Adaptive Virtual Queue Random Early Detection in Satellite Networks

Do Jun Byun and John S. Baras

Abstract Networks with scarce bandwidth and high propagation delays cannot afford to have an unstable active queue management (AQM). In this paper, problems with applying existing AQMs to satellite networks are identified and solved. The first problem is oscillatory queuing, which is caused by high buffering due to performance enhancing proxy (PEP) in satellite networks where congestion control after the PEP buffering does not effectively control traffic senders. The second problem is global synchronization due to tail-drop nature of virtual queue-based AQMs. A new AQM method called adaptive virtual queue random early detection (AVQRED) is proposed to solve the problems, and it is validated using a realistic emulation environment and a mathematical model.

1 Introduction

Due to exponential increases in Internet traffic, active queue management (AQM) has been heavily studied by numerous researchers. However, little is known about AQM in satellite networks. A microscopic examination of queuing behavior in satellite networks is conducted to identify problems with applying existing AQM methods. A new AQM method is proposed to overcome the problems and it is validated with a realistic emulation environment and a mathematical model.

Internet Protocol (IP) over Satellite (IPoS) has been commercially available for the last few decades. Due to its high availability and mobility, IPoS has been attractive to areas where terrestrial services are not available as well as enterprises with scattered branch offices. One big barrier that IPoS has faced is its high propagation delay between earth stations and satellite. A typical round trip time (RTT) for a two-way geosynchronous satellite is around 600 ms.

D.J. Byun (✉)
Institute of Systems Research, University of Maryland at College Park
e-mail: dbyun@hns.com

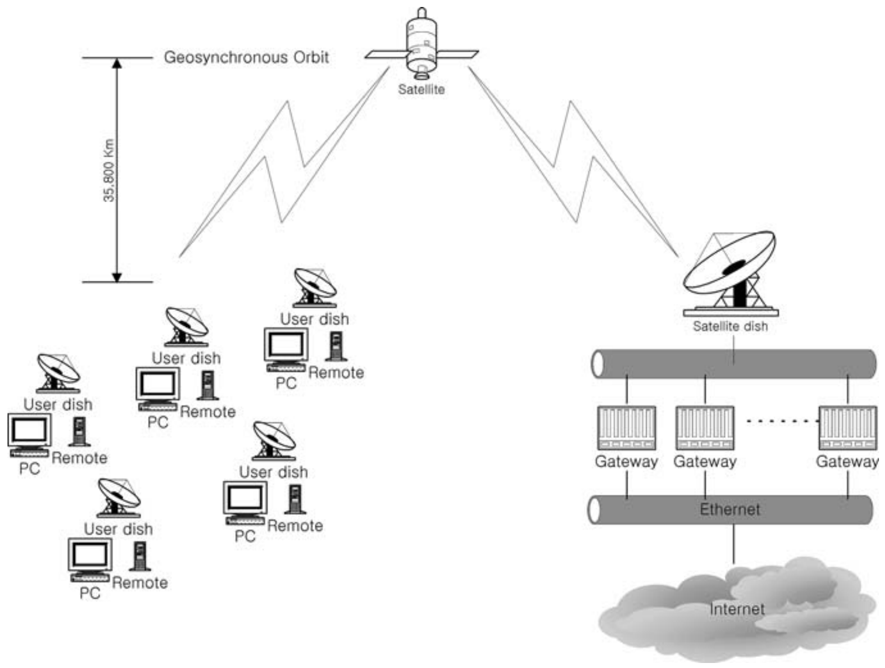

Fig. 1 IPoS system architecture

Figure 1 illustrates the system architecture of a typical two-way IPoS system where half of the 600 ms RTT occurs between the gateways and the satellite, the other half occurs between the remotes and the satellite. The biggest problem with such high propagation delay is the TCP performance. One aspect of the problem is the TCP slow start [1] phase where it takes a long time (RTT $\times \log_2 \times$SSTHRESH) to reach the maximum congestion window threshold (maximum rate at which the sender sends traffic) and the other aspect is that the maximum throughput of $65{,}535 \times 8/\text{RTT}$ is too low when the TCP Window Scale option is not supported. Even when the TCP Window Scale option is supported, unless all nodes support the option, fair bandwidth sharing becomes an issue. TCP Spoofing or performance enhancing proxy (PEP) [2] has been practiced by most of IPoS service providers to overcome this problem with TCP. For consistency, the term PEP will be used throughout this paper. The basic idea of PEP is to buffer at least one-round trip worth of data by locally acknowledging the data. Usually buffering only one-round trip worth of data is not enough because one has to account for queuing delays associated with congestion and inroute bandwidth allocations.

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 (ECN) [3] bits or signal congestion to traffic sources implicitly by dropping packets. ECN is not used in our study due to the following reasons:

1. The problems that we are trying to solve are not due to packet drops between gateways and senders.
2. ECN marking after PEP (transmit queue in Fig. 3) may seem to avoid retransmissions over satellite and fix the queuing instability problem discussed later, but it is too late to enforce ECN bits when data are already acknowledged without ECN bits by PEP.

When applying AQM to satellite networks, the following need to be considered:

1. The source of congestion is different in satellite networks, i.e., in satellite networks, congestion arises mainly due to the satellite link capacity, not due to the processing capacity. Therefore, gateways in satellite networks become congested when the offered load is greater than the allowed transmit rate, whereas gateways in terrestrial networks often become congested when the offered load is greater than the processing capacity.
2. Monitoring and marking packets after PEP is not a good idea because it involves retransmissions over satellite.
3. Monitoring (with real-queue-based AQM) and marking packets before PEP is not a good idea because the receive queue will never be congested when the congestion bottleneck is the spacelink capacity, not the processing capacity. This is not true for virtual-queue-based [4] AQMs such as adaptive virtual queue (AVQ) [5].

To address the above concerns, a new virtual-queue-based AQM, Adaptive virtual queue random early detection (AVQRED), was previously proposed [6, 7] and validated with realistic emulations. In this chapter, we extend the study by constructing a mathematical model and further validate the solution by comparing the MATLAB results with the emulation results.

2 Overview of PEP

PEP enhances the TCP performance by locally acknowledging one+ round trip worth of TCP data at the gateway over terrestrial links. Although there can be many different flavors of PEP, the core idea of buffering up one+ round trip worth of TCP data remains the same.

Figure 2 illustrates the end-to-end PEP flows in a two-way satellite network. To better visualize PEP in a gateway, Fig. 3 is provided. PEP is drawn in a typical gateway structure where PEP processes packets after the receive queue and the transmit queue resides after PEP. In Fig. 3, the congested queue is the transmit queue as discussed in the previous section.

It is common that PEP is also implemented in remote terminals to gain higher upload speeds and to keep the implementation symmetric, but congestion avoidance

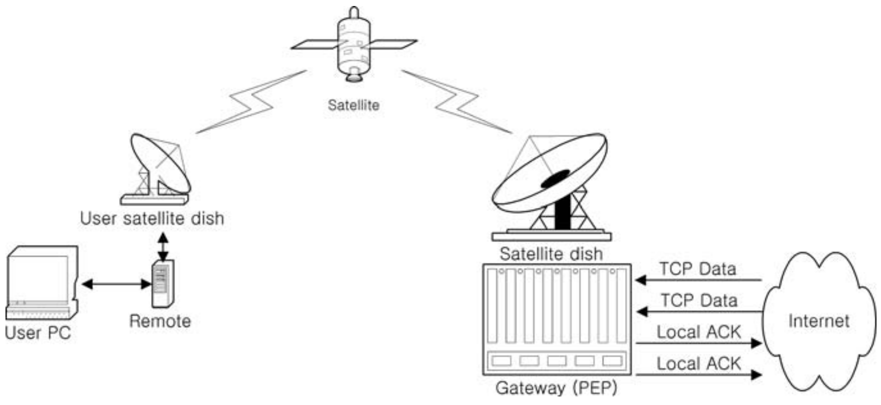

Fig. 2 PEP flows

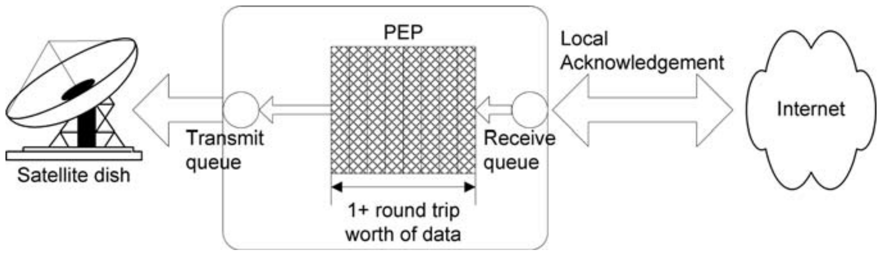

Fig. 3 Gateway with PEP

in the upload direction (from remote terminals to internet) is not discussed in this chapter as it involves different congestion paths.

3 Overview of AQM Methods

This section provides a high-level overview of two well-known AQM methods: random early detection [8] and adaptive virtual queue which will be compared with AVQRED via emulations and MATLAB.

3.1 RED

The RED [8] 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 is greater than max_{th}), and it also becomes higher as the distance between each marking gets larger. 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 or too slowly may result in queuing instability. max_p is a scaling factor for the marking probability which also controls how quickly the algorithm reacts to congestion

```
Initialization:
avg = 0
count = -1
for each packet arrival
if the queue is nonempty
   avg =$(1-w_q)$avg +$w_q$.q
else
   m = f(time-q_time)
   avg = $(1-w_q)^m$avg
if $min_{th}$ <= avg < $max_{th}$
   increment count
   calculate probability pa:
       $p_b$ = $max_p$(avg-minth)/($max_{th}$-$min_{th}$)
       $p_a$ = $p_b$ / (1-count.$p_b$)
    with probability $p_a$:
       mark the arriving packet
       count = 0
 else if $max_{th}$ <= avg
    mark the arriving packet
    count = 0
 else count = -1
 when queue becomes empty
 q_time = time
```

RED algorithm

3.2 AVQ

Gibbens–Kelly virtual queue (GKVQ) [4] 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) 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. Configured parameters are γ (target utilization), C (real capacity), and B (virtual queue limit).

```
At each packet arrival epoch do
/* Update Virtual Queue Size  */
```

```
VQ  = max (VQ - C′(t - s), 0)
If VQ + b > B
   Mark or drop packet in the real queue
else
   /* Update Virtual Queue Size */
   VQ = VQ + b
endif
/* Update Virtual Capacity */
C′ = max (min (C′+α*γ*C*(t-s),C) - α*b, 0)
/* Update last packet arrival time */
s = t

Variables:

B  = buffer size
s  = arrival time of previous packet
t  = current time
b  = number of bytes in current packet
VQ = number of bytes currently in the virtual queue
C′ = virtual capacity
C  = actual capacity
```

AVQ algorithm

4 Problems

The main objective we are trying to achieve is to avoid retransmissions over satellite, maintain queuing stability, and avoid global synchronization (consecutive packet drops) while preserving high link utilization. To avoid retransmissions over satellite, the option of dropping packets after PEP was not considered. Because real-queue-based AQMs such as RED can detect congestion only if it monitors the congested queue, RED monitoring is done in the transmit queue, while the packet marking is done in the receive queue to avoid retransmissions over satellite. Because virtual-queue-based AQMs such as AVQ can detect congestion regardless of the location of monitoring, both monitoring and marking are done in the receive queue to best synchronize packet marking and congestion detection by senders. Therefore, the following configurations are used throughout the emulations (Table 1).

Table 1 AQM Q and marking Q configuration

AQM method	Monitor Q	Marking Q
RED	Transmit queue	Receive queue
AVQ	Receive queue	Receive queue
AVQRED	Receive queue	Receive queue

4.1 Asynchronous Queuing Behavior

The problem with real-queue-based AQMs such as RED in satellite networks is synchronization between the monitored queue and the traffic senders. Synchronizing them is very difficult due to the high buffering that occurs between them, i.e., dropping a packet at the receive queue due to congestion in the transmit queue does not immediately reduce the congestion level of the transmit queue resulting in unwanted packet drops until the PEP buffers are all transmitted. These packet drops then result in less queue occupancy until senders' congestion windows evolve causing oscillatory queuing behavior. Figure 4 illustrates how an asynchronous queuing can occur. Note that the packets are consecutively dropped from T1 through T6 because the transmit queue is always occupied by the packets from the PEP layer. After PEP buffers are all used up, the transmit queue becomes almost empty and the PEP starts building up its buffers at T7. Until there are enough PEP buffers, the transmit queue does not drop packets at the receive queue causing oscillatory queuing behavior.

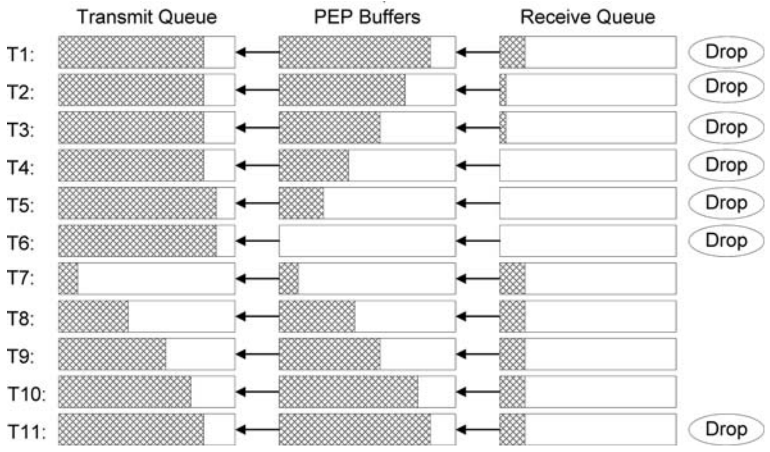

Fig. 4 Asynchronous queuing behavior

4.2 Global Synchronization

The problem with AVQ is global synchronization where consecutive packet drops occur due to its tail-drop nature of packet marking. When packets are dropped consecutively, multiple TCP connections will react to the drops simultaneously resulting in oscillatory link utilization among multiple TCP connections.

This problem is severe with RED due to its oscillatory queuing behavior described in the previous section. When the transmit queue congestion level and the senders congestion windows are not synchronized, the RED region will likely be exceeded resulting in tail-drop behavior.

5 Solution

A new AQM algorithm, adaptive virtual queue random early detection, is proposed to address the asynchronous queuing and the global synchronization problems.

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

   /* # of bytes that could have been transmitted */
   serviced_bytes <- v_capacity / 1000 / 8 * $\delta$

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

last_measure <- curr_time
q_size <- VQ / 1500

/* Feed VQ size to the RED algorithm */
if $min_{th}$ < q_size < $max_{th}$
   count <- count  + 1
   $p_b$ <- (q_size - $min_{th}$) /($max_{th}$ - $min_{th}$)
   $p_a$ <- $p_a$ / (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
```

AVQRED Algorithm

The AVQRED algorithm constructs a virtual queue and feeds the virtual queue size to the RED algorithm instead of feeding the weighted average queue size to it. By doing so, AVQRED essentially moves the transmit queue to the receive queue and produces better synchronization between the transmit queue and the traffic sources. 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 above 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. α is a low-pass filter for the actual capacity calculation. *min_capacity* and *max_capacity* define the range of processing capacity. For satellite networks where processing capacity is greater than spacelink capacity, *min_capacity* should be equal to *max_capacity* and α can be any value.

5.1 Asynchronous Queuing Behavior

AVQRED solves the asynchronous queuing problem by both monitoring and marking at the receive queue. Monitoring and marking at the receive queue is possible because AVQRED constructs a virtual queue which can be placed anywhere.

5.2 Global Synchronization

AVQRED solves the global synchronization problem by preserving global synchronization avoidance of the RED algorithm. Emulation results are provided to illustrate this point.

6 Emulation Framework

The actual gateway software, IP Gateway, from *Hughes Network Systems* was used to evaluate the AQM methods. The three AQM methods were implemented according to Table 2. The emulation environment was constructed using two IP Gateways (one serves as the actual IP Gateway that faces the Internet and the other serves as the satellite terminals for *N* different users) and a traffic generator called *Spirent*. A high-level illustration of the gateway internal structures is shown in Fig. 3. Both server and client IP Gateways have the same PEP code and some modifications to the software were done to resolve address translation and routing issues created by the client IP Gateway. Details of the modifications are not discussed here as they are

Table 2 AQM settings

AQM	Parameters			
	min_{th}	max_{th}	w_q	max_p
RED 1	60	120	0.02	0.5
RED 2	60	120	0.05	0.7
RED 3	60	120	0.10	0.5
RED 4	60	120	0.10	0.7
	γ	B		
AVQ	100%	123,750 Bytes		
	min_{th}	max_{th}	*min_capacity*	*max_capacity*
AVQRED	60	120	20 Mbps	20 Mbps

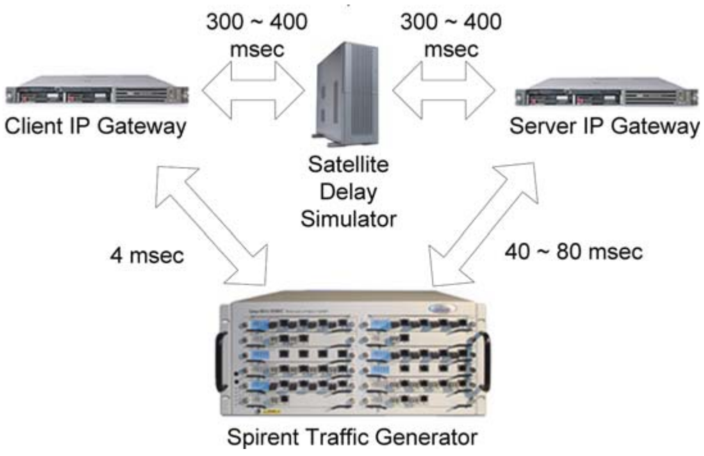

Fig. 5 Emulation flow

not relevant to the interest of this research. *Spirent* was used to best emulate real-life traffic characteristics.

Figure 5 illustrates the connectivity of the emulation setup. All links are lossless and 100 Mbps full duplex. A delay simulator was inserted between the two gateways to simulate satellite delays with uniform distribution between 300 and 400 ms each way. The round trip time (RTT) between the client IP Gateway and the *Spirent* is 4 ms, the RTT between the client IP Gateway and the server IP Gateway is 600–800 ms, and the RTT between the server IP Gateway and the Spirent is 40–80 ms resulting in an end-to-end RTT of 644–884 ms. Four hundred HTTP connections were generated between 200 clients and 60 servers with the following attributes.

1. At the startup, there are 20 new HTTP connections every 5 s with 5 s sleep time between each ramp up until 400 HTTP connections are established.
2. When a connection is closed, a new connection is created to fill the gap to maintain 400 HTTP connections.

3. Each web page contains 250–550 Kbytes of data with 10 s user think time.
4. The maximum download speed of each TCP connection is 5 Mbps.
5. Average birth and death rate of the connections is about 20 connections per second (approximately 5% of the total population).

6.1 Evaluation Methodologies

The following performance metrics were used for validation:

1. Link utilization – The purpose of this metric is to make sure that the proposed solution produces comparable link utilizations.
2. Queue size – The purpose of this metric is to compare queue size and queuing stability of each AQM method.
3. Packet drop – The purpose of this metric is to compare consecutive packet drops of each AQM method.

The measurements were taken after all 400 HTTP connections are established to best emulate a loaded scenario.

6.2 Parameter Settings

The following system parameters were used throughout the emulations:

- 20 Mbps downlink bandwidth (gateway to terminal direction).
- 1 Mbps uplink bandwidth (terminal to hub direction). This link is assumed to be non-congested link because the application is downlink-oriented web browsing.
- 5 ms transmit rate regulator latency in the server IP Gateway.
- Target transmit queuing delay of 33 ms.
- Average packet size is 1,400 bytes for the downlink direction.

For RED, there are four parameters to configure: min_{th}, max_{th}, max_p, and w_q. Sixty and 120 are configured for min_{th} and max_{th}, respectively. min_{th} is set slightly higher than $59(=20\text{ Mbps}/8/1{,}400 \times 0.033$ from the system parameters) to ensure full utilization of 20 Mbps bandwidth. max_{th} is set to at least twice min_{th} as [8] recommends. Several permutations of max_p and w_q were emulated as these parameters need to be fine-tuned according to traffic characteristics as shown in Table 2.

For AVQ, the target utilization, γ, is set to 100%, and the buffer size, B, is set to 123,750 bytes where $123{,}750 = 82{,}500\,(= 20\text{ Mbps}/8 \times 0.033) + 82{,}500/2$. Half of the buffer required for 20 Mbps (82,500/2) is added to ensure full utilization. The α is set to an arbitrary number as our optimal virtual capacity is pre-determined.

For AVQRED, 60 and 120 are chosen for min_{th} and max_{th}, respectively with the same reason as RED; α is set to an arbitrary number as our optimal virtual capacity is pre-determined. Target utilization is set to 100% by setting *min_capacity* = *max_capacity* = 20 Mbps.

7 Emulation Results

Each of the AQM settings in Table 2 was emulated for 20 min with the traffic described in the previous section. The link utilization, queue size, and consecutive packet drop were measured once every 100 ms and the following subsections discuss the results for each of the measurement metrics. Due to the space limitation, only RED 4 (which had the best results amongs the RED settings), AVQ, and AVQRED are presented.

7.1 Link Utilization

As Figs. 6 and 7 and Table 3 show, the utilization of AVQRED is comparable with the utilization of RED. Although there is about 0.5% loss in the mean utilization, there is about 0.25% gain in the stability (the standard deviation).

Utilization loss and stability gain can be explained by the queuing behavior of RED and AVQRED which is discussed more in the next section. Basically, AVQRED maintains just enough data to fill up the 20 Mbps pipe whereas RED's utilization is oscillatory and unstable due to its asynchronous queuing behavior discussed in the previous sections. Furthermore, RED's high utilization and low stability indicate that it tends to accept more data than the gateway capacity.

Although AVQ's algorithm is similar to AVQRED's in terms of approximating the virtual capacity, its utilization is lower than AVQRED. This result is consis-

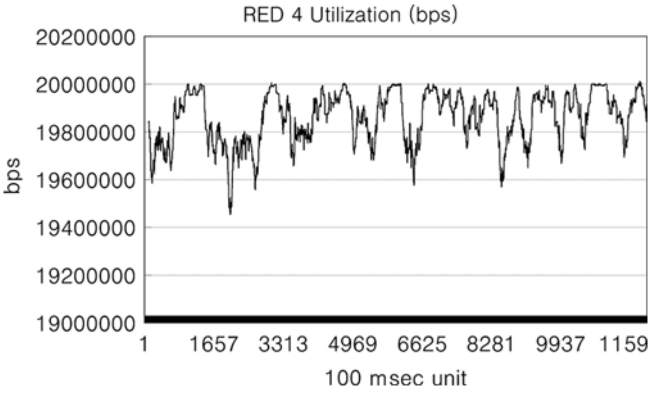

Fig. 6 RED 4 utilization

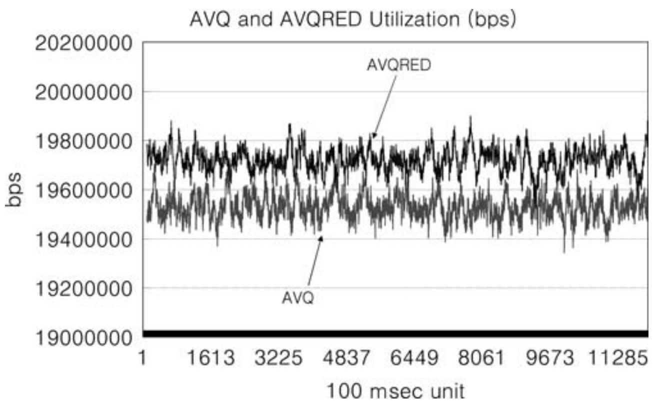

Fig. 7 AVQ and AVQRED utilization

Table 3 Link utilization mean and standard deviation

AQM	Mean (Mbps)	Standard deviation (Kbps)
RED 1	19.8	135
RED 2	19.8	117
RED 3	19.8	108
RED 4	19.8	107
AVQ	19.5	49
AVQRED	19.7	47

tent with the fact that AVQ has more consecutive packet drops because consecutive packet drops cause multiple senders to shrink their congestion windows synchronously resulting in lower link utilization.

7.2 Queue Size

Figures 8, 9, and 10 show the transmit queue size of RED, AVQ, and AVQRED. The queue size of RED is higher than AVQ and AVQRED because of its tendency to exceed the RED region (60–120) due to its oscillatory queuing behavior. To provide a better visualization of this point, Figs. 11 and 12 magnify Figs. 8 and 10 between 50th and 150th s (500th–1500th points according to the x-axis' scale).

This oscillatory queuing behavior is the asynchronous queuing behavior described earlier which is resulted from high PEP buffering between the transmit queue and the receive queue. Therefore, we can conclude that AVQRED and AVQ solve the asynchronous queuing problem by both monitoring and dropping at the receive queue. As discussed earlier, monitoring the receive queue with a real-queue-based AQM such as RED can not be done because the receive queue will never be

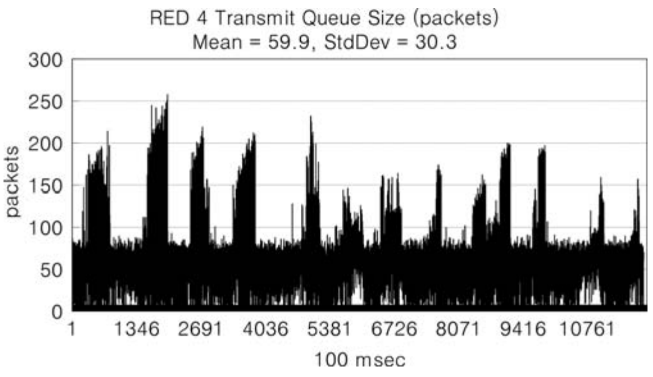

Fig. 8 RED 4 transmit queue size

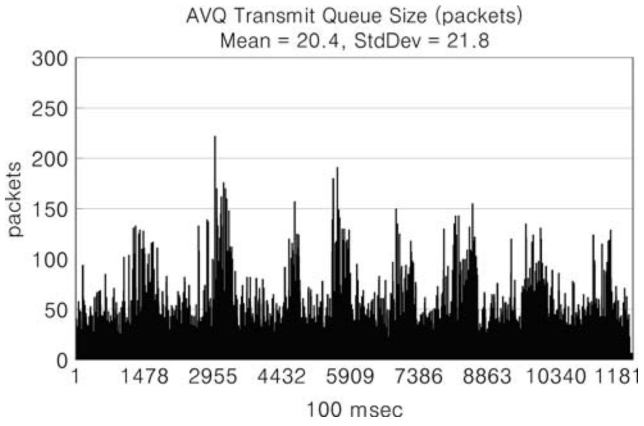

Fig. 9 AVQ transmit queue size

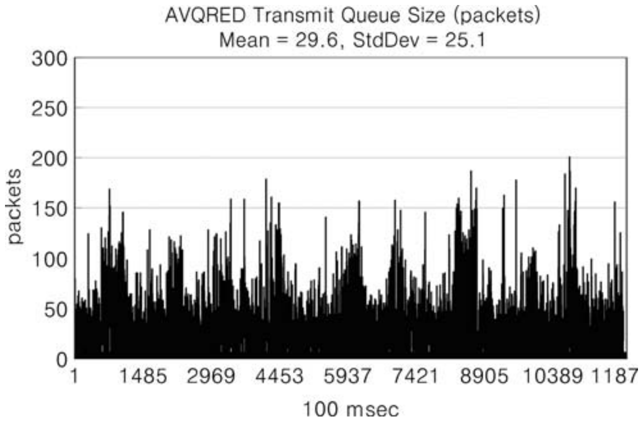

Fig. 10 AVQRED transmit queue size

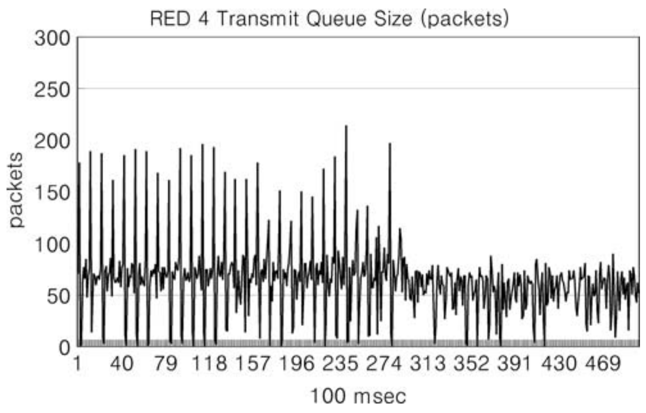

Fig. 11 RED 4 transmit queue size (50th–100th s)

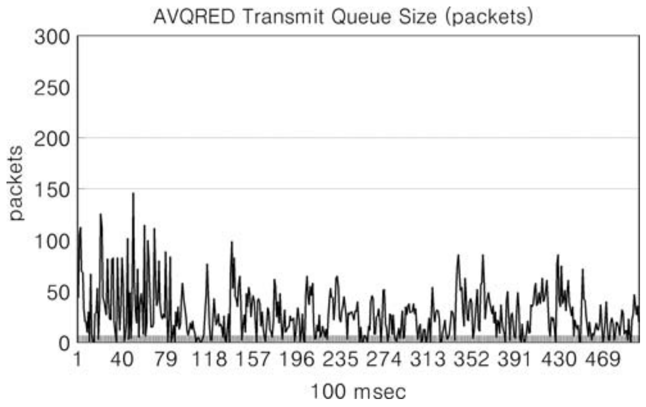

Fig. 12 AVQRED transmit queue size (50th–100th s)

congested when the bottleneck is the transmit queue by the spacelink bandwidth limitation.

7.3 Packet Drop

Because 20 min worth of the packet drop histogram is too long to present, only the first 1,000 packets are presented to show how packet drops are distributed. This illustration is valid because AVQRED has the least number of packet drops as shown in Table 4.

Figure 13 shows packet drops for the first 1,000 packets. Given that AVQRED has the least number of packet drops, having the least clustered packet drops proves that AVQRED has the least global synchronization level. RED packet drops are

Table 4 Total packet drops

AQM	Total packet drops
RED 1	486,932
RED 2	491,798
RED 3	492,025
RED 4	492,999
AVQ	484,639
AVQRED	484,582

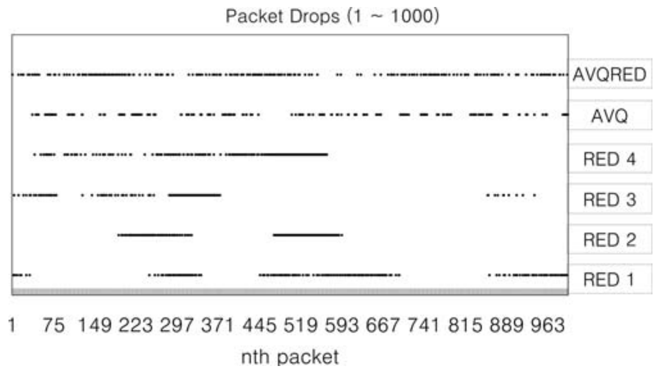

Fig. 13 Packet drops for 1st–1000th packets

more clustered than they should be due to the queuing oscillation discussed in the previous section.

From the data shown in this section, we can conclude that AVQRED solves the global synchronization problem of AVQ and RED by dropping packets more uniformly

8 Mathematical Model

This section provides a mathematical model for the queuing behavior to validate the asynchronous queuing problem and our solution, AVQRED. The global synchronization problem is not validated mathematically due to the space limitation. However, the same model can be used to show the marking behavior by analyzing the standard deviation of the marking probability:

$$\frac{d\lambda}{dt} = (1 - p(t)) \times \frac{m}{R^2} - p(t) \times \frac{\lambda^2}{2m} \tag{1}$$

$$\frac{dq}{dt} = -\mu + (1 - p(t - d(t))) \times \omega(t - d(t)) \times \lambda(t - d(t)) \tag{2}$$

$$\frac{dv}{dt} = -\tau \times \mu + (1 - p(t)) \times \omega(t) \times \lambda(t). \tag{3}$$

$$\frac{dw}{dt} = \frac{\log(1-B)}{\delta} \times w(t) - \frac{\log(1-B)}{\delta} \times q(t) \tag{4}$$

$$\frac{dp}{dt} = \begin{cases} -1.0 \times p(t) & \text{if } w(or\ v) < min_{\text{th}} \\ \frac{p_{max}}{(max_{\text{th}} - min_{\text{th}})} \times \frac{dq}{dt} & \text{if } min_{\text{th}} \le w \le max_{\text{th}}\ for\ RED \\ \frac{p_{max}}{(max_{\text{th}} - min_{\text{th}})} \times \frac{dv}{dt} & \text{if } min_{\text{th}} \le v \le max_{\text{th}}\ for\ \text{AVQRED} \\ 1.0 - p(t) & else \end{cases} \tag{5}$$

Equation (1) is the ODE of the arrival rate of the offered load where *R* is the RTT between the gateway and the Internet hosts and *m* is the number of TCP connections. Ref. [9] has the details on how it is mathematically derived. In our MATLAB experimentation, *R* was scaled down by 1/10 due to our time unit conversion from 1 s to 100 ms.

Equation (2) is the ODE of the transmit queue size where μ is the service rate (20 Mbps with 1,400 bytes per packet and 100 ms time unit), *p(t)* is the marking probability, *d(t)* is the Fourier series of the PEP buffering delays, and *ω(t)* is the Fourier series of the offered load variation. The ODE is derived from the Lindley equation and the delay factor was added to it to capture the PEP buffering effect. To best resemble our traffic model used for the emulations, Fourier series with 500 actual data points were used. For *d(t)*, the data points are the average duration that each PEP packet resides in the buffer during a 100 ms measurement period. For *ω(t)*, the data points are the mean offered load to the actual offered load ratio. All the data points were measured without AQM and bottlenecking transmit queue to avoid any feedback effects caused by AQM.

Equation (3) is the ODE of the AVQ size where τ is the target utilization. Note that it is similar to (2) except that it does not have the PEP buffering delays.

Equation (4) is the ODE of the weighted average transmit queue size from [10]. *B* is w_q and δ is the smallest time unit of our ODE approximation which is 1 ms (= 0.01 of 100 ms).

Equation (5) is the ODE of the marking probability which is just the first derivative of *p(t)* when the respective queue size (*q* or *v*) falls between min_{th} and max_{th}. For AVQ, this needs to be changed slightly,
i.e., $-1.0 \times p(t)$ if $v < (max_{th} - min_{th})/2$, and $1.0 - p(t)$, otherwise.

To validate the asynchronous queuing problem, the above ODEs were fed to MATLAB and the transmit queue size (2) was examined for both RED (with RED 4 parameters in Table 2) and AVQRED. As Fig. 14 shows, RED has the same oscillatory queuing behavior as the one that Fig. 8 shows. The mean is slightly lower than the emulation because the ODEs did not account for the 5 ms queuing latency caused by the output rate regulation. The standard deviation is slightly higher than the emulation because the Fourier series for the PEP buffering delays was

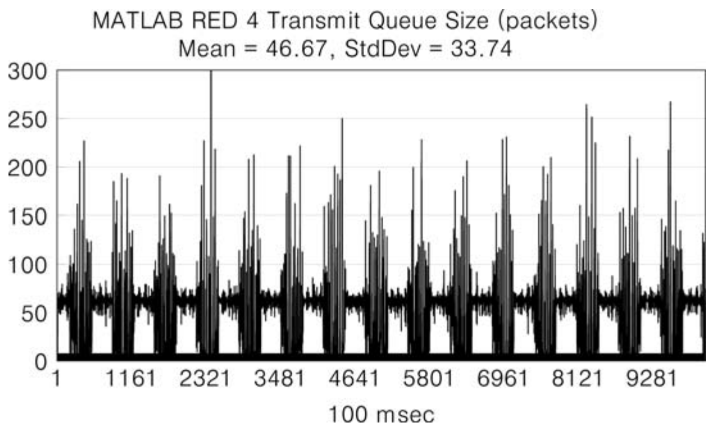

Fig. 14 RED 4 transmit queue size (MATLAB)

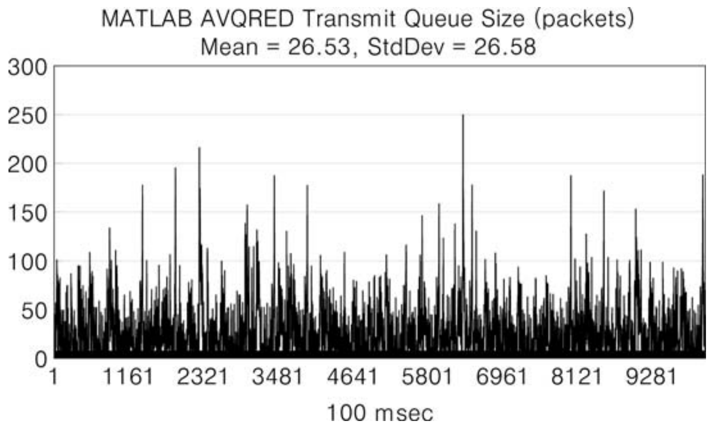

Fig. 15 AVQRED transmit queue size (MATLAB)

approximated using only 500 data points which resulted in more frequent and regular oscillations.

As Fig. 15 shows, AVQRED fixes the oscillatory queuing behavior. The mean and standard deviation are slightly different from the emulation because of the 5 ms latency and the relatively small Fourier sample space.

To summarize and compare the improvement percent of mean and standard deviation, Fig. 16 is provided. Figure 16 depicts that the MATLAB results concur with the emulation results. As stated earlier, the small discrepancies between emulation and MATLAB are from the 5 ms rate regulator latency and the small Fourier sample space.

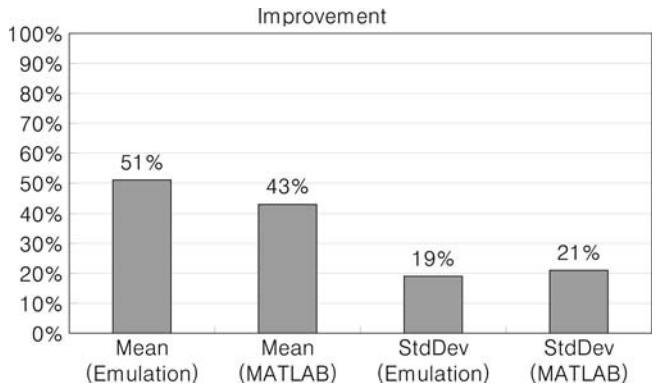

Fig. 16 Transmit queue size improvement by AVQRED

9 Conclusion and Future Work

In an effort to improve the gateway performance of satellite networks, AQM was applied to satellite networks. This study found that applying existing AQMs such as RED and AVQ has unwanted side effects: asynchronous queuing and global synchronization.

Emulations were conducted to validate the problems and the solution. The emulation environment was constructed with the real gateway software used in *Hughes Network Systems* and a traffic generator called *Spirent*.

A mathematical model was constructed to provide intuitive illustrations of the problems and the solution. The model was fed to MATLAB and the results concurred with the emulation results.

Because scarce bandwidth resources such as satellite require a good QoS handling, our future work will impose QoS as another measurement metric and involve enhancing the AVQRED algorithm to account for QoS related requirements.

References

1. W. Stevens, "TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms," RFC2001, January 1997.
2. J. Border, M. Kojo, J. Griner, G. Montenegro and Z. Shelby, "Performance enhancing proxies intended to mitigate link-related degradations," RFC3135, June 2001.
3. K. Ramakrishnan and S. Floyd, "A proposal to add explicit congestion notification (ECN) to IP," RFC 2481, January 1999.
4. 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.
5. 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.
6. D. Byun and J. Baras, "Adaptive virtual queue random early detection in satellite networks," in *Proceedings of Wireless Telecommunication Symposium*, April 26–28, 2007.

7. D. Byun and J. Baras, “A new rate-based active queue management: adaptive virtual queue RED,” in *Proceedings of the Fifth Annual Conference on Communication Networks and Services Research*, New Brunswick, Canada, May 14–17, 2007.
8. S. Floyd and V. Jacobson, “Random early detection gateways in congestion avoidance,” *IEEE/ACM Trans. Netw.*, 1(3), 397–413, 1993.
9. P. Kuusela, P. Lassila, J. Virtamo and P. Key, “Modeling RED with idealized TCP sources,” http://research.microsoft.com/˜peterkey/Papers/ ifipredtcp.pdf, 2001.
10. 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.
11. P. Lassila and J. Virtamo, “Modeling the dynamics of the RED algorithm,” in *Proceedings of QofIS’00*, September 2000, pp. 28–42.
12. 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.
13. W. Stevens. TCP/IP Illustrated, Vol. 1, *The Protocols.* Addison-Wesley, Boston, 1994.
14. W. Stevens, “TCP congestion control,” RFC2581, Apr 1999.
15. K. Ramakrishnan, S. Floyd and D. Black, “The addition of explicit congestion notification (ECN) to IP,” RFC3168, September 2001.
16. M. Karaliopoulos, R. Tafazolli and B. Evans, “Proxy-assisted TCP maximum receive window control in split-TCP capable GEO satellite networks,” in *Proceedings of American Institute of Aeronautics and Astronautics*, May 9–12, 2004.