Adaptive Virtual Queue Random Early Detection in Satellite Networks

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

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 on queueing 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.

1. Introduction

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 aren't 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 msec.

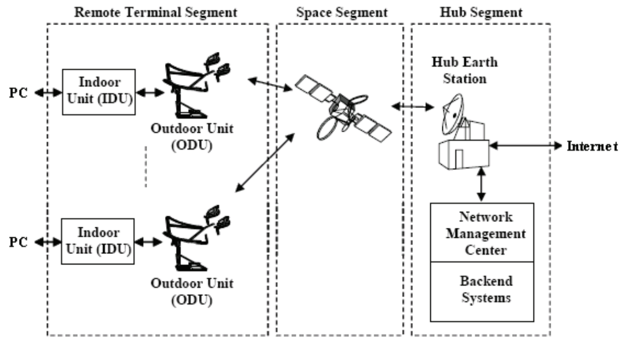

Figure 1. IPoS system architecture

Figure 1 illustrates the system architecture of a typical two-way IPoS system where half of the 600 msec RTT occurs between the *Hub Segment* and the *Space Segment*; the other half occurs between the *Remote Terminal Segment* and the *Space Segment*. The biggest problem with such high propagation delay is the TCP performance. One aspect of the problem is the TCP slow start [15] phase where it takes a long time (RTTx$\log_2$xSSTHRESH) 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 65,535x8/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) [22] has been practiced by most of IPoS 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 isn't enough because one has to account for queueing delays associated with congestion and 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) [20] 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 (see section 4) are not due to packet drops between gateway and senders.
2. ECN marking after PEP (*Transmit Q* in Figure 4) may seem to avoid retransmissions over satellite and fix the queueing instability problem discussed in section 4.1, 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 link.
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 isn't true for virtual-queue-based [21] AQMs such as Adaptive Virtual Queue (AVQ) [19], but they have the global synchronization (consecutive packet drops) problem.
4. Monitoring after PEP and marking packets before PEP is not a good idea because the high buffering between congested queue (transmit queue towards satellite) and receive queue may result in unstable queueing behavior. This phenomenon is discussed in more details in section 4.1.

A new virtual-queue-based AQM, Adaptive Virtual Queue Random Early Detection (AVQRED), is proposed to address the above concerns. The emulation results for RED [5], AVQ and AVQRED are compared to validate the proposed solution.

This paper is organized as follows: Section 2 provides an overview of PEP. Section 3 provides an overview of RED and AVQ. Section 4 defines the problems. Section 5 proposes a solution. Section 6 describes the emulation framework. Section 7 provides the emulation results. Finally, section 8 concludes the paper.

2. Overview of PEP

Performance Enhancing Proxy (PEP) [22] can be divided into two layers: application and transport layers. The application layer is responsible for locally acknowledging TCP packets and buffering up one+ round trip worth of data. The transport layer is responsible for transmitting and acknowledging data over spacelink. Some of its functionalities are SACK retransmissions, ACK reduction, traffic classification and compression.

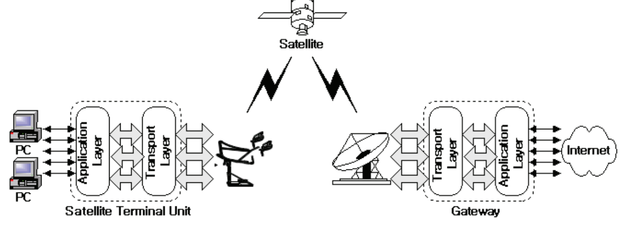

Figure 2. PEP flow diagram

Figure 2 illustrates the end-to-end PEP interactions in a two-way satellite network and Figure 3 is the ladder diagram of a simple HTTP transaction over PEP. Note that ACK(s) for Data 1 ~ Data 2 could be earlier.

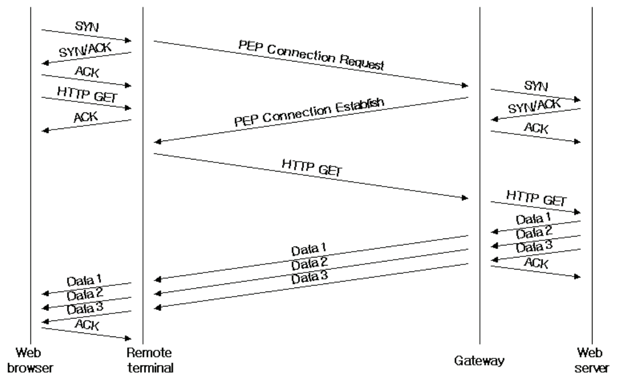

Figure 3. HTTP transaction over PEP

To better visualize PEP in a gateway, Figure 4 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 Figure 4, the congested queue is the *Transmit Q* as discussed in section 1.

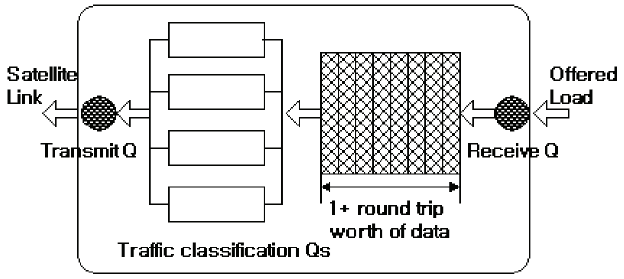

Figure 4. Gateway with PEP

It is common that PEP is also implemented in remote terminals to gain higher upload speed and keep the implementation symmetric, but congestion avoidance in the upload direction (from remote terminals to internet) is not discussed in this paper as it involves different congestion paths.

3. Overview of RED and AVQ

This section provides a high level overview of two well known AQM methods: Random Early Detection [5] and Adaptive Virtual Queue [19] which will be compared with AVQRED via emulations.

3.1. RED

The RED [5] 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 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 queueing 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) [21] 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) [19] 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 queueing 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 I. AQM Q and marking Q configuration

AQM method	Monitor queue	Marking queue
RED	Transmit queue	Receive queue
AVQ	Receive queue	Receive queue
AVQRED[1]	Receive queue	Receive queue

4.1. Asynchronous queueing 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

[1] Details of AVQRED are discussed in section 5.

in less queue occupancy until senders' congestion windows evolve causing oscillatory queueing behavior. Figure 5 illustrates how an asynchronous queueing 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 queueing behavior.

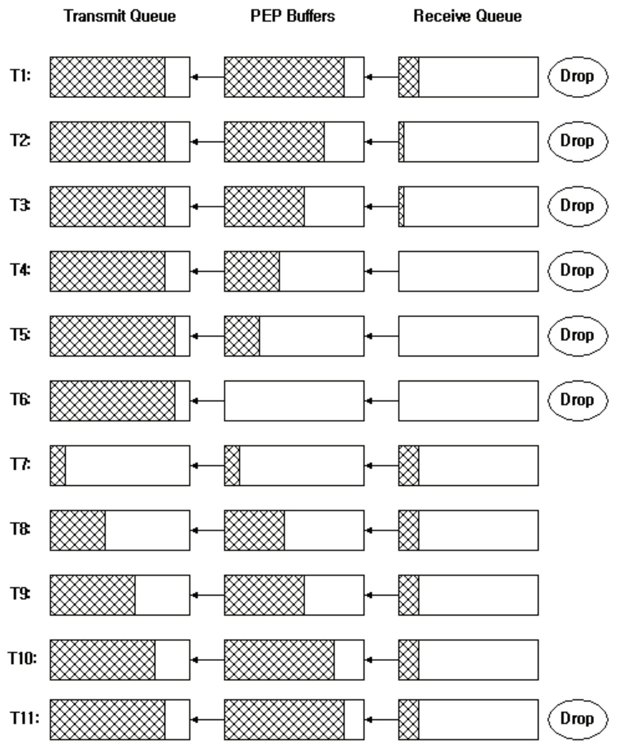

Figure 5. Asynchronous queueing behavior

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 global oscillatory link utilization amongst multiple TCP connections.

This problem is severer with RED due to its oscillatory queueing behavior described in section 4.1. 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 problems mentioned in section 4 (asynchronous queueing behavior and global synchronization).

```
Initialization:
  count <- -1
  last_measure <- curr_time
  prev_tx_bytes <- bytes_tx

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
      * have been 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 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
     /* b = number of bytes in current packet */
     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. 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. *alpha* 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 *alpha* can be any value.

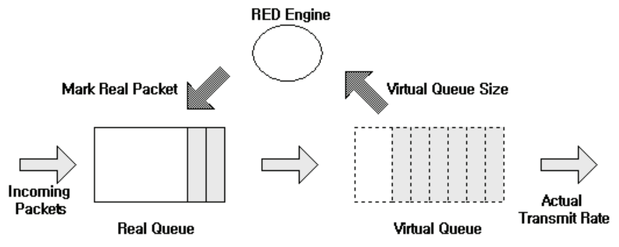

Figure 6. AVQRED flow diagram

As Figure 6 illustrates, RED monitors a virtual queue which is constructed based on the configured capacity and arrival packets, and marks packets in the real queue. This scheme essentially moves the transmit queue to the receive queue (before the PEP layer) in Figure 4 and produces better synchronization between the transmit queue and the traffic sources.

5.1. Asynchronous queueing behavior

AVQRED solves the asynchronous queueing behavior problem mentioned in section 4.1 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 in section 7 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 I. 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 Figure 4. 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 not relevant to the interest of this research. *Spirent®* was used to best emulate real life traffic characteristics.

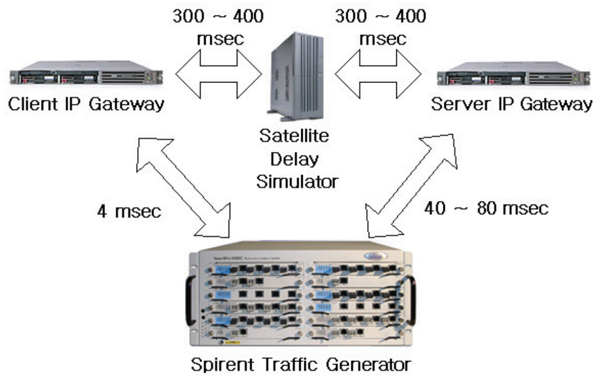

Figure 7. Emulation flow diagram

Figure 7 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 ~ 400 msec each way. 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 server IP Gateway is 600 ~ 800 msec, and the RTT between the server IP Gateway and the *Spirent* is 40~80 msec resulting in an end to end RTT of 644~884 msec. 400 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 seconds with 5 second 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 250Kbytes ~ 550Kbytes of data with 10 seconds user think time.
4. The maximum download speed of each TCP connection is 5Mbps.
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 comparisons:

1. Link utilization – The purpose of this metric is to make sure the proposed solution produces comparable link utilizations.
2. Queue size –The purpose of this metric is to compare queue size and queueing 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 are used throughout the emulations:

- 20 Mbps downlink bandwidth (hub 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 msec transmit rate regulator latency in the server IP Gateway.
- Target transmit queueing delay of 33 msec.
- Average packet size is 1500 bytes for the downlink direction.

For RED, there are four parameters to configure: min_{th}, max_{th}, max_p, and w_q. 60 and 120 are configured for min_{th} and max_{th} respectively. min_{th} is set slightly higher than 55 (= 20Mbps / 8 / 1500 x 0.033 from the above system parameters) to ensure full utilization of 20 Mbps bandwidth. min_{th} is set to at least twice min_{th} as [5] 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 II.

For AVQ, the target utilization, γ, is set to 100%, the buffer size, B, is set to 123,750 bytes where 123,750 = 82,500 (= 20Mbps / 8 x 0.033) + 82,500 / 2. Half of the buffer required for 20Mbps (82,5000 / 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; *alpha* 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* = 20Mbps.

7. Emulation results

Each of the following AQM setting was emulated for 20 minutes with the traffic described in section 6.

Table II. 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_cap*	*max_cap*
AVQRED	60	120	20 Mbps	20 Mbps

As discussed in 6.1, the link utilization, queue size and consecutive packet drop are measured during the emulations. The measurements were taken once every 100 msec and the following subsections discuss the results for each of the measurement metrics.

7.1. Link utilization

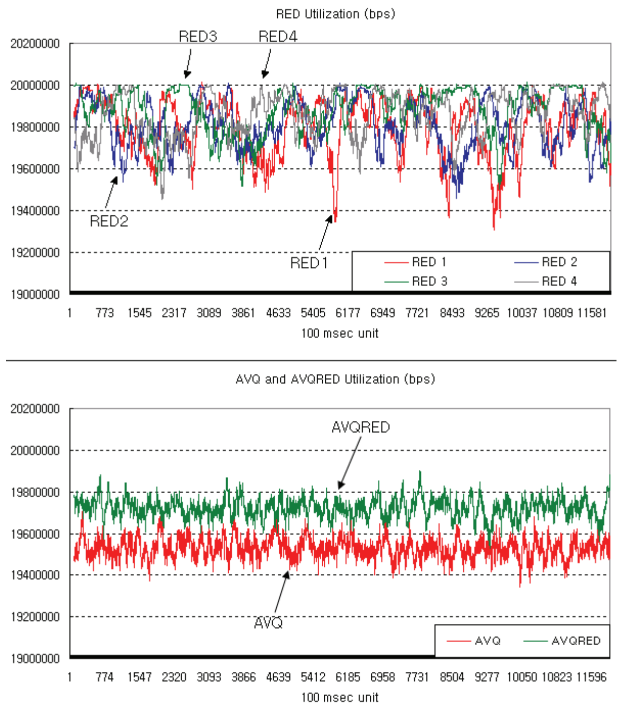

Figure 8. Link utilization comparison

As Figure 8 and Table III show, the utilization of AVQRED is comparable with the utilization of RED.

Although there is 0.5% loss in the mean utilization, there is 0.25% gain in the stability (the standard deviation).

Table III. Link utilization mean and stdev

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

Utilization loss and stability gain can be explained by the queueing behavior of RED and AVQRED which is discussed more in the next section. Basically, AVQRED maintains just enough data to fill up the 20Mbps pipe whereas RED's utilization is oscillatory and unstable due to its asynchronous queueing behavior which is discussed in section 4.1 and 7.2. 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 consistent with the fact that AVQ has more consecutive packet drops (discussed in section 7.3) because consecutive packet drops cause multiple senders to shrink their congestion windows synchronously resulting in lower link utilization.

7.2. Queue size

Figure 9 and Figure 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 queueing behavior. To provide a better visualization of this point, Figure 11 through Figure 14 magnify Figure 9 and Figure 10 between 50th and 150th seconds (500th ~ 1500th points according to the x axis' scale).

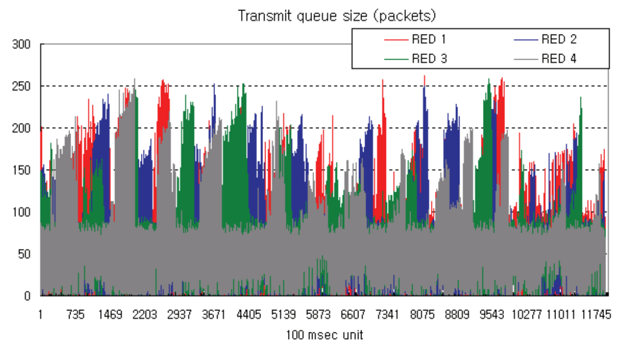

Figure 9. RED transmit queue

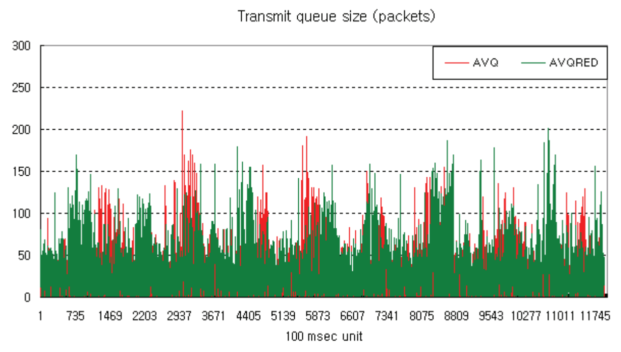

Figure 10. AVQ and AVQRED transmit queue

This oscillatory queueing behavior is the asynchronous queueing behavior described in section 4.1 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 queueing problem by both monitoring and dropping at the receive queue. As discussed in section 1, monitoring the receive queue with a real-queue-based AQM such as RED can't be done because the receive queue will never be congested when the bottleneck is the transmit queue by the spacelink bandwidth limitation.

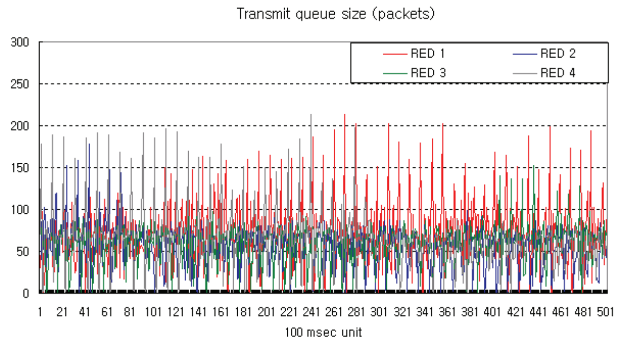

Figure 11. RED queue size (50th ~ 100th seconds)

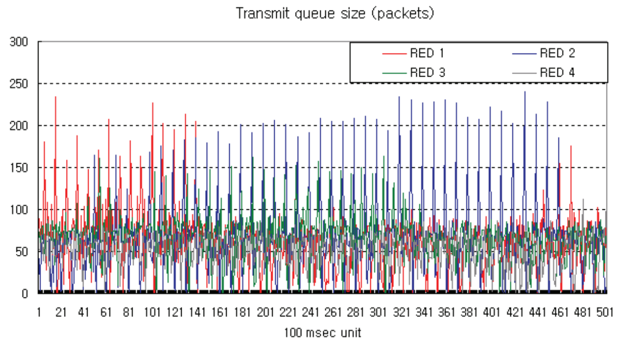

Figure 12. RED Q size (100th ~ 150th seconds)

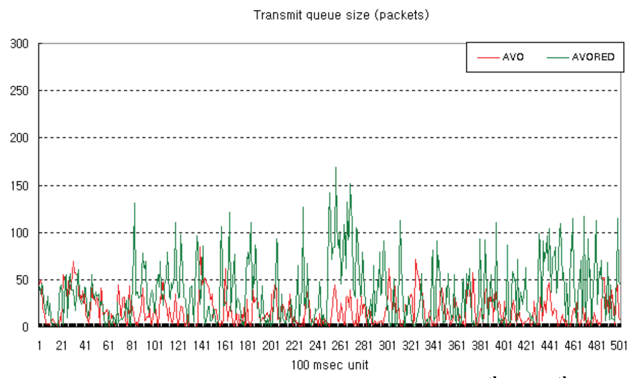

Figure 13. AVQ and AVQRED Q size (50th~100th sec)

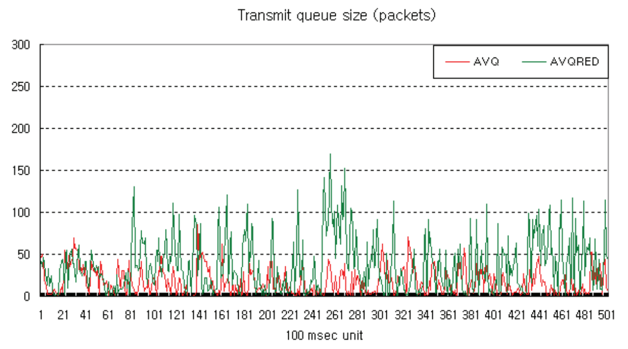

Figure 14. AVQ and AVQRED Q size (100th~150th sec)

7.3. Packet drop

Because 20 minutes worth of the packet drop histogram is too long to present, only the first 3000 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 IV.

Table IV. 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

Figure 15, Figure 16 and Figure 17 show packet drops for the first 3000 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 more clustered than they should be due to the queueing oscillation discussed in section 7.2.

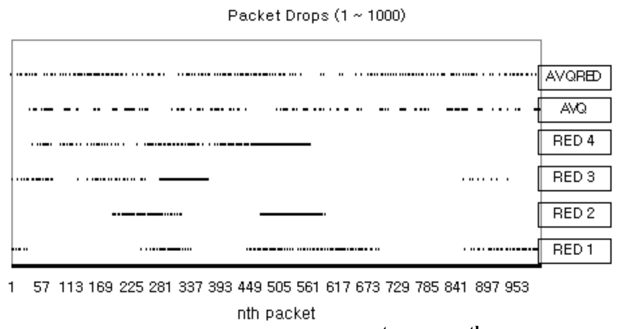

Figure 15. Packet drops for 1st ~ 1000th packets

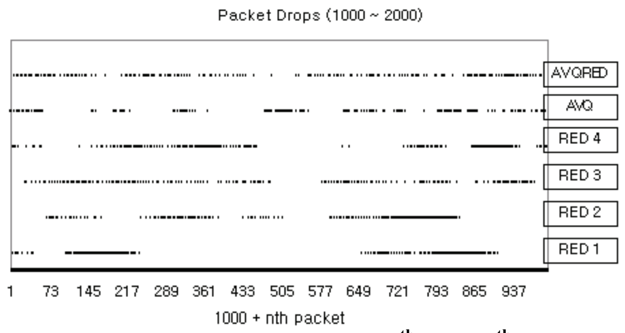

Figure 16. Packet drops for 1000th ~ 2000th packets

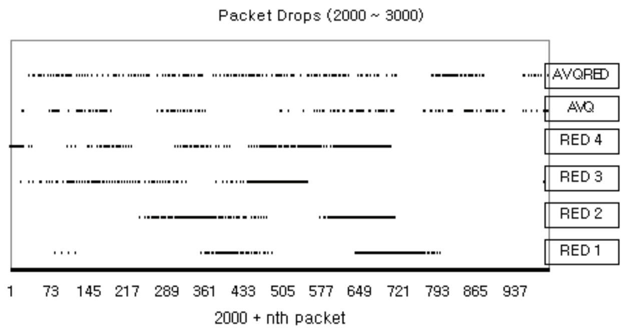

Figure 17. Packets drops for 2000th ~ 3000th packets

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. 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 such as queueing instability (asynchronous queueing behavior) and global synchronization (consecutive packet drops). The queueing instability problem is caused by real-queue-based AQM methods such as RED due to high PEP buffering and its inability to monitor the receive queue when the congested queue is the transmit queue. The global synchronization problem is caused by both real-queue-based and virtual-queue-based AQM methods due to queueing instability of real-queue-based AQMs and tail-drop nature of virtual-queue-based AQMs. To solve these two problems, a new AQM method, AVQRED, was developed.

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

The emulation results in section 7 confirmed that the problems are solved by AVQRED. AVQRED solves the queueing instability problem by synchronizing the congested (transmit) queue and the traffic sources. AVQRED solves the global synchronization problem by preserving the global synchronization avoidance of the RED algorithm.

In our future work, we plan to develop per-flow aware AVQRED to improve Quality of Service (QoS) for satellite networks. It is envisioned that implementing per-flow AVQRED does not require complex changes to the AVQRED algorithm because it only requires increasing the dimension of the algorithm by creating N different virtual queues.

9. References

[1] P. Lassila and J. Virtamo, "Modeling the dynamics of the RED algorithm," in *Proceedings of QofIS'00*, pp. 28-42, September 2000.

[2] C. Long, B. Zhao, X. Guan, and J. Yang, "The yellow active queue management algorithm," Computer Networks, vol. 47, no. 4, pp. 525–550, 2005.

[3] P. Kuusela, P. Lassila, J. Virtamo and P. Key, "Modeling RED with idealized TCP sources," http://research.microsoft.com/~peterkey/Papers/ifipredtcp.pdf, 2001.

[4] 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.

[5] 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.

[6] 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.

[7] S. Floyd et al., "Discussions on setting parameters," http://www.aciri.org/floyd/REDparameters.txt, November 1998.

[8] 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.

[9] 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.

[10] W. Feng, D. Dilip D. Kandlur, Debanjan Saha, and Kang G Shin, "A self-configuring RED gateway," in *Proceedings of IEEE/INFOCOM*, 1999.

[11] 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.

[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] T. J. Ott, T. V. Lakshman, and L. H. Wong, "SRED: stabilized RED," in *Proceedings of IEEE INFOCOM*, March 1999.

[14] W. Stevens. *TCP/IP Illustrated, Vol. 1 The Protocols.* Addison-Wesley, 1994.

[15] W. Stevens, "TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms," RFC2001, Jan 1997.

[16] W. Stevens, "TCP congestion control," RFC2581, Apr 1999.

[17] M. May, T. Bonald, and J. Bolot, "Analytic evaluation of RED performance," in *Proceedings of IEEE IFOCOM*, March 2000.

[18] 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.

[19] 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.

[20] K. K. Ramakrishnan and S. Floyd, "A proposal to add explicit congestion notification (ECN) to IP," RFC 2481, Jan. 1999.

[21] 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.

[22] J. Border, M. Kojo, J. Griner, G. Montenegro and Z. Shelby, "Performance enhancing proxies intended to mitigate link-related degradations," RFC3135, Jun 2001.

[23] K. Ramakrishnan, S. Floyd and D. Black, "The addition of explicit congestion notification (ECN) to IP," RFC3168, Sep 2001.

[24] R. Pan, B. Prabhakar, and K. Psounis, "CHOKe: A stateless AQM scheme for approximating fair bandwidth allocation," in *Proceedings of IEEE INFOCOM*, March 2000.

[25] H. Lim, K.-J. Park, E.-C. Park, and C.-H. Choi, "Virtual rate control algorithm for active queue management in TCP networks," *IEE Electronics Letters* pp. 873-874, 2002.