Mean and standard deviation of link utilizations are compared.
2. Queuing delay – Mean and standard deviation of queuing delays are compared.
3. Packet loss – Mean and standard deviation of packet losses are compared.

4.1. Parameter settings

The processing capacity ranges between 13 to 25¹ Mbps. Throughout the experiments, each AQM method has the same target queue size or queuing delay, 22² packets or 10 msec respectively. In other words, the queue size based AQM methods will use the target queue size of 22 packets, and the rate based AQM methods will use the target queuing delay of 10 msec. For each AQM method, results from three 15 minute emulation runs for 10 to 27 different parameter settings were used to fine-tune the parameters. Details on how the tuning was done are described in the next subsection, 4.1.1. The parameter permutations and the results are not shown due to the space limitation.

4.1.1. Parameter selection rule. Parameter settings are ranked for each performance metric. Based on the rank, it is given a weight, $x = 2^{\text{rank}}$. Each setting is then given a weight which is the sum of the weights (x 's) of all its metrics. The settings with the three lowest weights are chosen assuming each metric is equally important. Only two settings are chosen for ARED due to its adaptive adjustments for max_p .

4.1.2. Final parameter settings. Table 1 shows the parameter settings that produced the best emulation results which were used throughout the final emulations.

¹ This can be a hypothetical number that may or may not be achievable by the gateway due to dynamics of the gateway resource allocation.

² 22 is derived from the desired queuing delay of 10 msec with 25 Mbps maximum processing capacity and 1400 bytes average packet size.

Table 1. Final parameter settings

AQM	PARAMETERS			
RED 1	$w_q = 1.0$	$min_{th} = 15$	$max_{th} = 30$	$max_p = 0.7$
RED 2	$w_q = 1.0$	$min_{th} = 15$	$max_{th} = 30$	$max_p = 1.0$
RED 3	$w_q = 0.1$	$min_{th} = 15$	$max_{th} = 30$	$max_p = 0.5$
ARED 1	$w_q = 1.0$	$min_{th} = 15$	$max_{th} = 30$	ARED $max_p = 0.7$
ARED 2	$w_q = 0.1$	$min_{th} = 15$	$max_{th} = 30$	ARED $max_p = 0.7$
BLUE 1	$L = 30$	$d1 = 0.01$	$d2 = 0.05$	$freeze_time = 50$ msec
BLUE 2	$L = 30$	$d1 = 0.01$	$d2 = 0.05$	$freeze_time = 100$
BLUE 3	$L = 30$	$d1 = 0.001$	$d2 = 0.0005$	$freeze_time = 50$ msec
GKVO 1	Q Size = 16250 Bytes		Capacity = 13 Mbps	
GKVO 2	Q Size = 17500 Bytes		Capacity = 14 Mbps	
GKVO 3	Q Size = 18750 Bytes		Capacity = 15 Mbps	
AVQ 1	Q Size = 16250 Bytes		Capacity = 13 Mbps $\alpha = 0.1$	
AVQ 2	Q Size = 17500 Bytes		Capacity = 14 Mbps $\alpha = 0.05$	
AVQ 3	Q Size = 17500 Bytes		Capacity = 14 Mbps $\alpha = 0.1$	
PI 1	Interval = 10 msec	$a = 0.05$	$b = 0.01$	QRef = 22
PI 2	Interval = 10 msec	$a = 0.1$	$b = 0.02$	QRef = 22
PI 3	Interval = 10 msec	$a = 0.05$	$b = 0.02$	QRef = 22
AVORED1	Min=13 Mbps	Max=25 Mbps	$\alpha = 0.005$	$min_{th} = 10$ $max_{th} = 21$
AVORED2	Min=13 Mbps	Max=25 Mbps	$\alpha = 0.001$	$min_{th} = 10$ $max_{th} = 21$
AVORED3	Min=13 Mbps	Max=25 Mbps	$\alpha = 0.05$	$min_{th} = 10$ $max_{th} = 21$

5. Emulation results

Table 2. Final results (mean of three iterations)

AQM	A	B	C	D	E	F	G	H
RED 1	17.8	428	9.5	4.3	153	55	8802	6
RED 2	18.0	427	8.7	3.9	152	57	4390	5
RED 3	17.9	413	10.5	4.8	153	75	19096	8
ARED 1	17.9	411	9.7	4.3	152	64	1836	3
ARED 1	17.9	403	9.7	4.5	153	73	9158	7
BLUE 1	18.4	585	85.0	81.6	137	37	655366	20
BLUE 2	18.91	697	77.8	74.4	137	41	524294	18
BLUE 3	15.6	478	8.2	9.7	154	25	75782	10
GKVO 1	12.6	158	8.6	8.5	186	57	263306	15
GKVO 2	13.6	242	12.3	13.3	178	69	100358	12
GKVO 3	14.6	352	17.6	19.1	169	85	278560	17
AVQ 1	12.6	156	9.1	8.7	186	56	526432	19
AVQ 2	13.5	283	11.6	12.2	180	64	144400	14
AVQ 3	13.6	240	12.2	13.4	178	70	102404	13
PI 1	17.8	460	11.4	6.9	155	83	70912	9
PI 2	17.9	436	12.1	7.7	154	84	91664	11
PI 3	18.0	411	11.5	7.2	153	91	265028	16
AVORED1	17.1	364	6.9	5.9	152	44	1178	2
AVORED2	17.0	261	5.9	5.0	151	44	2100	4
AVORED3	17.4	372	6.1	4.5	153	41	686	1

A = Mean link utilization (Mbps)
 B = Standard deviation link utilization (Kbps)
 C = Mean queuing delay (msec)
 D = Standard deviation queuing delay (msec)
 E = Mean packet drop (packet drops/100 msec)
 F = Standard deviation packet drop (packet drops/100 msec)
 G = Rank weight
 H = Rank

Each setting in Table 1 was run for 15 minutes, three times. The three results are averaged and their standard deviations were also calculated. Three iterations of emulations were averaged to produce Table 2. Standard deviations (not shown due to the space limitation) of the three iterations were relatively small, thus validating the use of the mean values. Using the results in Table 2, the following were done:

1. All AQM methods in Table 2 were ranked for each

performance metric and the same weighted ranking rule discussed in the previous section was applied to produce the rank weights, column G in Table 2. This measurement indicates that the AVQRED ranked first and second.

- For each AQM method, the parameter setting that scored the best rank weight was picked and the results are compared with AVQRED in Figure 3, Figure 4, Figure 5, Table 3, Table 4 and Table 5.

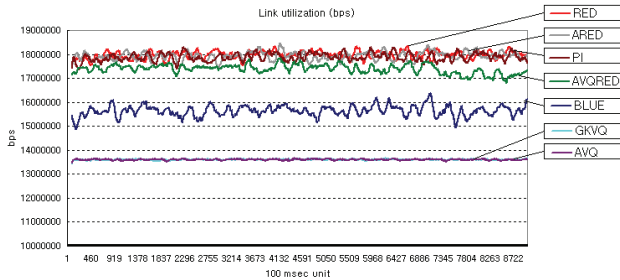


Figure 3. Link utilization

Table 3. Link utilization improvement by AVQRED

	RED	ARED	BLUE	GKVQ	AVQ	PI
Link Util Increase %	-3.2%	-2.7%	11.2%	27.7%	27.7%	-2.5%
Stdev Decrease %	12.8%	9.4%	22.0%	-54.2%	-55.3%	19.0%

Figure 3 and Table 3 show that AVQRED’s link utilization degrades by less than 5%, and 9% or more link utilization stability is gained. GKVQ and AVQ are exceptional cases because their link utilizations are well below the actual gateway capacity which resulted in lower variations than AVQRED.

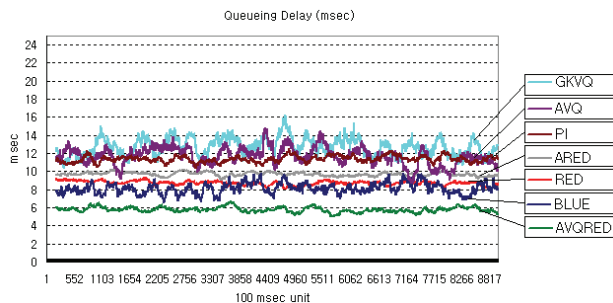


Figure 4. Queuing delay

Table 4. Queuing delay improvement AVQRED

	RED	ARED	BLUE	GKVQ	AVQ	PI
Q Delay Decrease %	30.0%	37.2%	25.1%	50.3%	49.8%	46.5%
Stdev Decrease %	-14.4%	-5.4%	53.5%	66.1%	66.3%	34.6%

Figure 4 and Table 4 show that AVQRED’s queuing delays are at least 25% lower than the other AQM methods while delay jitters are about 5 to 15% higher than RED and ARED. However, it is also important to note that having low delay jitters is not always good because it means that short-lived and bursty traffic could be dropped.

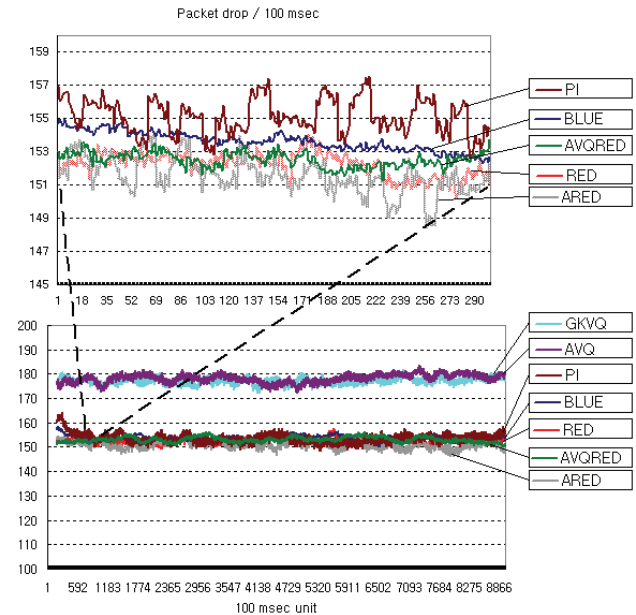


Figure 5. Packet loss

Table 5. Packet loss improvement by AVQRED

	RED	ARED	BLUE	GKVQ	AVQ	PI
Packet Drop Decrease %	0.1%	0.2%	0.9%	14.5%	14.4%	1.7%
Stdev Decrease %	28.0%	36.0%	-62.0%	41.0%	41.8%	50.8%

Figure 5 and Table 5 show that AVQRED’s packet loss rate is up to 15% lower and the standard deviation is 28 to 50% lower than the other AQM methods except for BLUE³. It is important to note that a low drop standard deviation implies that the distance between drops is more uniform resulting in less consecutive drops. About 30 second time period was magnified in Figure 5 to better visualize the packet loss behavior.

6. Model and analysis

This section discusses the mathematical model used for validating the emulation results. Although details on how

³ Higher queuing delays and higher queue depth resulted in lower packet drop rates and variations.

each equation is derived are not discussed here due to the space limitation, they can be found in [1], [2] and [5].

For simplicity, RED, ARED and AVQRED (which produced the best emulation results) were modeled with slight modifications to the models from [1], [2] and [5]. Note that the actual queue size ODE was modeled with M/M/1 assumption and the AVQRED queue size ODE was modeled with M/D/1 assumption to expose the main characteristic of the AVQRED algorithm which is deterministic service time. Our model does not capture some of the TCP dynamics such as TCP slow start, timeouts, variable packet size and variable propagation delay which caused a slight discrepancy between the emulation results and the MATLAB results. Even with such discrepancy, the MATLAB results are consistent with the emulation results with respect to the important characteristics (utilization, stability, unbiased drops against transient traffic, global synchronization level and queueing delay) of the AQM methods. The following ODEs are used to model RED, ARED and AVQRED, and the details on how they are derived are in [1], [2] and [5].

$$\begin{aligned}\frac{dx}{dt} &= \frac{\ln(1-\alpha)}{\delta} x(t) - \frac{\ln(1-\alpha)}{\delta} q(t) \\ \frac{dq}{dt} &= \lambda(t)(1-p(t)) - u \frac{q(t)}{1+q(t)} \\ \frac{d\lambda}{dt} &= (1-p(t)) \frac{m}{R^2} - p(t) \frac{\lambda^2}{2m} \\ \frac{ds}{dt} &= -\tau \cdot u + \lambda \cdot (1-p) \\ \frac{dl}{dt} &= u \cdot \left(\frac{q(t)}{1+q(t)} \right) - l(t-1)\end{aligned}$$

Variables:

$$\frac{dx}{dt} = \text{Weighted average queue size. } \frac{dx}{dt} \text{ from [5].}$$

$$\frac{dq}{dt} = \text{Actual Q size. Equation (6) from [1].}$$

$$\frac{d\lambda}{dt} = \text{Arrival rate. Equation (3) from [2].}$$

$$\frac{ds}{dt} = \text{AVQRED queue size from the Lindley equation.}$$

$$\frac{dl}{dt} = \text{Link utilization.}$$

Parameters:

α = Queue weight for weighted average

δ = Delta time between each packet arrival

λ = Aggregate arrival rate

p = AQM packet loss probability

u = Service rate with uniform distribution between [80%, 100%]

m = Number of concurrent TCP connections

R = Round trip delay

τ = Target link utilization %

Either q (when RED or ARED is used) or s (when AVQRED is used) is fed to the following equation from [5] to compute the marking probability:

$$p(x) = \begin{cases} 0, & 0 \leq x < \min_{th} \\ \frac{x - \min_{th}}{\max_{th} - \min_{th}} \max_p, & \min_{th} \leq x \leq \max_{th} \\ 1, & \max_{th} < x \end{cases}$$

One modification made to the models proposed in [1], [2] and [5] is the variation factor in the service rate. This could be due to many reasons but the most common reason is a single processing resource shared by multiple processes and threads. Emulation results also confirm that the variation is a valid assumption. From the emulation results, it was also confirmed that the servers we used for the emulations have a uniform distribution between 80% and 100% of the maximum service rate

7. Validation

Three AQM methods (AVQRED with setting 3, ARED with setting 1 and RED with setting 2) that have the best emulation results are fed to MATLAB using the equations provided in section 6. Then the validation is done by comparing the MATLAB results with the emulation results.

There are 200 TCP connections at the beginning and the traffic volume is increased by 200 connections every 5 seconds until 4000 connections are established. 5% of the population goes in and out of the system every second to resemble the traffic characteristics used for the emulations. The propagation delay is set to 200 msec and the average packet size is set to 1400 bytes. The comparison metrics are queue size, arrival rate and link utilization. Although we may not see perfectly matching graphs between the MATLAB results and the emulation results due to dynamics of traffic such as TCP slow start, timeouts, variable packet size and variable propagation delay, we are still able to show that the mathematical model captures the important characteristics of the three AQM methods such as utilization, stability, unbiased drops against transient traffic, global synchronization level and queueing delay.

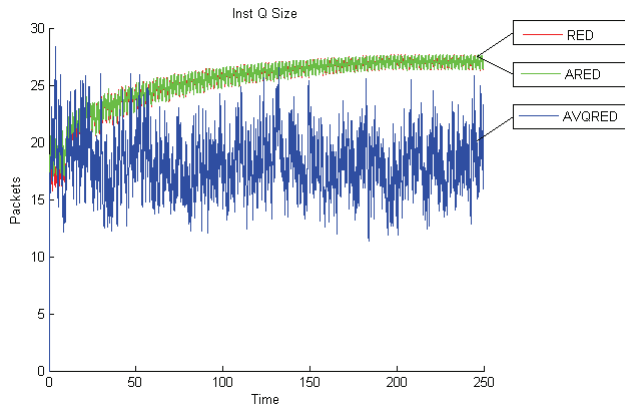


Figure 6. Queue size

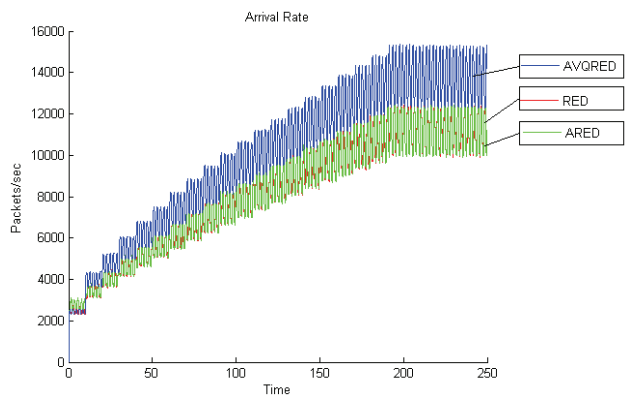


Figure 7. Arrival rate

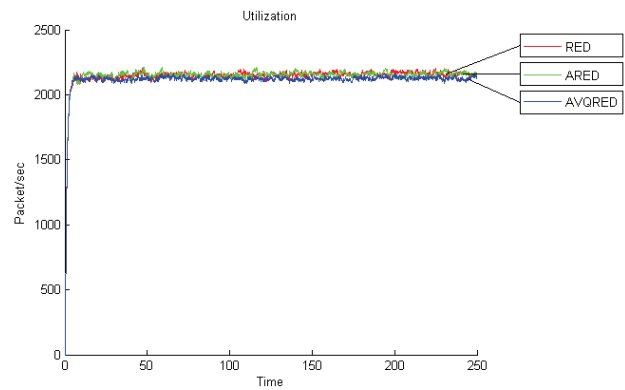


Figure 8. Link utilization

The MATLAB results for queue size, Figure 6, are consistent with the emulation results for queueing delay, Figure 4 and Table 4, because both show that the AVQRED has lower queueing delay with wider variation (which is due to unbiased drops against transient traffic). The MATLAB results for arrival rate, Figure 7, are consistent with the emulation results for packet loss variation, Figure 5 and Table 5, because a higher arrival rate implies less number of consecutive drops which cause significant

degradations in TCP window evolution. The MATLAB results for link utilization are also consistent with the emulation results, Figure 3 and Table 3, because both show that AVQRED has more stable utilization with slightly less utilization although the difference in stability is smaller in the MATLAB results. We believe that this smaller difference in stability is due to the fact that some of the traffic dynamics such as TCP slow start, timeouts, variable packet size and variable propagation delay are not captured by the model.

8. Conclusion

A new rate-based AQM method, AVQRED, was developed to improve performance of congested gateways. AVQRED essentially combines AVQ and RED and enhances the way virtual capacity is adjusted to adapt to dynamics of gateway resources. AVQRED was compared with six other AQM methods, RED, ARED, BLUE, GKVQ, AVQ and PI. Comparisons between the seven AQM methods were carried out using a realistic emulation environment created by a traffic generator, *Spirent*[®]. For fair comparisons, all AQM parameters were fine-tuned via emulations. The performance metrics used for the comparisons were link utilization, queueing delay and packet loss.

Table 6 shows the overall improvement made by AVQRED with an assumption that each metric is equally important. The overall improvement was computed by averaging all the improvement measurements in Table 3, Table 4 and Table 5.

Table 6. Overall improvement by AVQRED

	RED	ARED	BLUE	GKVQ	AVQ	PI
Overall Improvement	8.8%	12.4%	8.5%	24.2%	24.1%	25.0%

Comparisons between AVQRED and the other six AQM methods can be concluded as:

1. AVQRED has slightly lower link utilization with higher stability.
2. AVQRED has lower queueing delay with higher delay jitter but it is less biased against short-lived traffic.
3. AVQRED has slightly fewer packet drops with lower global synchronization level.

To provide intuitive properties and validate the emulation results, a mathematical model was proposed and fed to MATLAB. The MATLAB results were compared with the emulation results and they were

consistent each other.

We are currently evaluating the use of AVQRED in satellite networks where queue size based AQM methods do not fit well due to high buffering.

10. References

- [1] P. Lassila and J. Virtamo, "Modeling the dynamics of the RED algorithm," in *Proceedings of QoIS'00*, pp. 28-42, September 2000.
- [2] P. Kuusela, P. Lassila, J. Virtamo and P. Key, "Modeling RED with idealized TCP sources," <http://research.microsoft.com/~peterkey/Papers/ifipredtcp.pdf>, 2001.
- [3] S. Athuraliya, V. H. Li, S. H. Low, and Q. Yin, "REM: Active queue management," *IEEE Network*, vol. 15, no. 3 pp. 48-53, May/June 2001.
- [4] S. Floyd and V. Jacobson, "Random early detection gateways in congestion avoidance," *IEEE/ACM Transactions on Network*, vol. 1 no. 3, pp.397-413, 1993.
- [5] V. Misra, V. Gong, and D. Towsley, "A fluid-based analysis of a network of AQM routers supporting TCP flows with an application to RED," in *Proceedings of ACM SIGCOMM*, August 2000, pp. 151-160.
- [6] S. Floyd et al., "Discussions on setting parameters," <http://www.aciri.org/floyd/REDparameters.txt>, November 1998.
- [7] S. Floyd, R. Gummadi, S. Shenker, and ICSI. Adaptive RED: An algorithm for increasing the robustness of RED's active queue management, Berkeley, CA. <http://www.icir.org/floyd/red.html>.
- [8] W. Feng, D. Kandlur, D. Saha, and K. Shin, "Blue: A new class of active queue management algorithms," Tech. Rep., UM CSE-TR-387-99, 1999.
- [9] W. Feng, D. Dilip D. Kandlur, Debanjan Saha, and Kang G Shin, "A self-configuring RED gateway," in *Proceedings of IEEE/INFOCOM*, 1999.
- [10] D. Clark and W. Fang, "Explicit allocation of best-effort packet delivery service," *IEEE/ACM Transactions of Networking*, vol. 6 no. 4 pp. 362-373, August 1998.
- [11] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. "Modeling TCP throughput: A simple model and its empirical validation," in *Proceedings of ACM/SIGCOMM*, 1998.
- [12] T. J. Ott, T. V. Lakshman, and L. H. Wong, "SRED: stabilized RED," in *Proceedings of IEEE INFOCOM*, March 1999.
- [13] W. Stevens. *TCP/IP Illustrated, Vol. 1 The Protocols*. Addison-Wesley, 1994.
- [14] W. Stevens, "TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms," RFC2001, Jan 1997.
- [15] M. May, T. Bonald, and J. Bolot, "Analytic evaluation of RED performance," in *Proceedings of IEEE INFOCOM*, March 2000.
- [16] C. Hollot, V. Misra, D. Towsley, and W-B. Gong, "A control theoretical analysis of RED," in *Proceedings of IEEE INFOCOM*, 2001.
- [17] C. Hollot, V. Misra, D. Towsley, and Wei-Bo Gong, "On designing improved controllers for AQM routers supporting TCP flows," in *Proceedings of IEEE/INFOCOM*, April 2001.
- [18] S. Kunniyur and R. Srikant, "Analysis and design of an adaptive virtual (AVQ) algorithm for active queue management," in *Proceedings of ACM/SIGCOMM*, August 2001.
- [19] K. K. Ramakrishnan and S. Floyd, "A proposal to add explicit congestion notification (ECN) to IP," RFC 2481, Jan. 1999.
- [20] R. J. Gibbens and F. P. Kelly, "Distributed connection acceptance control for a connectionless network," in *Proceedings of the 16th Intl. Teletraffic Congress*, June 1999.