ABSTRACT

Title of Document: CONGESTION CONTROL IN SATELLITE NETWORKS.

Do Jun Byun, Doctor of Philosophy, 2007

Directed By: Professor John S. Baras,
Department of Computer Science

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 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 using a realistic emulation environment and a mathematical model. Three problems that were discovered during the research are discussed in this dissertation.

The first problem is oscillatory queueing, 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. Existing AQMs that can solve this problem have tail drop queueing that results in consecutive packet drops (global synchronization). A new AQM method called Adaptive Virtual Queue Random Early Detection (AVQRED) is proposed to solve this problem.

The second problem is unfair bandwidth sharing caused by inaccurate measurements of per-flow bandwidth usage. AVQRED is enhanced to accurately measure per-flow bandwidth usage to solve this problem without adding much complexity to the algorithm.

The third problem is queueing instability caused by buffer flow control where TCP receive windows are adjusted to flow control traffic senders instead of dropping received packets during congestion. Although buffer flow control is quite attractive to satellite networks, queueing becomes unstable because accepting packets instead of dropping them aggravates the congestion level. Furthermore, buffer flow control has abrupt reductions in the TCP receive window size due to high PEP buffering causing more instability. AVQRED with packet drop is proposed to solve this problem.

Networks with scarce bandwidth and high propagation delays can not afford to have an unstable AQM. In this research, three problems that are caused by existing AQMs are identified and a new AQM is proposed to solve the problems. This research can be used by the satellite industry to improve gateway performances and provide better end-user experiences.

CONGESTION CONTROL IN SATELLITE NETWORKS

By

Do Jun Byun

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2007

Advisory Committee:
Professor John S. Baras, Chair
Professor Raymond Miller
Professor Yiannis Aloimonos
Professor David Mount
Professor Tobias von Petersdorff

Dedication

To God;

To my father, Khil Nam Byun, and my mother, Jung Hie Seu;

To my brother, Sok Jun Byun, and his family and

To my future wife and children.

Acknowledgements

Pursuing a Ph.D. degree while I am working as a full-time software engineer at Hughes Network Systems has been an unforgettable long journey in my life. I would like to take this opportunity to thank those who helped me through this journey.

First, I would like to give the sincerest thanks to my advisor, Professor John S. Baras. Without his encouragement, guidance and advises, I would have not been able to find the problems and the solutions that composed my research. Thank you for enduring the journey with me since 2002.

I would like to thank Professor Raymond Miller, Professor Yiannis Aloimonos, Professor David Mount and Professor Tobias von Petersdorff for kindly serving on the committee and reviewing the dissertation.

I would like to thank Mr. Rod Ragland, Mr. George Choquette and Mr. Jeff Biber for their support as my supervisors in HNS. Special thanks to Mr. Rod Ragland for allowing me to use the Spirent load test-bed during nights and weekends for my research.

I would like to thank Mr.Young Chae Kim's heartful prayers for me to successfully finish my Ph.D. degree. I would also like to thank his family and the members of One-Heart including Dae Min Kim, Jae Min Kim, Yu Sun Kim, Jae Wook Kim, Jumi Yoon, Jong Woo Choi and Woo Hyuk Choi for their spiritual support throughout the journey.

I would like to thank Dr. Baras' research coordinator, Ms. Kim Edwards, for her dedicated coordination to arrange meetings with Dr. Baras and to organize paperwork.

As I live and work far from College Park, her precise meeting arrangements have saved me tremendous amount of time.

I would like to thank my friend, Seiwon Hong, for amusing me time to time during this dry journey. I would also like to thank my friend, Rohit Tripathi, for proof-reading my dissertation, and Dong Seong Kim for helping me to distribute the thesis to the committee members.

Finally, I would like to express my deepest gratitude to God and my parents for their love and support.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1. Internet over Satellite

Internet Protocol (IP) over Satellite (IPoS) has been commercially available for the last few decades. Due to its high mobility, IPoS has been attractive to areas where terrestrial services are not available as well as enterprises with geographically dispersed 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 1.1 illustrates the system architecture of a typical two-way IPoS system where half of the 600 msec 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 $65{,}535 \times 8/RTT$ 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 behind 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 queueing delays associated with congestion and inroute (uplink) bandwidth allocations.

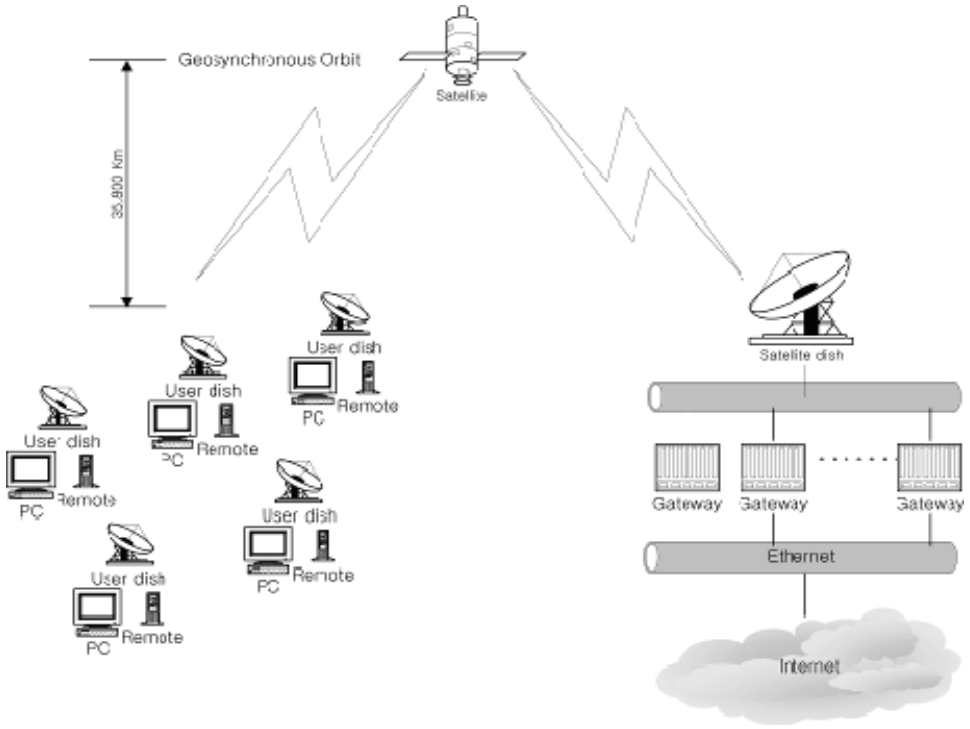

Figure 1.1: Typical IPoS system

1.2. Overview of AQM

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 Notifica-

tion (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. In fact, not dropping packets causes more problems as discovered in Chapter 6.
2. ECN marking after PEP (transmit queue in Figure 1.2) may seem to avoid retransmissions over satellite and fix the queueing instability problem discussed later, but it is too late to enforce ECN bits when data are already acknowledged without ECN bits by PEP.

In satellite networks, gateways can also indicate congestion to traffic senders by advertising smaller TCP receive windows because PEP in the gateways replaces the actual TCP hosts at the remote terminal side. This method will be referred to as buffer flow control throughout this dissertation.

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 dropping 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].

AQMs with both drop and buffer flow control marking methods are examined in this thesis. The problems and the solutions are revealed and validated through a realistic emulation environment constructed with the actual gateway software used in *Hughes Network Systems' HughesNet*® networks. Intuitive analysis of the problems and the solutions is given via mathematical models. The emulation results and the MATLAB results are compared to validate the analysis.

1.3. Statement of the Problems

The main objectives we are trying to achieve are to avoid retransmissions over satellite, maintain queueing stability and avoid global synchronization (consecutive packet drops) while preserving high link utilization.

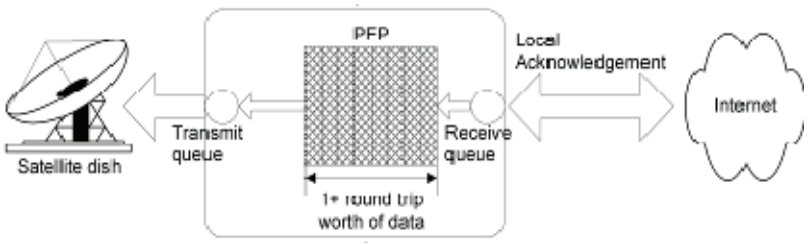

Figure 1.2: Gateway with PEP in satellite networks

1.3.1. Asynchronous Queueing and Global Synchronization

Given that the congested queue is the transmit queue (in Figure 1.2), when AQMs with packet drop marking is used in satellite networks, dropping packets after PEP is not a good idea because it involves retransmissions over satellite. Because the congested queue is the transmit queue, monitoring must occur after the PEP unless the AQM is virtual-queue-based. Because there is significant delay due to PEP buffering between the AQM monitoring queue (transmit queue) and the marking queue (receive queue), the monitoring queue (transmit queue) will have an oscillatory queueing behavior. This problem will be referred as asynchronous queueing throughout this dissertation.

The asynchronous queueing is fixed by virtual-queue-based AQMs because the monitoring queue is a virtual queue which can be placed anywhere. So, in satellite networks, the monitoring queue can be placed at the receive queue (in Figure 1.2) to avoid the asynchronous queueing. However, virtual-queue-based AQMs have the global synchronization problem (consecutive packet drops) due to their tail-drop nature which degrades TCP link utilization.

1.3.2. Fair Bandwidth Sharing

Because satellite networks have far scarcer bandwidth than terrestrial networks, precise fair bandwidth sharing is demanded. Several representative per-flow sensitive AQMs are examined and none of them is found to compute a precise enough fair bandwidth share for each flow. The solution to the previous problem (asynchronous queueing and global synchronization) is enhanced to provide a far more precise fair bandwidth share to each of the TCP and non-TCP flows.

1.3.3. Buffer Flow Control Instability

When AQMs with buffer flow control marking method are applied to satellite networks, the congested queue becomes unstable due to the following two phenomena:

1. When a received packet is marked due to congestion, it is still accepted to the congested queue aggravating the congestion.
2. When the gateway is congested, any new TCP connections cause initial bursts until their TCP receive windows are adjusted to small enough windows and the adjustments can take a long time due to big PEP buffers.

1.4. Contributions

First, we have found the following unique properties of satellite networks:

1. Congestion arises due to the satellite link capacity not the processing capacity.
2. There is high buffering between the traffic sources and the congested queue due to PEP.
3. There is a limitation on where to drop packets because dropping packets after PEP involves retransmissions over satellite.

Second, we have found the following problems when applying existing AQMs to satellite networks and provided the solutions for them:

1. Real-queue-based AQMs have the asynchronous queueing problem due to its inability to synchronize with traffic senders, and virtual-queue-based AQMs have the global synchronization problem due to their tail-drop nature.

2. Existing per-flow sensitive AQMs do not have precise enough fair bandwidth distributions.
3. AQMs with buffer flow control marking have a queueing stability problem because they accept packets instead of dropping them during congestion and new connections have bursty startups.

Third, we have constructed an emulation environment with the real gateway software used in *Hughes Network Systems' HughesNet*® networks and a traffic generator called *Spirent* to produce far more realistic traffic and performance measurements than simulations. To provide intuitive illustrations of the first and the third problems, we have constructed mathematical models which agreed with the emulations results.

1.5. Dissertation Organization

The arrangement of this dissertation is as follows: Chapter 2 provides the overview of PEP and existing AQMs, and justifies the selection of the AQMs that are compared with the solutions. Chapter 3 provides the emulation framework and parameter settings. Chapter 4 discusses the first problem (asynchronous queueing and global synchronization) and its solution. The emulation results and the MATLAB results are provided. Chapter 5 discusses the second problem (fair bandwidth sharing) and its solution. The emulation results are provided. Chapter 6 discusses the third problem (buffer flow control instability) and its solution. The emulation results and the MATLAB results are provided. Finally, chapter 7 concludes this dissertation.

Chapter 2

Background and Related Work

2.1. Overview of PEP

This section illustrates PEP and provides the limitations on active queue management that it imposes in satellite networks. 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.

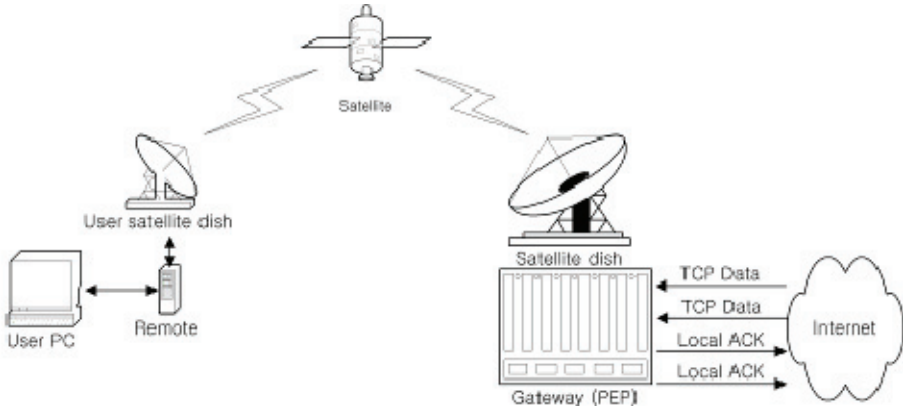

Figure 2.1: PEP flows

Figure 2.1 illustrates the end-to-end PEP flows in a two-way satellite network, and Figure 2.2 is the ladder diagram of a simple HTTP transaction over PEP. Note that ACK(s) for Data 1 ~ Data 2 could be earlier. To better visualize PEP in a gateway, Figure 1.2 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 1.2, the congested queue is the transmit queue.

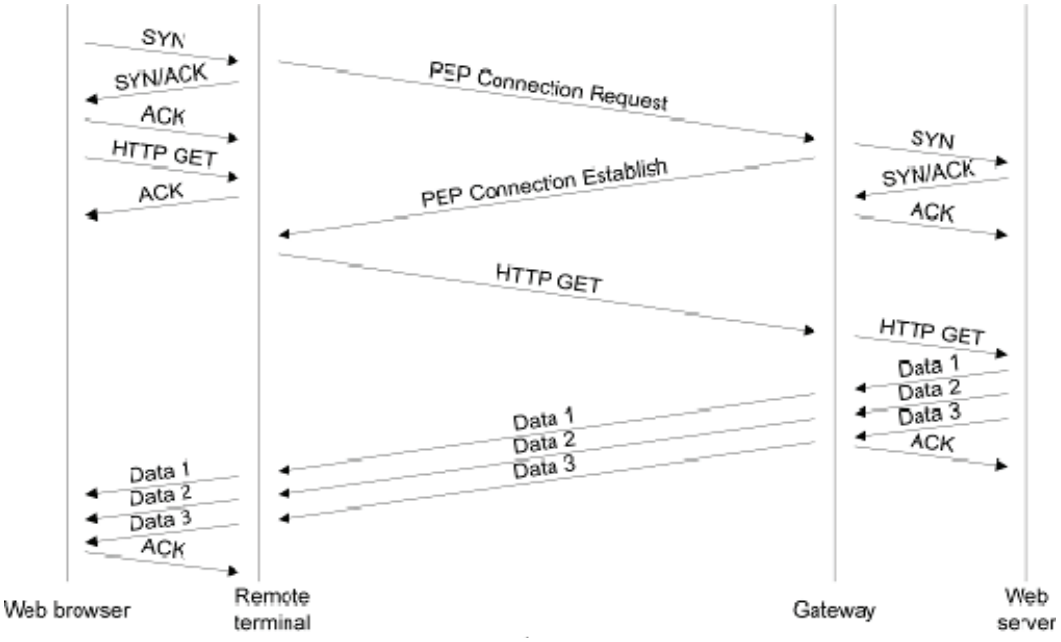

Figure 2.2: HTTP transaction over PEP

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

2.2. Overview AQM Methods

This section provides an overview of eight AQM methods and justifies the selections of the AQM methods that are compared with our solutions, AVQRED [6][7][8] and PFAVQRED [9]. The eight AQM methods are Random Early Detection [10], Stabilized Random Early Drop [11], Yellow [12], CHOKe [13], VRC [14], Adaptive Virtual Queue [5], VQ-RED [15] and REDFC [16]. In summary, RED and AVQ are se-

lected for the asynchronous queueing and global synchronization problems. RED, AVQ and SRED are selected for the fair bandwidth sharing problem. And RED and AVQRED are selected for the buffer flow control instability problem.

2.2.1. RED

The RED [10] 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 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

RED was selected for the comparisons to show the queueing instability problem of real-queue-based AQMs. Although choosing the best real-queue-based AQM for the comparisons is not within the scope of this research, one of our previous studies [8] revealed that RED performed the best amongst RED, BLUE [17] and PI [18]. Therefore, RED was selected to represent the real-queue-based AQM class.

2.2.2. SRED

Stabilized Random Early Drop (SRED) [11] is a real-queue-based AQM which maintains a list of M recently seen flows called *zombies*. The intuition is to identify misbehaving flows by randomly choosing a flow from the *zombie* list and comparing it with the received packet. When the packet matches any of the flows, the algorithm penalizes the packet by applying a higher drop probability hoping that misbehaving flows will have more hits.

$$Hit(t) = \begin{cases} 0 & \text{if no hit (match not found in zombies)} \\ 1 & \text{if hit (match found in zombies)} \end{cases}$$

$$P(t) = (1-\alpha)\cdot P(t-1) + \alpha \cdot Hit(t), \qquad 0<\alpha<1$$

$$p_{sred}(q) = \begin{cases} p_{\max} & \text{if } \frac{1}{3}B \le q < B \\ \frac{1}{4}\times p_{\max} & \text{if } \frac{1}{6}B \le q < \frac{1}{3}B \\ 0 & \text{if } 0 \le q < \frac{1}{6}B \end{cases}$$

$$p_{new\_sred}(q) = p_{red}(q)$$

$$p_{zap} = p_{sred} \times \min\left(1, \frac{1}{(256\times P(t))^2}\right) \times \left(1 + \frac{Hit(t)}{P(t)}\right)$$

$$p_{new\_zap} = p_{new\_sred} \times \left(1 + \frac{Hit(t)}{P(t)}\right)$$

p_{zap} is the drop probability proposed by Ott et al [11] and p_{new_zap} is our modified drop probability. The original p_{sred} is also slightly modified and named as p_{new_sred} to produce smoother transitions of the drop probability where p_{red} is the RED drop probability, p_a, described in section 2.2.1. Because the second term of the right hand side of p_{zap}, $\min\left(1, \frac{1}{(256 \times P(t))^2}\right)$, can only lower the drop probability and does not help penalizing misbehaving flows, it was removed (or the 256 part was changed to 1) in the modified SRED drop probability, p_{new_zap}. Furthermore, the authors of SRED stated that the choice of 256 in p_{zap} was arbitrary, and our preliminary emulation results (which are not shown) with p_{zap} produced worse results than p_{new_zap}. Therefore, p_{new_zap} was used throughout the emulations.

SRED was selected to compare the fairness metric as it is per-flow sensitive.

2.2.3. Yellow

Yellow [12] is a real-queue-based AQM method which is similar to BLUE [17]. The main idea is to increase or decrease the packet marking probability using the load factor which is the ratio between the offered load and the available virtual capacity. The available virtual capacity, c', is calculated by the following equations:

$$f(q) = \begin{cases} \max\left(ODLF, \frac{\gamma \cdot \alpha \cdot q_{ref}}{(\alpha - 1) \cdot q + q_{ref}}\right) & \text{for } q > q_{ref} \\ \frac{\gamma \cdot \beta \cdot q_{ref}}{(\beta - 1) \cdot q + q_{ref}} & \text{for } 0 \le q \le q_{ref} \end{cases}$$

$$c'(q) = f(q) \cdot c$$

$$z = \frac{\text{Offered load}}{c'(q)}$$

Variables:

- ODLF (Queue drain limit factor) = Minimum utilization factor.
- q_{ref} = Target queue length.
- α = Parameter that controls the reaction speed when congestion is detected ($q > q_{ref}$). The bigger this value is, the faster and less stable the algorithm becomes.
- β = Parameter that controls the reaction speed when congestion is not detected ($0 \le q \le q_{ref}$).
- γ = Target utilization factor that ranges up to 1.0.

```
if (z >= 1 + delta)
   if ((now - last_update) > freeze_time)
      pmark = pmark + d*z
      last_update = now
if (z < 1)
   if ((now - last_update) > freeze_time)
      pmark = pmark - d/z
      last_update = now
```

Yellow algorithm

Yellow was not selected for the comparisons because it is also a real-queue-based AQM which is not any better than RED when applied to satellite networks due to the asynchronous queueing behavior described in section 4.1. In fact, some emulation results revealed that its `freeze_time` adds more delays between the AQM queue and the traffic senders aggravating the asynchronous queueing effect.

2.2.4. CHOKe

CHOKe [13] is a real-queue-based AQM which is very similar to RED except it penalizes a flow (which is the flow of a randomly chosen packet from the congested queue) if the received packet belongs to the same flow. If the received packet belongs to the flow, both the received packet and the packet in the congested queue are

dropped. Otherwise, the RED drop probability is applied to the received packet. The objective of CHOKe is to find misbehaving flows by comparing the received packet with a randomly chosen packet from the congested queue and penalize them.

```
For each arrival packet:
  if (avg_q_size < min_th)
    admit pkt_rx
  else if (min_th <= avg_q_size <= max_th)
    pick pkt_rand from the congested Q
    if (pkt_rx.conn == pkt_rand.conn)
      drop pkt_rx and pkt_rand
    else if (avq_q_size <= max_th)
      drop pkt_rx with RED drop prob, p_a
    else
      drop pkt_rx
```

Choke algorithm

There are two problems with CHOKe when applying it to satellite networks. The first problem is that we can not drop randomly chosen packets in the congested queue because dropping them causes retransmissions over satellite. The second problem is that dropping PEP packets can penalize well-behaving TCP flows because one PEP transport layer flow can contain multiple TCP flows. Therefore, CHOKe was not selected for our comparisons.

2.2.5. VRC

Virtual Rate Control (VRC) [14] is a real-queue-based AQM that maintains a virtual target rate and computes the marking probability which is derived from the difference between the input rate and the virtual target rate. The reason for deriving the marking probability using the virtual target rate instead of the actual target rate is that the input rate does not converge to the actual target rate. The reason why it doesn't converge is

that the input rate exceeds the target rate when the marking probability is directly derived from the difference between them.

$$p(t) = [\alpha(x(t) - x_r(t))]^+, \qquad \alpha > 0, \ [\cdot]^+ \equiv \max(\min(\cdot, 1), 0)$$

where *p(t)* is the drop probability, *x(t)* is the input rate, and $x_r(t)$ is the target rate. *p(t)* starts dropping packets only when the input rate is greater than the target rate and the result is queueing instability and overflows. Therefore, the virtual target rate is used in the actual VRC algorithm. The virtual target rate is slightly lower than the actual target rate and adjusted according to the difference between the input rate and the actual target rate.

```
For every Ts sampling interval
  /* Calculate the target rate xr */
  xr = C + γ * (qr − q)

  /* calculate the virtual target rate xv */
  Δxv = Δxv + β * (x − xr)
  xv = xr − Δxv

  /* calculate the marking probability */
  p = α * (x − xv)
```

VRC algorithm

VRC was not selected for our comparisons because there is no need to estimate the target rate when the output (spacelink) capacity is well-known. Furthermore, start marking packets before the full utilization (by introducing a virtual target rate) would result in lower link utilization, and the RED algorithm has more intuitive global synchronization avoidance.

2.2.6. AVQ

Gibbens-Kelly Virtual Queue (GKVQ) [4] is a virtual-queue-based AQM which 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) [5] 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)

   s = t  /* Update last packet arrival time */

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

AVQ was selected for our comparisons because the algorithm builds a virtual queue which can solve the asynchronous queueing problem (described in section 4.1).

It also provides the target utilization (desired link utilization, γ) which fits well for satellite networks where the spacelink capacity is known and static.

2.2.7. VQ-RED

VQ-RED [15] is a virtual-queue-based AQM that aims at improving the fairness of RED by creating N virtual queues and applying a RED-like algorithm to each virtual queue.

```
For every period:
  /* estimate average rate */
  rate = rate*0.1 + avg_rate_per_conn*0.9

For each inactivity period:
 Delete inactive vq

For each arrival packet (pkt):
  find (or create) flow i that pkt belongs to
  if (vq[i].length > max_th)
    drop
  else if (vq[i].length < min_th)
    vq[i].length += pkt.lenght
  else
    /* compute RED-like marking probability */
    p = (vq[i].length – min_th)/(max_th - min_th)
    if (!dropped)
      vq[i].length += pkt.length

  vq[i].length -= (now – vq[i].last) * rate
  vq[i].length = max(vq[i].length, 0.0)
  vq[i].last = now
```

VQ-RED algorithm

The main difference between our solutions (AVQRED and PFAVQRED) and VQ-RED is that VQ-RED does not have the target utilization and it reacts to the congestion level of each virtual queue, whereas AVQRED and PFAVQRED control the congestion level of the aggregate traffic via imposing the target utilization on a single virtual queue. This difference makes VQ-RED very sensitive to the accuracy level of

the number of flows estimate because an accurate per-flow rate relies on an accurate number of flows estimate. Furthermore, VQ-RED can have unnecessary packet drops. For example, if the link capacity is 1 Mbps and there are only two flows with 100 Kbps and 50 Kbps each, then the average rate per flow will be 75 Kbps. The virtual queue for the flow with 100 Kbps will grow and eventually cause packet drops although the aggregate utilization is well under the capacity. Most importantly, configuring min_{th} and max_{th} is not intuitive because they depend on the number of flows. The following equation depicts this point:

$$pkts = \min_{th} \times n + (1 - p) \times (\max_{th} - \min_{th}) \times n$$

where *pkts* is the total number of packets accepted to the real queue, n is the total number flows, $min_{th} \sim max_{th}$ is the congestion region for each virtual queue, and p is the overall marking probability. This equation implies that having a static congestion region ($min_{th} \sim max_{th}$) for each virtual queue makes the real queue unstable when n varies.

VQ-RED was not selected for our comparisons due to the problems mentioned above.

2.2.8. REDFC

Random Early Detection Flow Control (REDFC) [16] is a real-queue-based AQM designed for satellite networks. The main idea is to apply the RED algorithm to the transmission rate. It controls the transmission rate by using the RED marking probability, p_a.

```
for each arrival packet
  calculate the avg_q_size
  if avg_q_size < max_th
```

```
  Transmit the packet
else if min_th<=avg_q_size<=max_th
  calculate RED mark probability, p_a
  with p_a:
    transmit the packet
if the packet is not transmitted
  wait for time T, then try again
```

REDFC algorithm

Although REDFC can avoid packet drops and retransmissions over satellite, it was not selected for our comparisons due to the following problems:

- There is no nice retransmissions timeout, *T*, that can avoid duplicate retransmissions from PEP or TCP. When there are many duplicate retransmissions, the transmit queue can be filled with retransmitted packets and the actual end-user throughputs can degrade significantly while the gateway utilization is high.
- There can be high buffering in the remote terminals because delaying the transmissions causes holes in a sequence of PEP or TCP transmissions which cause the resequencing queue to grow in the remote terminals.
- If the remote terminals do not maintain resequencing queue, delaying transmissions can trigger unnecessary retransmission logic (such as transmitting SACKs) in the remote terminals.

2.3. Overview of AQM Marking Methods

Two types of marking method are used with the AQM methods selected: packet drop marking and buffer flow control marking [19]. Marking packets with ECN bits is not considered due to the reasons stated in section 1.2. Therefore, when a packet is

marked, either the PEP buffers will shrink to advertise smaller TCP windows to the sender or the packet will be dropped to trigger congestion avoidance at the sender.

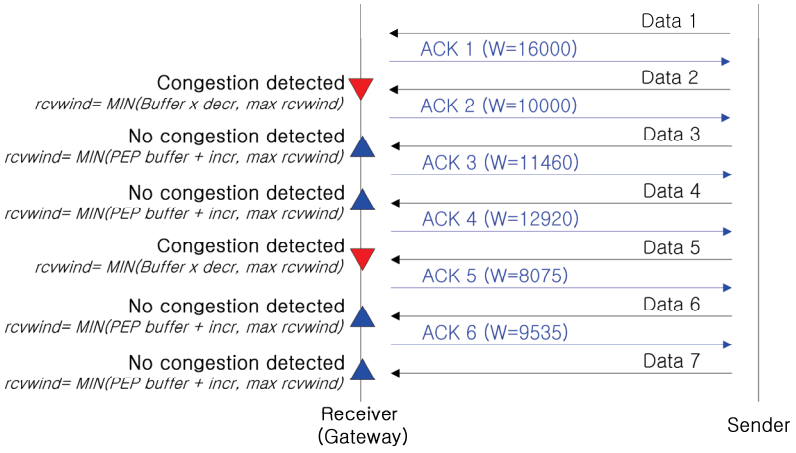

Figure 2.3: Buffer flow control packet marking

Figure 2.3 illustrates how marking (or unmarking) packets with buffer flow control causes the sender to slow down (or speed up) the transmissions. When a packet is marked due to congestion, the PEP buffers are reduced by a configurable amount and the reduction ends up reducing the TCP receive window. When a packet is not marked, the PEP buffers are increased by a configurable amount and the TCP receive window is re-opened gradually allowing the sender to send more data. To avoid oscillatory behavior of buffer adjustments, a configurable freeze time (usually an average RTT) is applied to each increment of the buffers. The intuition behind decrementing (or incrementing) the PEP buffers is to have the buffer adjustments indirectly cause the senders to behave as if they are in congestion avoidance (or recovery) phase described in RFC 2001 [1], RFC 2581 [20] and [21]. The following equation summarizes the buffer flow control mechanism where W is the advertised TCP receive window, B is the available PEP buffers and *MaxRcvWind* is the maximum TCP receive window.

$$W_i = \begin{cases} MIN(B_{i-1} \times decr, MaxRcvWind) & \text{if congestion is detected} \\ MIN(B_{i-1} + incr, MaxRcvWind) & \text{if congestion is not detected} \end{cases} \quad (2.1)$$

decr and *incr* are configurable parameters that control the adjustments of the PEP buffers. Our emulation results showed that reducing the buffer limit by 1/2 upon detecting congestion results in very oscillatory queueing behavior because 1/2 of the PEP buffers is quite a lot (unlike 1/2 of the maximum TCP cwnd, 1/2 ×64 KB). For our emulations, we explored several values of *decr* to find the best setting(s).

Figure 2.4 illustrates how dropping (or not dropping) packets causes the TCP sender to slow down (or speed up) the transmissions. The assumption is that the senders honor the congestion control method described in RFC 2001 [1], RFC 2581 [20] and [21]. When a packet is dropped due to congestion, three duplicate ACKs from the receiver (gateway) cause a fast retransmission from the sender and a reduction in the congestion window by 1/2. Whenever an ACK for new data is received, the congestion window is incremented by one segment (or by a configurable amount).

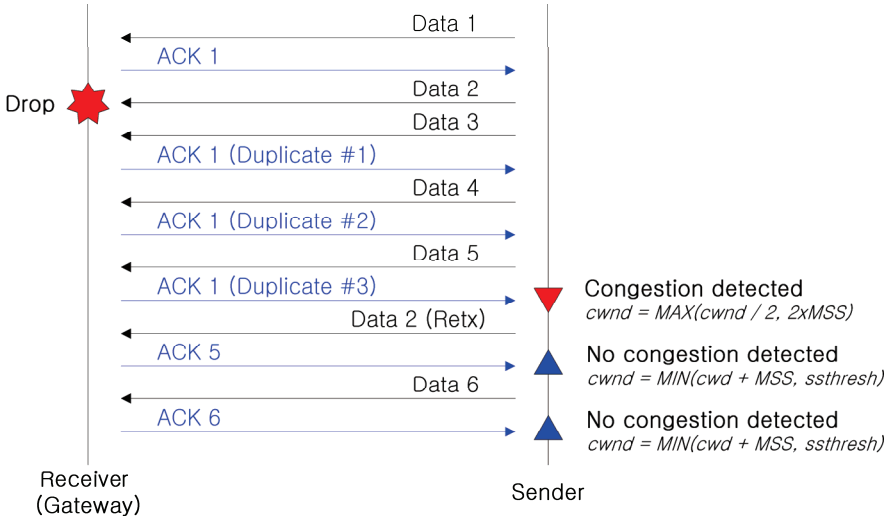

Figure 2.4: Packet drop marking and TCP congestion avoidance

Please note that Figure 2.4 is just one scenario that causes a retransmission. Other scenarios such as timeouts and SACKs are not described in this dissertation as the main idea of how senders react to congestion remains the same.

2.4. Summary

Table 2.1 summarizes the existing AQM methods we discussed in section 2.2. Three of them are selected for the comparisons with our solutions and the reasons for not selecting the others are provided in the corresponding subsection. The fairness metric will also be compared with the optimal share of the bandwidth to better visualize the fairness of each algorithm.

Table 2.1: Existing AQM method summary

AQM method	Queue type	Flow aware	Selected
RED	Real queue	No	Yes
SRED	Real queue	Yes	Yes
Yellow	Real queue	No	No
CHOKe	Real queue	Yes	No
VRC	Real queue	No	No
AVQ	Virtual queue	No	Yes
VQ-RED	Virtual queue	Yes	No
REDFC	Real queue	No	No

For the asynchronous queueing, global synchronization and fair bandwidth sharing problems, the above selected AQMs are used with packet drop marking method. For the buffer flow control instability problem, only RED and AVQ will be used with both packet drop and buffer flow control marking methods. The reason why SRED was not used for the buffer flow control problem is that the problem is not related to the fairness of bandwidth usage.

Chapter 3

Emulation Framework

3.1. Components and Traffic

The actual gateway software, IP Gateway, from *Hughes Network Systems' HughesNet*® networks was used to evaluate the AQM methods. The three AQM methods were implemented according to Table 2.1. 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 1.2. 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 3.1 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[1] 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 (500 for the fair bandwidth sharing problem) HTTP connections are established.
2. When a connection is closed, a new connection is created to fill the gap to maintain 400 (500 for the fair bandwidth sharing problem) HTTP connections.
3. Each web page contains 250 Kbytes ~ 550 Kbytes of data with 10 seconds 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 (25 for the fair bandwidth sharing problem) connections per second (approximately 5 % of the total population).

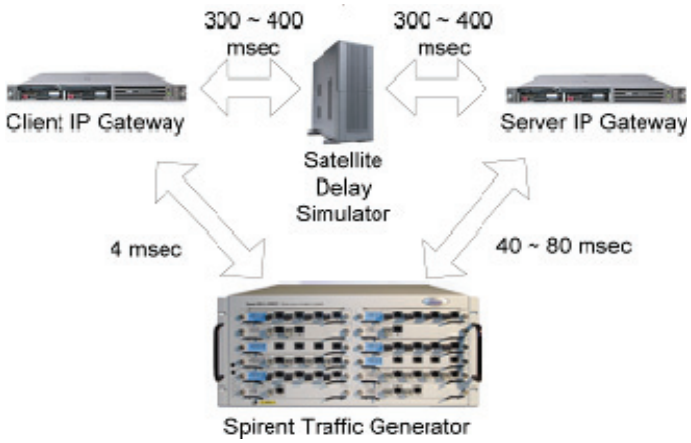

Figure 3.1: Emulation flow

[1] 500 connections were emulated for the fair bandwidth sharing problem.

To introduce misbehaving flows for the fair bandwidth sharing problem, three UDP flows were also generated with 80 Kbps, 160 Kbps and 320 Kbps constant bit rates (CBRs).

3.2. Evaluation Methodologies

The following performance metrics were used for validation:

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.
4. Per-flow throughput – The purpose of this metric is to compare fair sharing of the bandwidth. This metric is measured only for the fair bandwidth sharing problem.

The measurements were taken after all 400 (500 for the fair bandwidth sharing problem) HTTP connections are established to best emulate a loaded scenario.

3.3. 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 gateway 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 1400 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 59 (= 20 Mbps / 8 / 1400 $\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 [10] 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 3.1.

For SRED, there are three more parameters than RED. The low-pass filter, α, for estimating the hit rate, P(t), is set to 0.1 and setting it to a different value would not make a big difference in results because the dynamics of our traffic model are not drastic (only 5% of the total population are changing). The number of *zombies*, *M*, was set to 500 since there are about 500 flows. The number of lookups to find a hit in the *zombie* list was set to 1 and 50. The other parameters that are RED specific were set to the values that produced the best emulation results with RED. i.e. min_{th} = 60, max_{th} = 120, w_q = 0.1, max_p = 0.7.

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 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 buffer flow control (see equation (2.1)), there are three parameters to configure besides the AQM specific parameters: *decr*, *incr* and *RTT*. *RTT* is set to 200 msec

(160 msec + fuzzy factor, obtained from the system attributes described above), and *incr* is set to only 1% of the available PEP buffers to keep the recovery from congestion slow and conservative. Several permutations of *decr* were experimented to fine-tune it as we found that applying the 1/2 window reduction rule from RFC 2001 [1], RFC 2581 [20] and [21] is not applicable to high buffering systems. 97%, 95%, 90%, 80% and 50% were experimented for *decr*.

Parameters for AVQRED are discussed in the next chapter where its algorithm is described.

Table 3.1: AQM Parameters

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
	RED	α	*M*	*Lookups*
SRED 1	RED 4	0.1	500	1
SRED 2	RED 4	0.1	500	50
	γ	*B*		
AVQ	100%	123,750 Bytes		

Table 3.2: Buffer flow control parameters

Buffer FC AQMs	Parameters		
	decr	*incr*	*RTT [msec]*
RED FC 1	0.97	0.01×Buffer	200
RED FC 2	0.95	0.01×Buffer	200
RED FC 3	0.90	0.01×Buffer	200
RED FC 4	0.80	0.01×Buffer	200
RED FC 5	0.50	0.01×Buffer	200
AVQRED FC 1	0.97	0.01×Buffer	200
AVQRED FC 2	0.95	0.01×Buffer	200
AVQRED FC 3	0.90	0.01×Buffer	200
AVQRED FC 4	0.80	0.01×Buffer	200
AVQRED FC 5	0.50	0.01×Buffer	200

Chapter 4

Asynchronous Queueing and Global Synchronization

4.1. Problems

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 queueing behavior. Figure 4.1 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.

The problem with virtual-queue-based AQMs such as AVQ is global synchronization where consecutive packet drops occur due to their tail-drop nature of packet

marking. When packets are dropped consecutively, multiple TCP connections will react to the drops simultaneously resulting in under-utilization amongst multiple TCP connections.

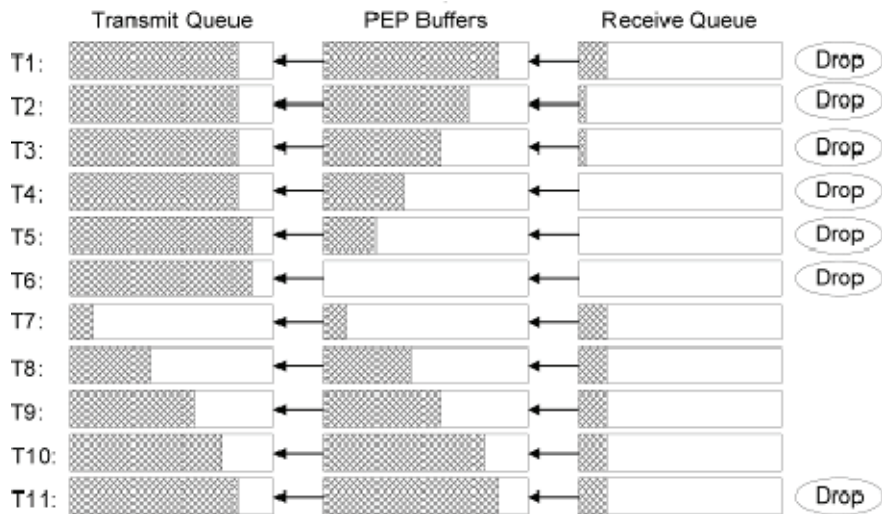

Figure 4.1: Asynchronous Queueing

This problem is severer with RED due to its oscillatory queueing behavior. 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.

4.2. Solution

A new AQM algorithm, Adaptive Virtual Queue Random Early Detection [6][7][8], is proposed to address the asynchronous queueing and the global synchronization problems.

```
for each packet arrival
/* Calculate virtual queue size */
δ <- curr_time - last_measure
if δ > 1
   /* Compute actual output rate in bps */
   tx_bytes <- bytes_transmitted
```

```
   output_rate <- (tx_bytes – prev_tx_bytes)* 8000 / δ
   prev_tx_bytes <- tx_bytes

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

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

last_measure <- curr_time
q_size <- VQ / AvgPktSize

/* 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 assumed to be greater than spacelink capacity, *min_capacity* should be equal to *max_capacity* and α can be any value.

AVQRED solves the asynchronous queueing 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.

AVQRED solves the global synchronization problem by preserving the global synchronization avoidance of the RED algorithm.

4.2.1. Parameter Settings

For our emulations, 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.

4.3. Emulation Results

RED, AVQ and AVQRED were emulated for 20 minutes with the parameter settings specified in Table 3.1 and section 4.2.1. Link utilization, queue size and packet drop metrics were measured once every 100 msec and the following subsections discuss the results for each metric. For link utilization and queue size histograms, only RED 4

out of the four RED settings is presented as it performed the best amongst the four RED settings.

4.3.1. Link Utilization

As Figure 4.2, Figure 4.3, and Table 4.1 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 56% gain in the stability (the standard deviation).

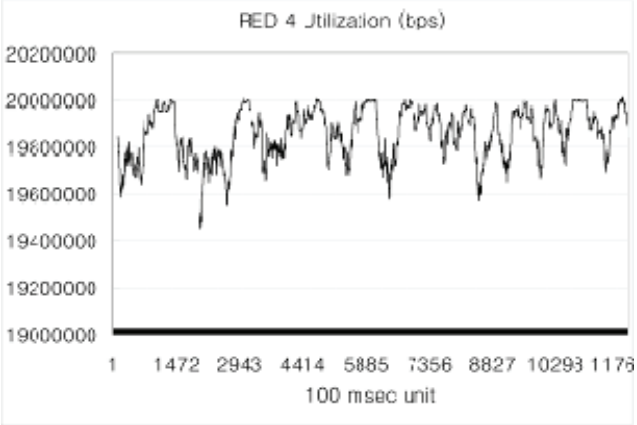

Figure 4.2: RED 4 link utilization

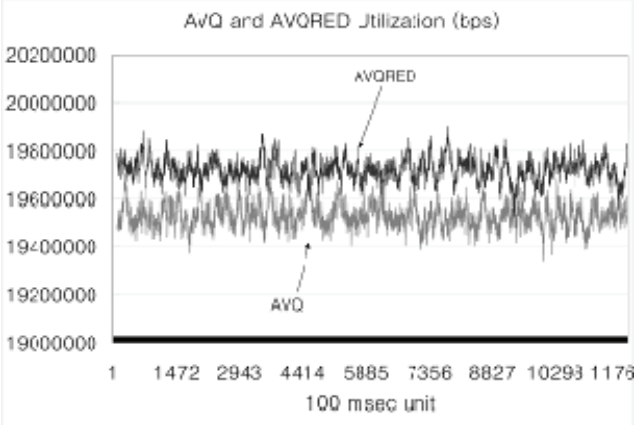

Figure 4.3: AVQ and AVQRED link utilzation

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 20 Mbps pipe whereas RED's utilization is oscillatory and unstable due to its asynchronous queueing behavior. Furthermore, RED's high utilization and low stability indicate that it tends to accept more data than the gateway capacity.

Table 4.1: 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

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 because consecutive packet drops cause multiple senders to shrink their congestion windows synchronously resulting in lower link utilization.

4.3.2. Queue Size

Figure 4.4, Figure 4.5 and Figure 4.6 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 4.7 and Figure 4.8 magnify Figure 4.4 and Figure 4.6 between 50^{th} and 150^{th} seconds (500^{th} ~ 1500^{th} points according to the x axis' scale). Table 4.2 summarizes mean and standard deviation of the queue size results.

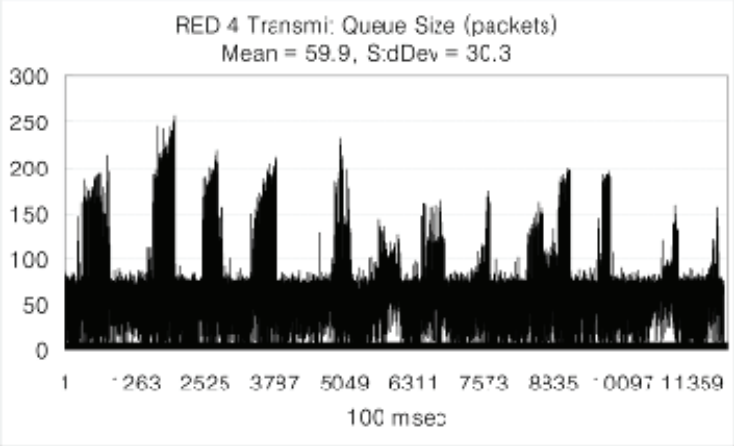

Figure 4.4: RED 4 transmit queue size

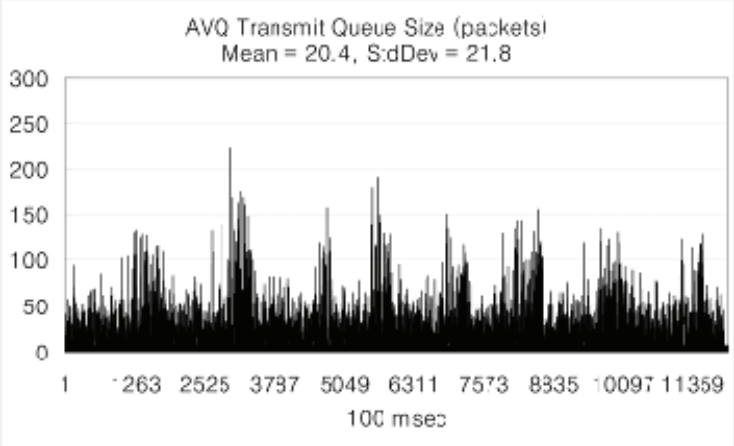

Figure 4.5: AVQ transmit queue size

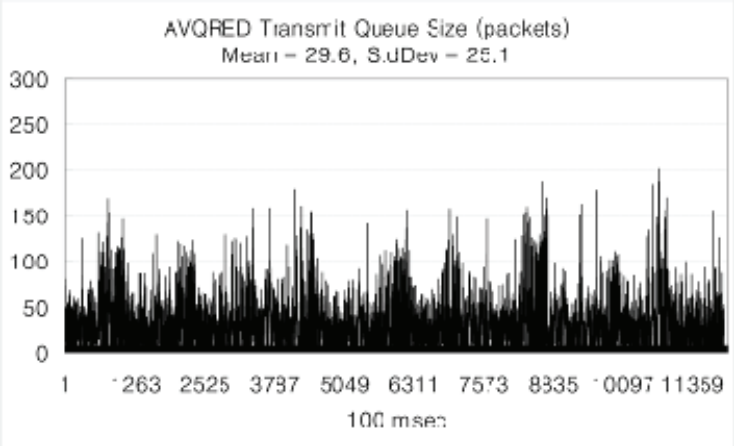

Figure 4.6: AVQRED transmit queue size

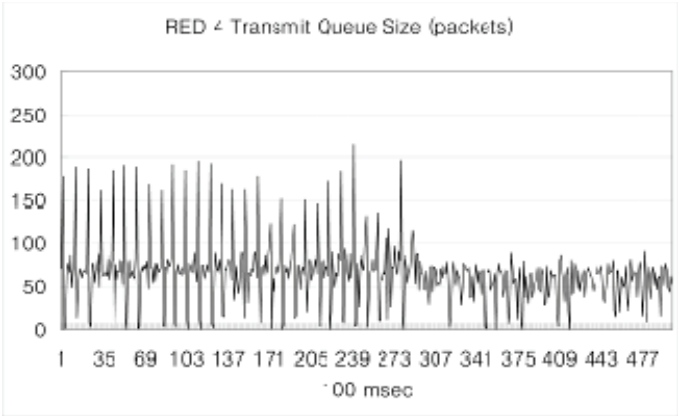

Figure 4.7: RED 4 transmit queue size (50^{th}~100^{th} seconds)

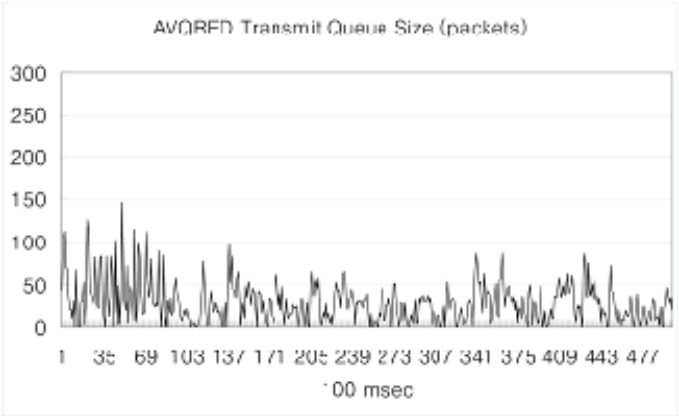

Figure 4.8: AVQRED transmit queue size (50^{th} ~ 100^{th} seconds)

Table 4.2: Queue size mean and standard deviation

AQM	Mean [packets]	Standard Deviation [packets]
RED 1	63.39	36.59
RED 2	59.15	33.42
RED 3	63.54	30.79
RED 4	59.91	30.34
AVQ	20.45	21.81
AVQRED	29.62	25.11

This oscillatory queueing behavior is the asynchronous queueing 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 queueing 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 cannot be done because the receive queue will never be congested when the bottleneck is the transmit queue by the spacelink bandwidth limitation.

4.3.3. Packet Drop

Because 20 minutes worth of the packet drop histogram is too long to present, only the first 1000 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 4.3.

Table 4.3: 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 4.9 shows packet drops for the first 1000 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 asynchronous queueing.

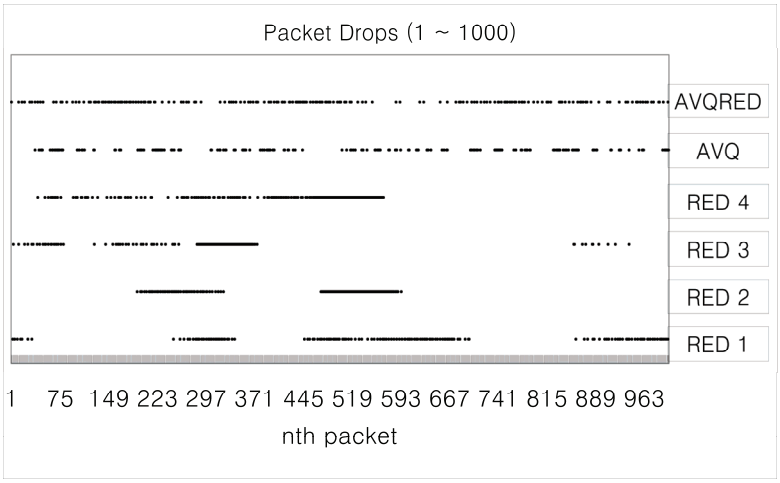

Figure 4.9: Packet drops for 1st ~ 1000th packets

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.

4.4. Mathematical Model

This section provides a mathematical model for the queueing behavior to validate the asynchronous queueing problem and our solution, AVQRED. The global synchronization problem is not validated mathematically, but the same model can be used to show the marking behavior by analyzing the standard deviation of the marking probability. Furthermore, we believe that the evidence of the global synchronization problem is quite visible in the AVQ algorithm and the emulation results.

$$\frac{d\lambda}{dt} = (1 - p(t)) \times \frac{m}{R^2} - p(t) \times \frac{\lambda^2}{2m} \tag{4.1}$$

$$\frac{dq}{dt} = -\mu + (1 - p(t - d(t))) \times \omega(t - d(t)) \times \lambda(t - d(t)) \tag{4.2}$$

$$\frac{dv}{dt} = -\tau \times \mu + (1 - p(t)) \times \omega(t) \times \lambda(t) \tag{4.3}$$

$$\frac{dw}{dt} = \frac{\log(1-B)}{\delta} \cdot w(t) - \frac{\log(1-B)}{\delta} \cdot q(t) \tag{4.4}$$

$$\frac{dp}{dt} = \begin{cases} -1.0 \times p(t) & \text{if } w \text{ (or } v) < \min_{th} \\ \dfrac{p_{\max}}{(\max_{th} - \min_{th})} \times \dfrac{dw}{dt} & \text{if } \min_{th} \le w \le \max_{th} \text{ for RED} \\ \dfrac{p_{\max}}{(\max_{th} - \min_{th})} \times \dfrac{dv}{dt} & \text{if } \min_{th} \le v \le \max_{th} \text{ for AVQRED} \\ 1.0 - p(t) & \text{else} \end{cases} \tag{4.5}$$

(4.1) is the ODE of the arrival rate of the offered load where R is the RTT between the gateway and internet hosts and *m* is the number of TCP connections. [22] 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 second to 100 msec.

(4.2) is the ODE of the transmit queue size where μ is the service rate (20 Mbps with 1400 bytes per packet and 100 msec 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 (shown in Figure 4.10 and Figure 4.11) 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 msec 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.

(4.3) is the ODE of the virtual queue size where τ is the target utilization. Note that it is similar to (4.2) except that it does not have the PEP buffering delays.

(4.4) is the ODE of the weighted average transmit queue size from [23]. B is w_q and δ is the smallest time unit of our ODE approximation which is 1 msec (= 0.01 of 100 msec).

(4.5) 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.

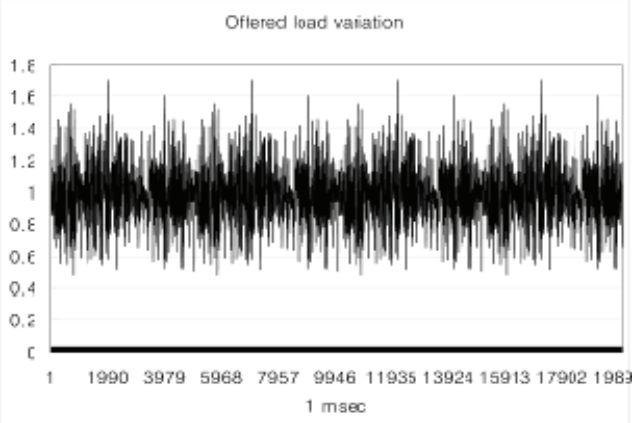

Figure 4.10: Fourier series for offered load variation

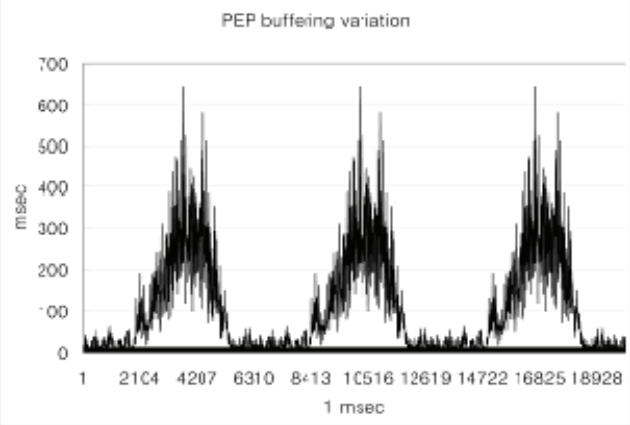

Figure 4.11: Fourier series for PEP buffering variation

4.5. MATLAB Results

To validate the asynchronous queueing problem, the above ODEs were fed to MATLAB and the transmit queue size (4.2) was examined for both RED (with RED 4 parameters in Table 3.1) and AVQRED. As Figure 4.12 shows, RED has the same oscillatory queueing behavior as the one that Figure 4.4 shows. The mean is slightly lower than the emulation because the ODEs did not account for the 5 msec queueing latency caused by the output rate regulator. The standard deviation is slightly higher than the emulation because the fourier series for the PEP buffering delays was approximated using only 500 data points which resulted in more frequent and regular oscillations.

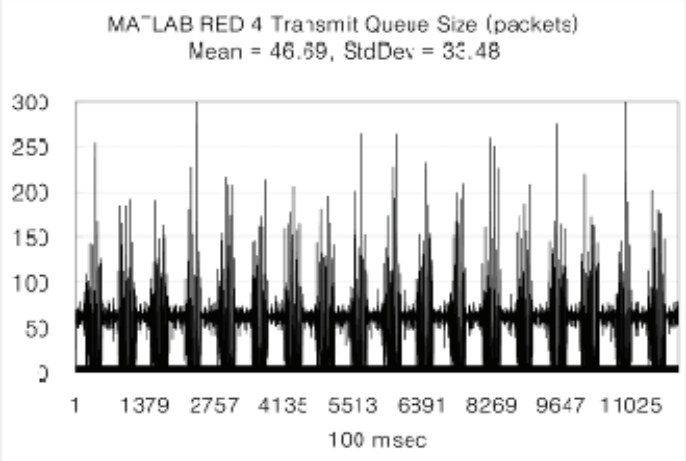

Figure 4.12: RED 4 transmit queue size (MATLAB)

As Figure 4.13 shows, AVQRED fixes the oscillatory queueing behavior. The mean and standard deviation are slightly different from the emulation because of the 5 msec latency and the relatively small fourier sample space.

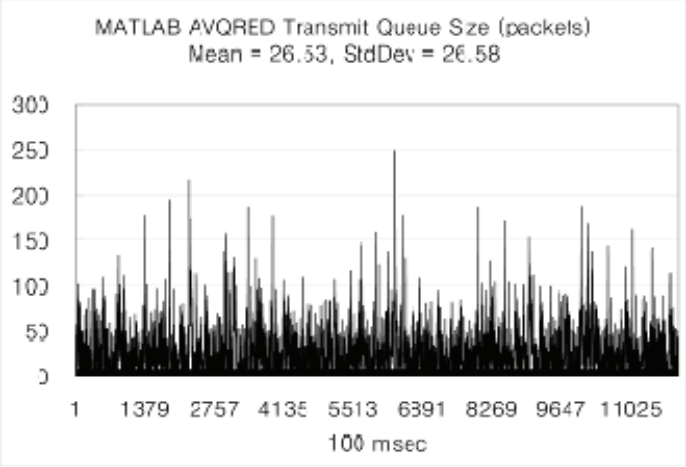

Figure 4.13: AVQRED transmit queue size (MATLAB)

To summarize and compare the improvement percent of mean and standard deviation, Figure 4.14 is provided. Figure 4.14 depicts that the MATLAB results concur with the emulation results. As stated earlier, the small discrepancies between emulation and MATLAB are from the 5 msec rate regulator latency and the small fourier sample space.

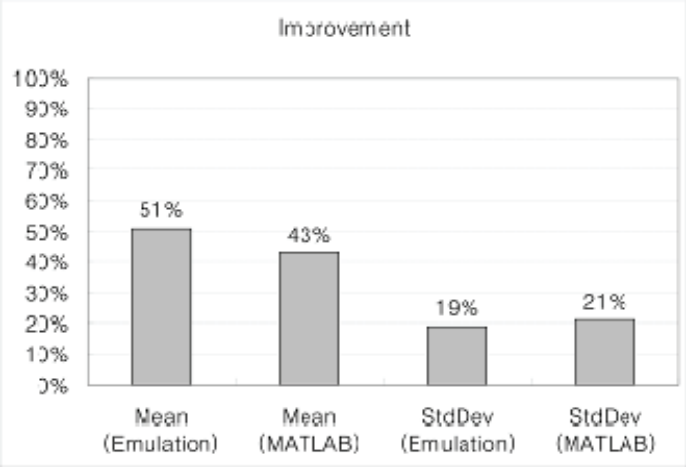

Figure 4.14: Transmit queue size improvement by AVQRED

4.6. Summary

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 queueing and global synchronization. A new AQM method, AVQRED, was developed to fix the problems.

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' HughesNet*® networks and a traffic generator called *Spirent*.

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

Chapter 5

Fair Bandwidth Sharing

5.1. Problem

Becase the bandwidth is scarce in satellite networks, an unfair distribution of the bandwidth can easily sacrifice light users. Therefore, the fairest bandwidth distribution is necessary to ensure each user gets at least the minimum bandwidth he deserves during congestion.

The main reason why existing AQMs do not perform well in terms of the fairness metric is that they do not maintain full per-flow states to avoid adding complexity to the algorithm. However, the complexity added to AVQRED to make it fully per-flow aware is not much due to its virtual-queue-based queueing. Even if the complexity is much, giving up the fairness in networks with limited bandwidth does not seem reasonable.

5.2. Solution

The solution we propose is an extension of the AVQRED algorithm called PFAVQRED [9]. We extend the algorithm by adding a per-flow weight to the AVQRED marking probability. The per-flow weight, pf_{weight}, and the final marking probability, p_a, are calculated by the following equations:

$$pf_{weight} = \begin{cases} 0 & \text{if flow age } \leq 100 \text{ msec} \\ \left(\frac{pf_{rate}}{pf_{fair\_rate}} \right)^2 & \text{otherwise} \end{cases}$$

$$pf_{rate} = \frac{bytes\_transmitted \times 8}{duration}$$

$$pf_{fair\_rate} = \frac{v\_capacity}{total\_number\_of\_flows}$$

$$p_a = p_a \times pf_{weight}$$

Because we want to avoid packet drops during the connection startup, 0 is assigned to pf_{weight} when the connection was started less than 100 msec ago. The $(.)^2$ part of the pf_{weight} calculation is to provide stiffer penalties for misbehaving flows. By applying pf_{weight} to the existing AVQRED marking probability, p_a, there is no need to create N different virtual queues for N flows. This is the main difference between other virtual-queue-based flow-aware AQMs such as VQ-RED. By having only one virtual queue, configuration of the congestion region ($min_{th} \sim max_{th}$) is intuitive and the outcomes of the virtual queue processing are much more predictable and accurate.

The complexity overhead of PFAVQRED is not any worse than other flow-aware AQM methods such as SRED, CHOKe and VQ-RED because they all need a mechanism to identify flows and store the flow information for some (or all) active flows. The only difference is the memory usage because AQMs such as SRED and CHOKe store the flow information of some active flows whereas PFAVQRED stores it for all active flows. However, we claim that consumption of a few extra kilo-bytes is not an issue with gateways these days.

5.2.1. Parameter Settings

As there are no extra parameters for PFAVQRED, the same parameters used for AVQRED are used for the PFAVQRED emulations.

5.3. Emulation Results

As noted in section 3.2, three UDP flows were injected to the emulations, and per-flow throughput was measured for AVQRED, PFAVQRED and each of the AQMs in Table 3.1. Because the link utilization and queue size results of SRED and PFAVQRED are almost the same as the results of RED and AVQRED, these two metrics are not presented in this chapter as they are already presented in Chapter 4. Therefore, only the packet drop and per-flow throughput metrics are presented in this chapter.

5.3.1. Packet Drops

Table 5.1: Total packet drops

AQM	Total packet drops
RED 1	465,052
RED 2	464,273
RED 3	466,136
RED 4	466,416
SRED 1	466,772
SRED 2	515,847
AVQ	450,021
AVQRED	450,010
PFAVQRED	631,931

Table 5.1 shows that SRED and PFAVQRED have more packet drops than their ancestors, RED and AVQRED respectively. This confirms that their fairness enforcement is the best amongst all of the AQMs experimented. The fact that SRED 2 has

more packet drops than SRED 1 confirms that the SRED algorithm performs better when there are more number of lookups for *zombie* list hits.

5.3.2. Per-flow Throughput

Figure 5.1 through Figure 5.9 show the per-flow throughputs for each AQM setting we experimented. The purpose of this data is to provide how uniform the bandwidth distribution is amongst all the flows. The x axis represents the 3 UDP and the 500 HTTP flows, and the y axis represents the average throughput in bps for each flow. The optimal bandwidth share for each flow would be around 40 Kbps (20 Mbps / 503 flows). The wider the variation is, the less fair the bandwidth sharing is. All of the AQMs have similar variation except for SRED 2 and PFAVQRED. SRED 2 produced fairer bandwidth sharing compared to SRED 1 because it has a higher hit rate by doing more *zombie* list lookups which penalized misbehaving flows more. The fact that PFAVQRED produced the lowest variation in throughputs proves that it has the fairest bandwidth allocation for each flow. To summarize the throughput variation, standard deviation of each AQM is provided in Table 5.2.

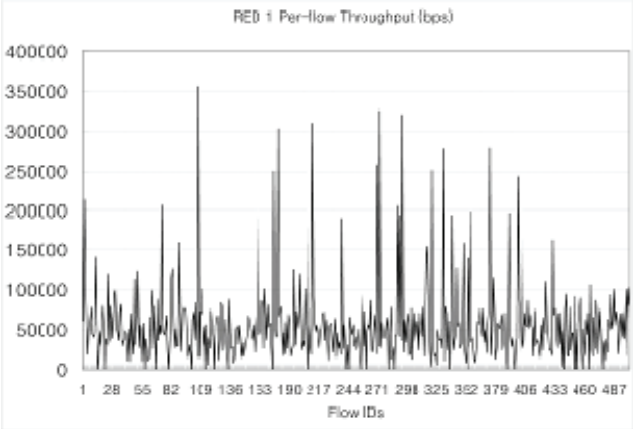

Figure 5.1: RED 1 per-flow throughput

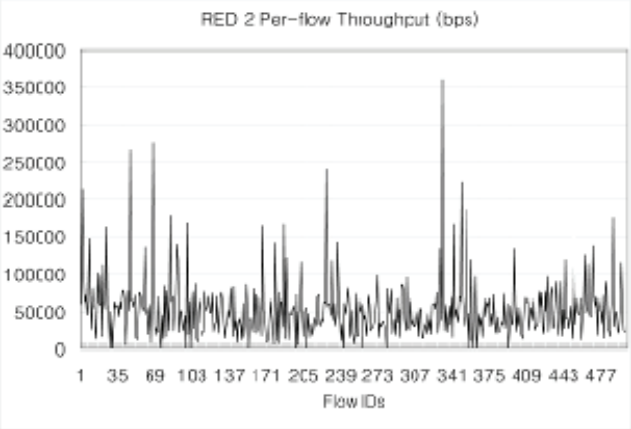

Figure 5.2: RED 2 per-flow throughput

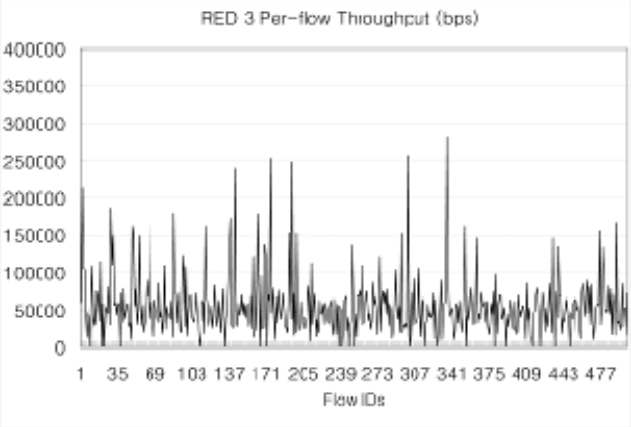

Figure 5.3: RED 3 per-flow throughput

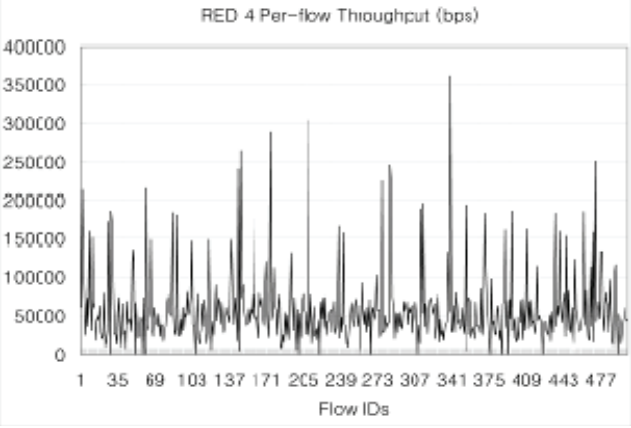

Figure 5.4: RED 4 per-flow throughput

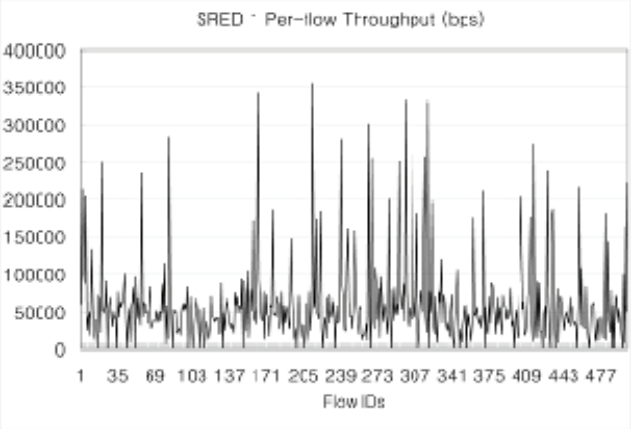

Figure 5.5: SRED 1 per-flow throughput

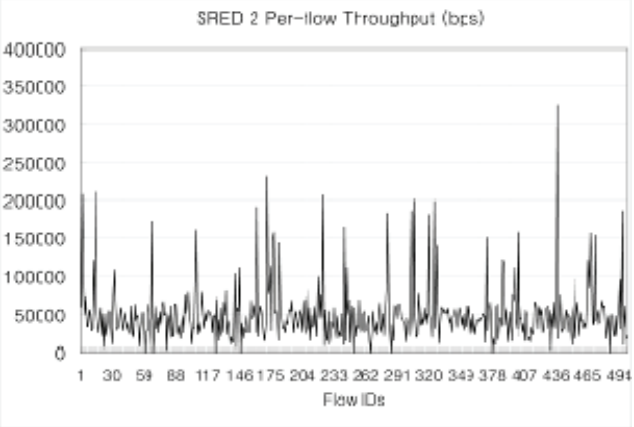

Figure 5.6: SRED 2 per-flow throughput

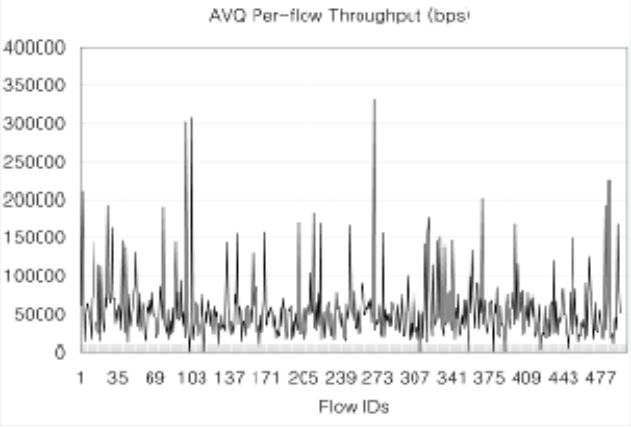

Figure 5.7: AVQ per-flow throughput

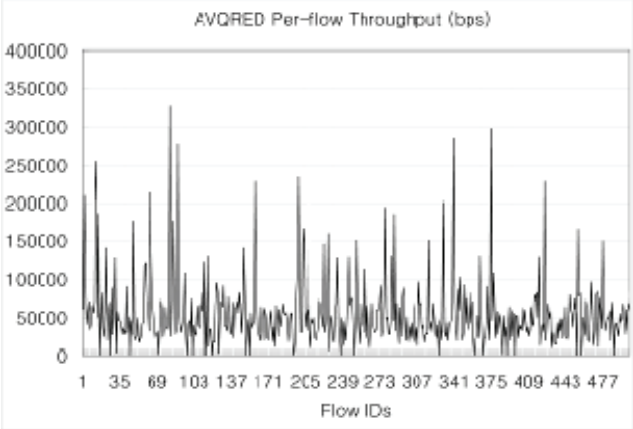

Figure 5.8: AVQRED per-flow throughput

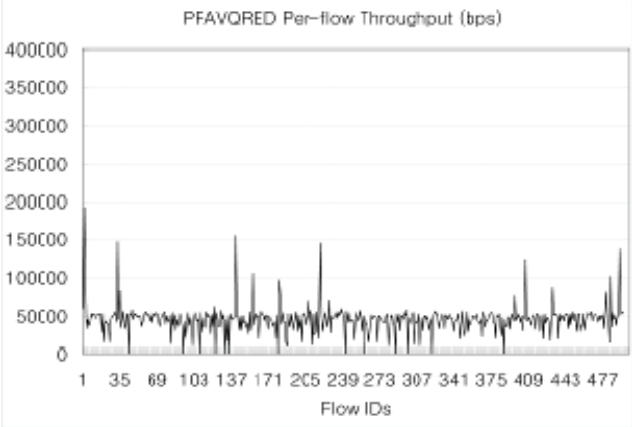

Figure 5.9: PFAVQRED per-flow throughput

Table 5.2: Per-flow throughput standard deviation

AQM	Standard deviation [bps]
RED 1	51,161
RED 2	39,841
RED 3	39,640
RED 4	47,892
SRED 1	55,953
SRED 2	37,189
AVQ	42,351
AVQRED	45,793
PFAVQRED	18,150

Figure 5.10 shows the throughputs for each of the three greedy (misbehaving) UDP flows (80 Kbps, 160 Kbps, 320 Kbps). The purpose of the charts is to show how

much each UDP flow is penalized by each AQM method. The closer to the optimal share (40 Kbps) the throughput is, the fairer bandwidth allocation the AQM provides. It confirms that SRED 2 and PFAVQRED penalize misbehaving flows more than the other AQMs. It also confirms that PFAVQRED's fairness enforcement is the best amongst these AQMs which is consistent with what Figure 5.1 through Figure 5.9 show.

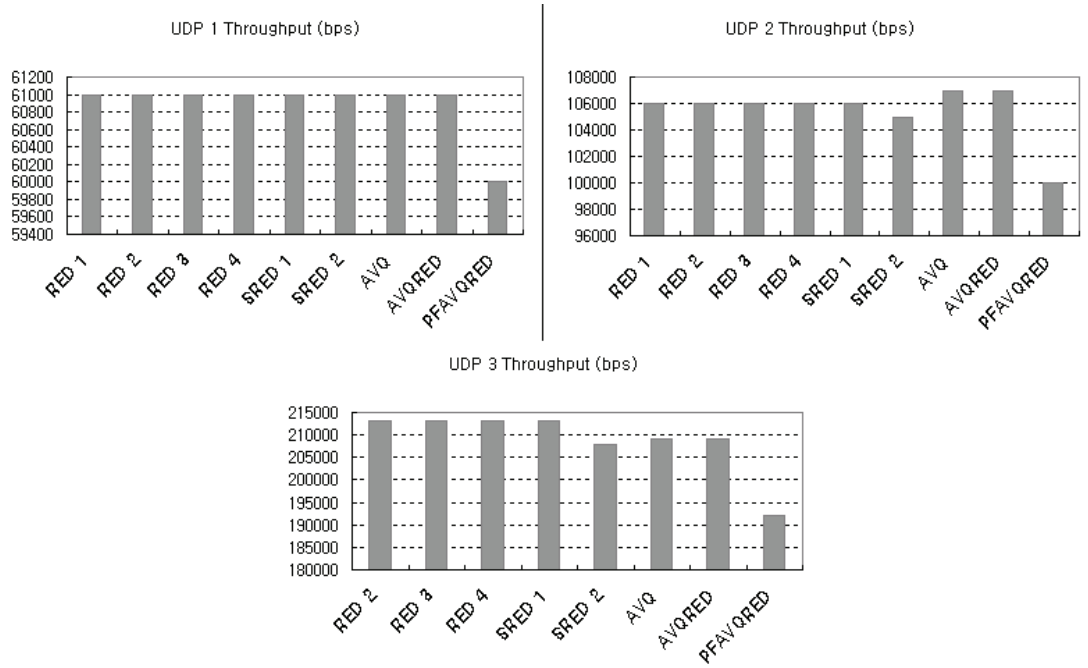

Figure 5.10 UDP throughputs

From the data presented in this section, we can conclude that PFAVQRED successfully enhances AVQRED and provides close to the optimal bandwidth sharing. The per-flow throughput results are also consistent with the packet drop results because PFAVQRED has the highest number of packet drops due to its ability to penalize misbehaving flows.

5.4. Summary

As an extension of AVQRED, PFAVQRED was developed to improve the fairness. The key difference between AVQRED and PFAVQRED is that PFAVQRED applies a per-flow weight (which is derived from the bandwidth usage of each flow) to the marking probability to penalize misbehaving flows and to maintain fair bandwidth usage.

PFAVQRED was compared with RED, SRED, AVQ and AVQRED with an additional performance metric, per-flow throughput. RED and SRED performed similarly in terms of link utilization and queue size. AVQRED and PFAVQRED also performed similarly in terms of link utilization and queue size. However, SRED and PFAVQRED dropped more packets due to their ability to penalize misbehaving flows. In terms of the fairness, PFAVQRED performed the best as it produced close to the optimal bandwidth distribution.

Chapter 6

Buffer Flow Control Instability

6.1. Problems

One of the problems with buffer flow control is its abrupt and unstable TCP receive window adjustments for new connections. Because each PEP connection has to make sure its maximum PEP buffer limit is at least one round trip worth, the TCP receive window is usually smaller than the PEP buffer limit. This discrepancy introduces significant delays (between the start of buffer reduction and the actual TCP receive window reduction) that cause queueing instability and failure to effectively back pressure the senders. Figure 6.1 illustrates a buffer flow control scenario where 625 KB is allocated for the PEP buffers of each user, the PEP buffer limit is decremented by 10 % of the previous limit upon a packet mark, and the maximum TCP receive window is set to 16,000 Bytes. The figure depicts that the congestion meter is required to be lower than 0.0256 to advertise a smaller TCP window which translates into 36 consecutive packet marks. Until then, the senders do not slow down the transmissions causing not only abrupt reductions in the transmission rate but also oscillatory queueing behavior in the gateway.

The other problem is that the arrival packets are always accepted regardless of the congestion level of the queue aggravating the congestion.

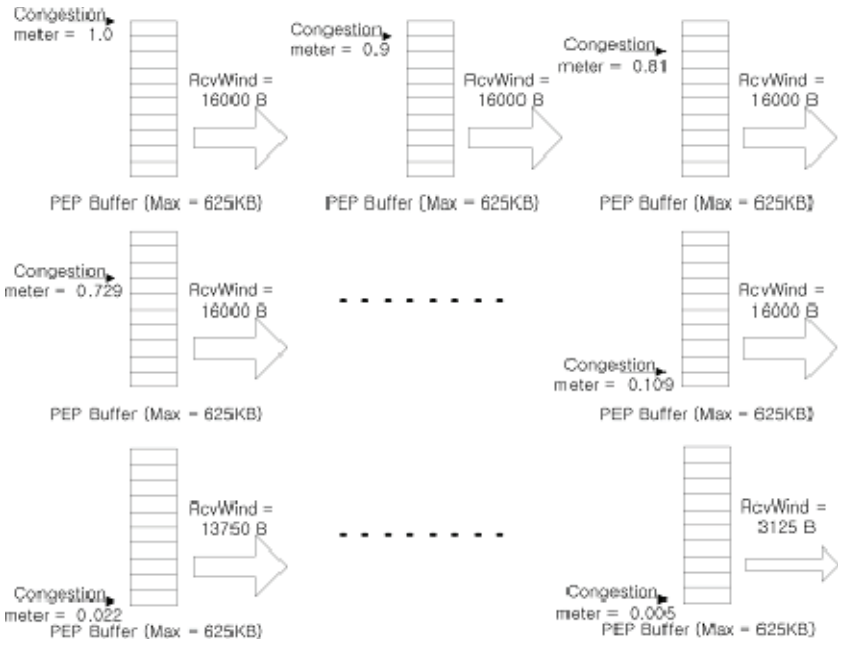

Figure 6.1: Abrupt TCP receive window adjustments by buffer flow control

6.2. Solution

To fix the abrupt TCP receive window adjustments and the congestion aggravation problems, we propose to use AQM with packet drop marking to shift the rate controlling entity from the receiver to the senders where rate adjustments are smoother because there is no PEP buffering in the senders. As we discussed in Chapter 4, we recommend AVQRED for the AQM method because other existing AQMs have the asynchronous queueing and/or global synchronization problems.

6.3. Emulation Results

Each of the AQM methods in Table 3.2, RED 4 and AVQRED were emulated for 15 minutes. All three performance metrics were collected: link utilization, transmit queue size and packet drop. For the link utilization metric, both input and output link

utilizations were collected to show the savings in the input link utilization by not dropping packets with the buffer flow control based AQM methods. For the packet drop metric, the marking probability is shown instead of the actual marking instances to provide a global visualization in terms of how effective each AQM method is in controlling the senders' transmit rate. Per-flow throughput metric was not collected as the problem is not related to the fairness.

6.3.1. Output Link Utilization

As Figure 6.2 and Figure 6.3 show, the output link utilization of buffer flow control AQMs becomes unstable as the decrement factor, *decr*, increases. As Figure 6.2, Figure 6.3, Figure 6.4 and Table 6.1 show, the first two settings of each AQM (RED FC 1, RED FC 2, AVQRED FC 1 and AVQRED FC 2) yield output link utilizations that are somewhat comparable to RED and AVQRED. However, the queue size and stability are far worse than the packet drop AQMs as revealed in section 6.3.3.

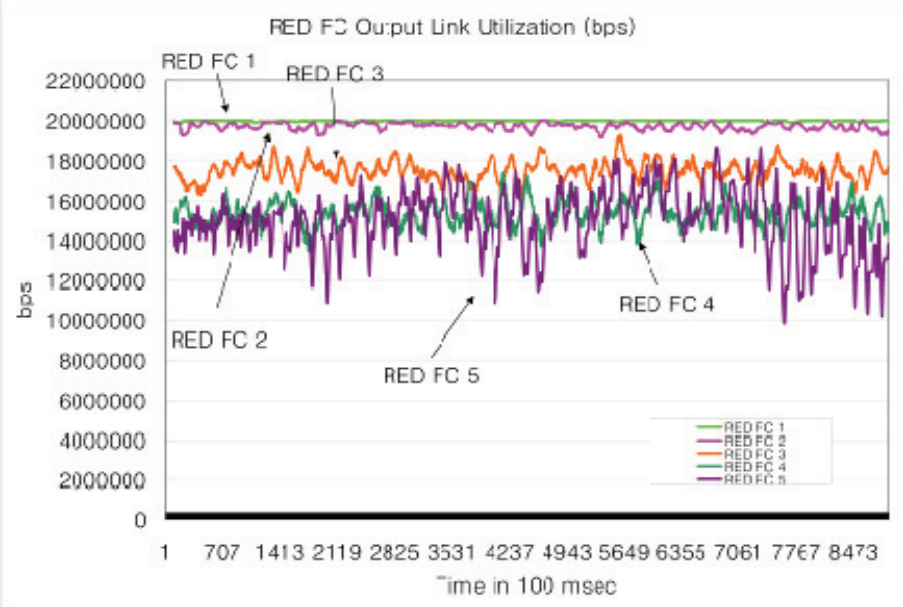

Figure 6.2: Buffer flow control RED output link utilization

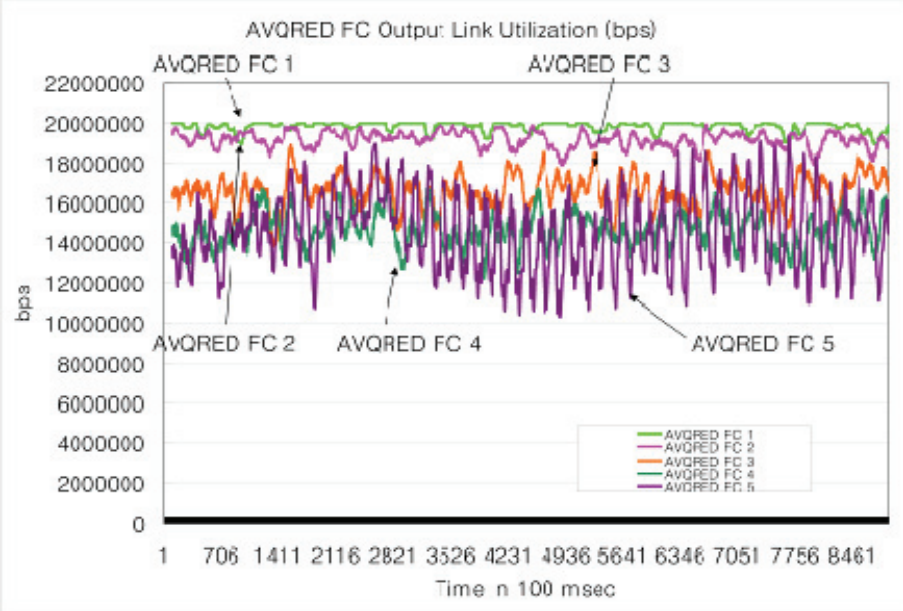

Figure 6.3: Buffer flow control AVQRED output link utilization

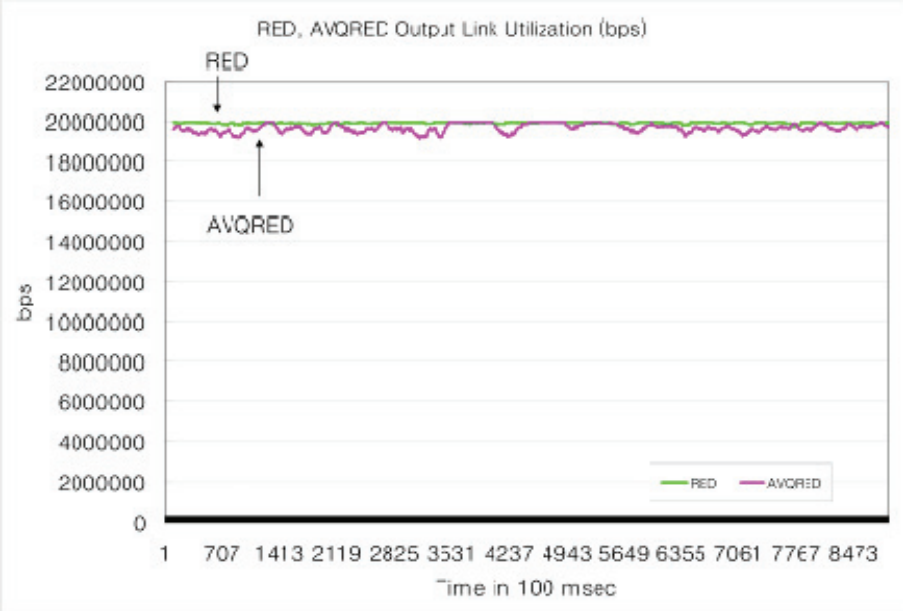

Figure 6.4: Packet drop RED and AVQRED output link utilization

Although analyzing the differences between RED and AVQRED (for both buffer flow control and packet drop methods) is not within the scope of this chapter, some explanation is provided for completeness as follows: The reason why AVQRED has lower link utilization and higher standard deviation is that it tends to react to conges-

tion earlier than RED resulting in slightly lower link utilization but lower and more stable queue size as depicted in section 6.3.3. The lower standard deviation of RED utilization seems to contradict the results in Table 4.1, but the standard deviation of RED is lower in this case because the transmit rate regulator lowers the output variation while the transmit queue is over-loaded by the asynchronous queueing and higher offered load. The higher offered load was caused by hundreds of extra TCP connections[2] that were created by unstable AQMs (RED FC 5 and AVQRED FC 5) that were executed right before RED and AVQRED were executed. In other words, the utilization becomes flatter as the transmit queue is occupied more frequently; and the transmit queue is occupied unnecessarily frequently by RED due to its asynchronous queueing problem when the offered load is abnormally high.

Table 6.1: Mean and standard deviation of output link utilization

AQM	Mean [bps]	StdDev [bps]
RED FC 1	19,976,464	32,300
RED FC 2	19,710,622	169,424
RED FC 3	17,491,225	503,592
RED FC 4	15,351,695	627,044
RED FC 5	15,006,684	1,434,812
AVQRED FC 1	19,834,171	213,647
AVQRED FC 2	19,181,604	379,949
AVQRED FC 3	16,633,357	809,237
AVQRED FC 4	14,672,899	832,690
AVQRED FC 5	14,580,907	1,916,090
RED	19,966,805	34,937
AVQRED	19,715,454	204,944

In summary, there exist some parameter settings (RED FC 1, RED FC 2, AVQRED FC 1, and AVQRED FC 2) for buffer flow control AQMs that result in output link utilizations that are comparable to packet drop AQMs.

[2] This may be a limitation with the *Spirent* traffic generator which will not be explored any further.

6.3.2. Input Link Utilization

The input link utilization captures the link utilization before the AQM marking and the purpose of this metric is to show the terrestrial bandwidth usage. Although it is clear that the buffer flow control AQMs provide much more efficient usage by not dropping marked packets, it needs to be understood that stable queueing, low queue size and high output utilization are far more important factors in satellite networks because the scarce resource is the satellite link almost all the time (not the terrestrial links fed to satellite gateways).

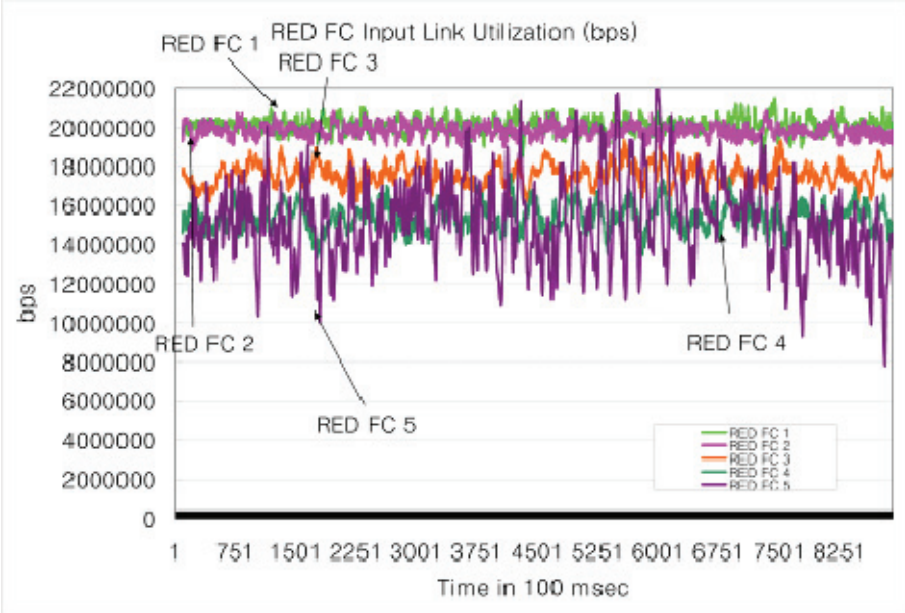

Figure 6.5: Buffer flow control RED input link utilization

Table 6.2 is consistent with Table 6.1 except that 1) the packet drop AQMs (RED and AVQRED) have higher link utilization due to their packet drop marking and 2) higher variation in utilization due to the unregulated offered load. Packet drop AQMs dropped about 19 % of the original offered load after AQM marking which is

consistent with the average packet drop probability (16.7% for RED and 18.1% for AVQRED).

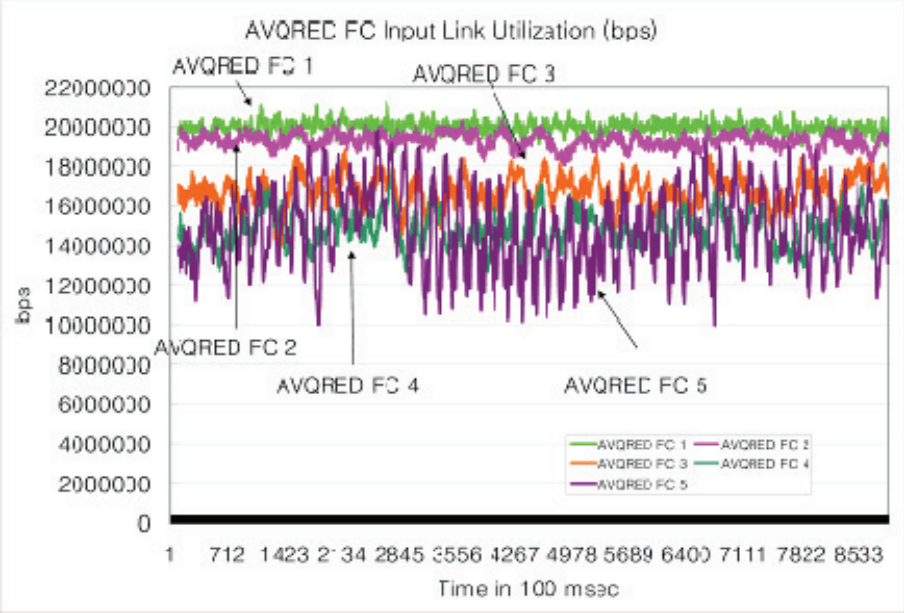

Figure 6.6: Buffer flow control AVQRED input link utilization

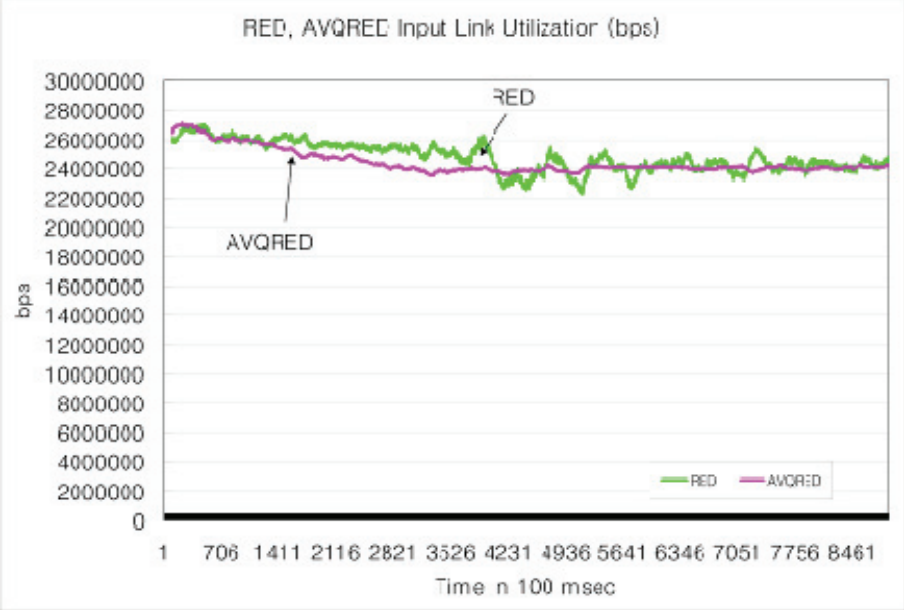

Figure 6.7: Packet drop RED and AVQRED input link utilization

In summary, buffer flow control AQMs improved the terrestrial bandwidth usage efficiency by the average packet drop probability. In our traffic model, the aver-

age packet drop probability was around 17~18 % and about 19 % efficiency improvement was achieved by the buffer flow control AQMs in the terrestrial links.

Table 6.2: Mean and standard deviation of input link utilization

AQM	Mean [bps]	StdDev [bps]
RED FC 1	20,124,689	372,613
RED FC 2	19,840,931	325,317
RED FC 3	17,592,220	540,831
RED FC 4	15,441,918	660,253
RED FC 5	15,180,693	2,112,514
AVQRED FC 1	19,981,757	277,339
AVQRED FC 2	19,300,427	359,035
AVQRED FC 3	16,729,259	757,070
AVQRED FC 4	14,798,855	837,137
AVQRED FC 5	14,819,918	1,980,519
RED	24,823,058	1,009,220
AVQRED	24,447,902	860,513

6.3.3. Transmit Queue Size

Except for the ones that resulted in heavy under-utilization, buffer flow control AQMs resulted in very high and unstable queue size compared to packet drop AQMs. As Table 6.3 summarizes, all of the buffer flow control AQMs are worse than packet drop AQMs in terms of transmit queue size. The most comparable AQMs are RED FC 4, AVQRED FC 3 and AVQRED FC 4 but they have heavy under-utilization. These results confirm the problem described in section 6.1. For example, slow reaction to congestion by RED FC 1 and AVQRED FC 1 confirms that buffer adjustments do not effectively flow control the senders. Faster reaction to congestion but lower utilization by RED FC 4 and AVQRED FC 4 confirm that PEP buffers need to be brought well below the full-utilization level (below the maximum TCP receive window = 16,000 Bytes) in order to effectively flow control the senders. The results for

RED FC 5 and AVQRED FC 5 will not be considered as their queueing stability goes beyond a reasonable range.

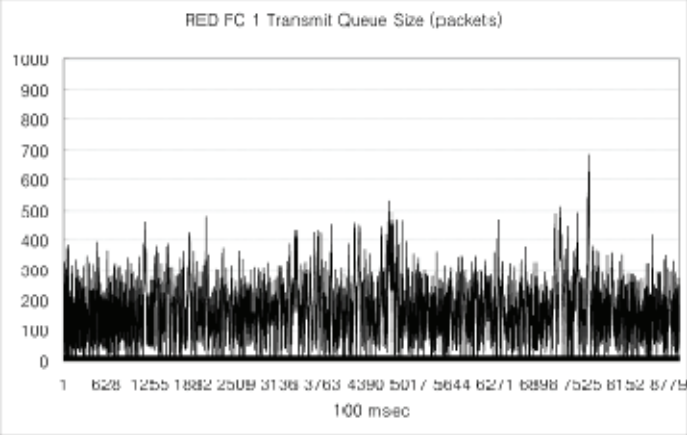

Figure 6.8: RED FC 1 transmit queue size

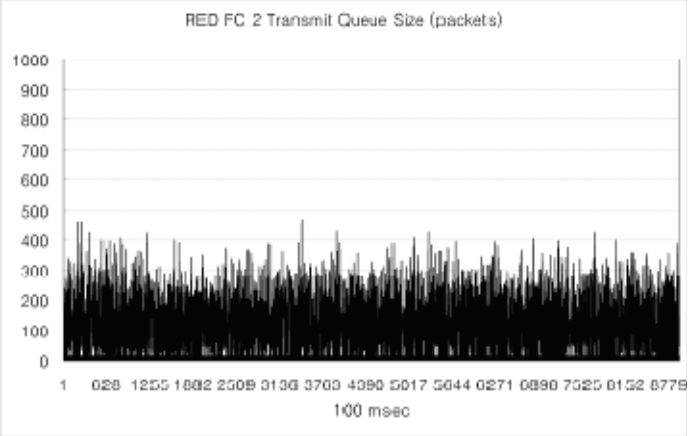

Figure 6.9: RED FC 2 transmit queue size

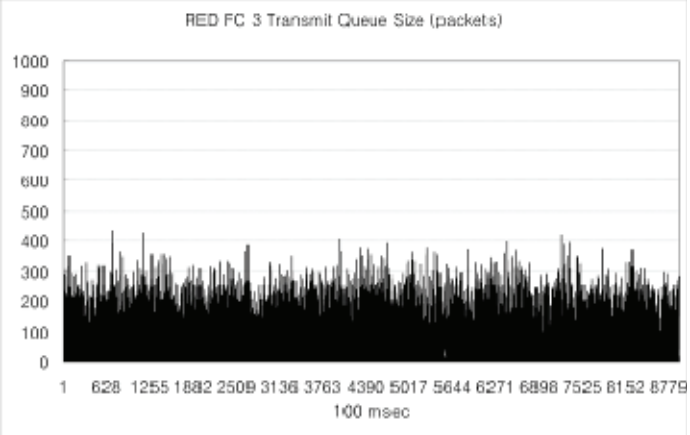

Figure 6.10: RED FC 3 transmit queue size

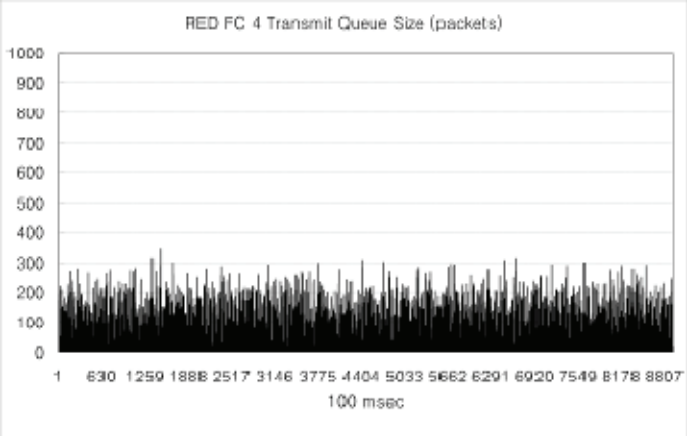

Figure 6.11: RED FC 4 transmit queue size

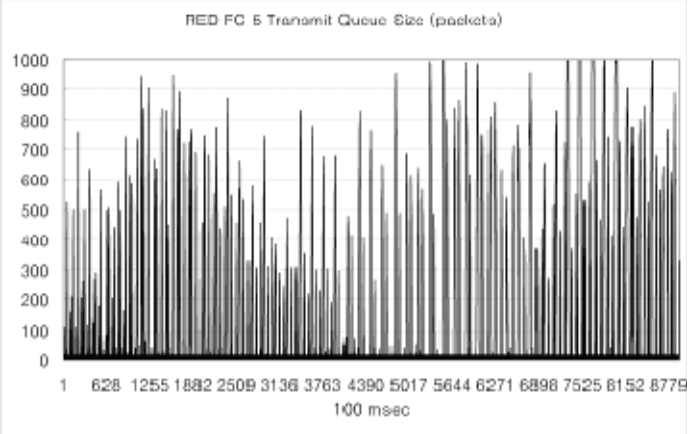

Figure 6.12: RED FC 5 transmit queue size

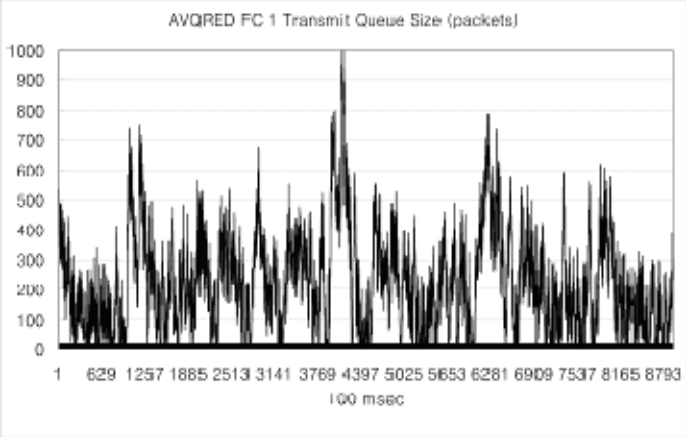

Figure 6.13: AVQRED FC 1 transmit queue size

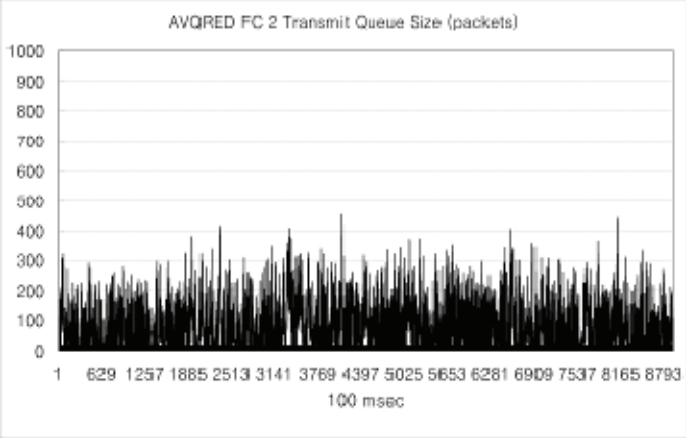

Figure 6.14: AVQRED FC 2 transmit queue size

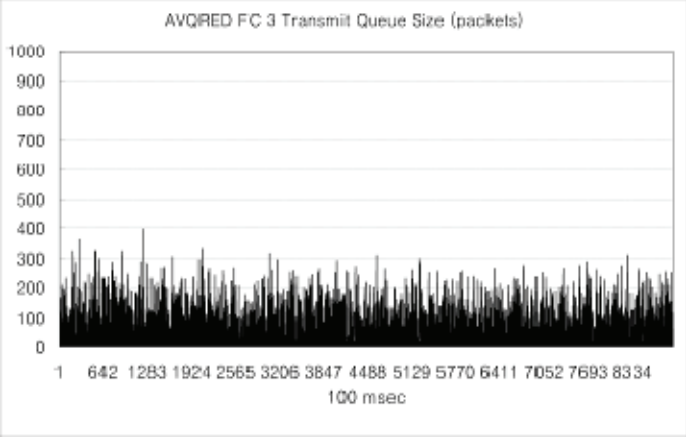

Figure 6.15: AVQRED FC 3 transmit queue size

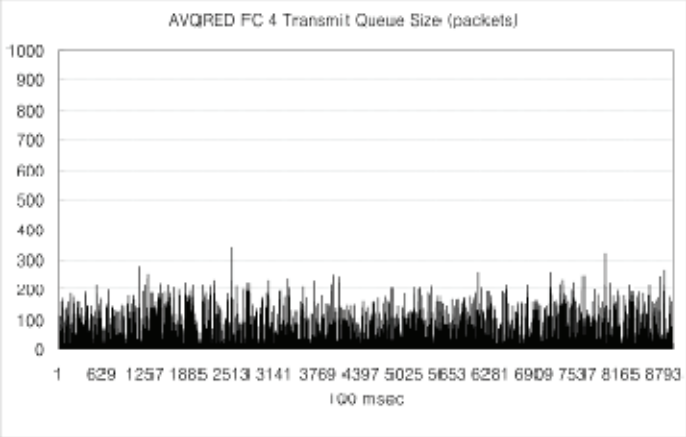

Figure 6.16: AVQRED FC 4 transmit queue size

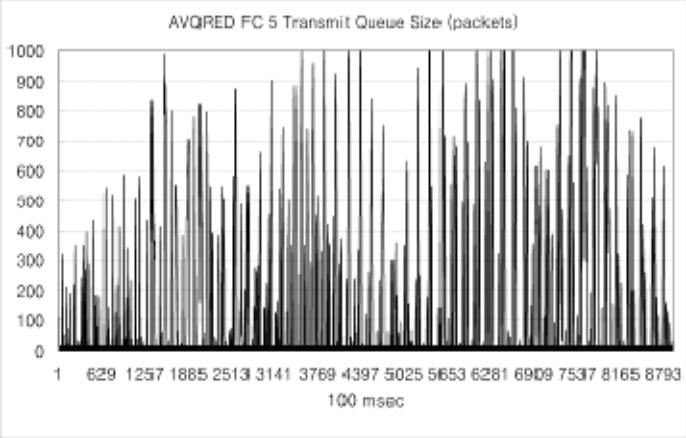

Figure 6.17: AVQRED FC 5 transmit queue size

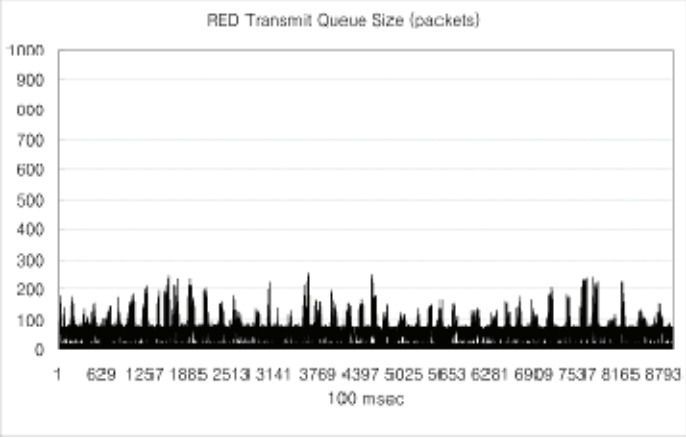

Figure 6.18: RED transmit queue size

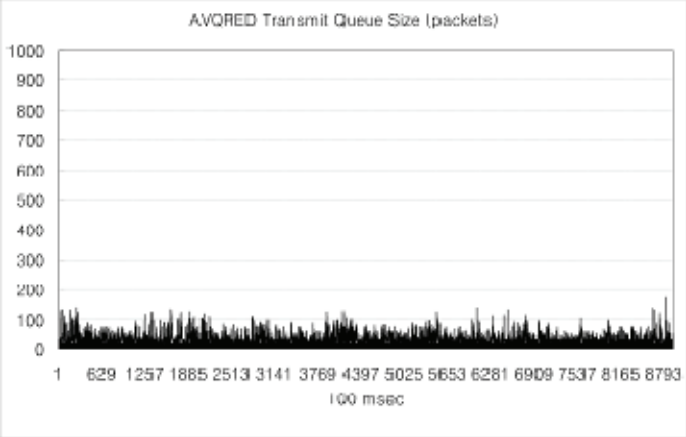

Figure 6.19: AVQRED transmit queue size

The spikes in the RED transmit queue size histogram (Figure 6.18) are due to the asynchronous queueing problem discussed in section 4.1.

Table 6.3: Mean and standard deviation of transmit queue size

AQM	Mean [packets]	StdDev [packets]
RED FC 1	157.52	90.09
RED FC 2	119.59	90.48
RED FC 3	81.68	91.61
RED FC 4	45.17	63.65
RED FC 5	158.26	271.02
AVQRED FC 1	213.21	162.41
AVQRED FC 2	78.12	77.17
AVQRED FC 3	47.13	62.68
AVQRED FC 4	27.38	45.26
AVQRED FC 5	153.06	259.57
RED	55.81	30.49
AVQRED	28.48	23.05

In summary, none of the buffer flow control AQMs maintained the target queue size (= less than 90 packets) while preserving utilization close to 20 Mbps. On the other hand, all of the packet drop AQMs maintained the target queue size with close to the full utilization. The results confirm that the buffer flow control AQMs do not effectively flow control the senders because they accept packets even when the

transmit queue detects congestion and new connections allow undesirable bursts to enter the congested queue.

6.3.4. Marking Probability

Marking probability shows how effective each AQM method is. As shown in Figure 6.20 through Figure 6.29, all of the buffer flow control AQMs have higher marking probabilities than the packet drop AQMs. These results confirm that the buffer flow control method does not effectively flow control the senders as depicted in section 6.3.3. One of the noticeable consequences of the ineffectiveness in flow controlling the senders is queue instability. As shown in section 6.3.3, all of the buffer flow control AQMs have more unstable queueing behavior than the packet drop AQMs.

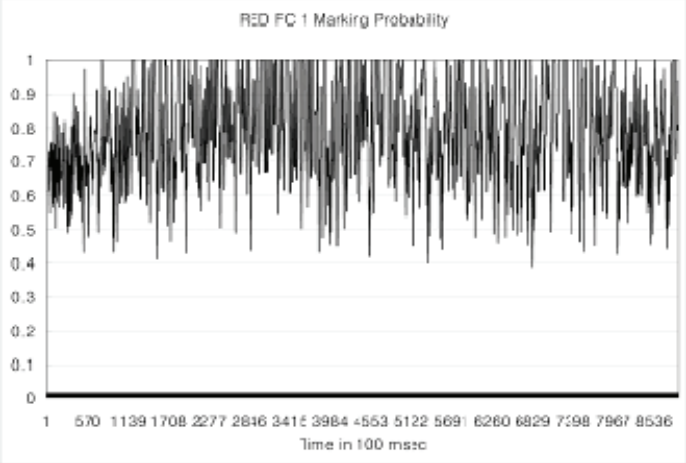

Figure 6.20: RED FC 1 marking probability

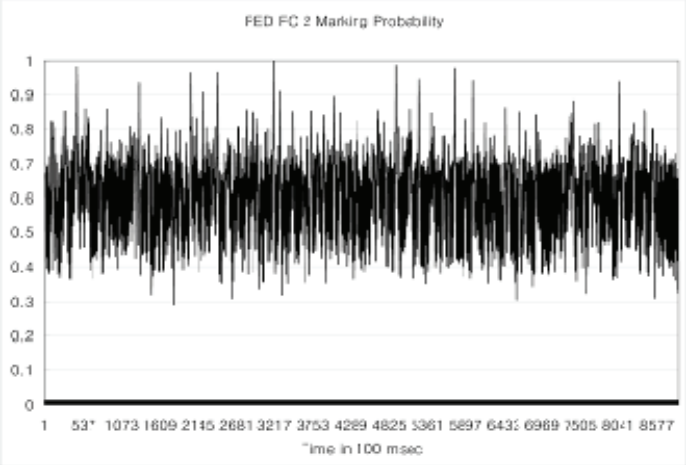

Figure 6.21: RED FC 2 marking probability

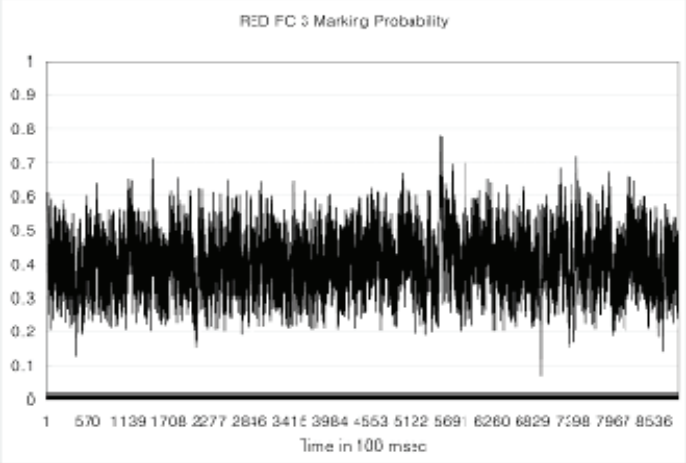

Figure 6.22: RED FC 3 marking probability

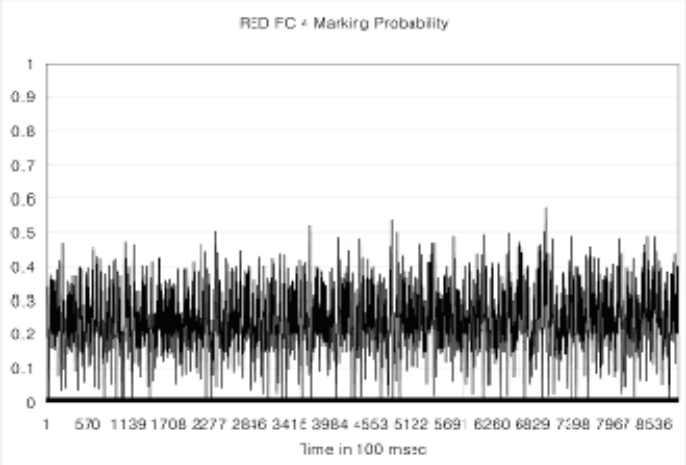

Figure 6.23: RED FC 4 marking probability

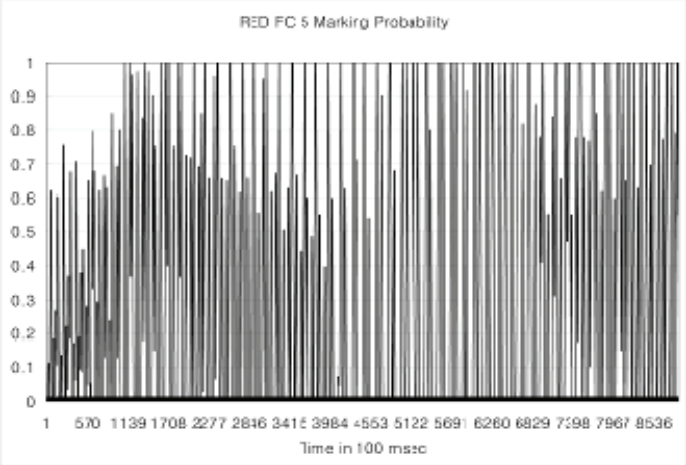

Figure 6.24: RED FC 5 marking probability

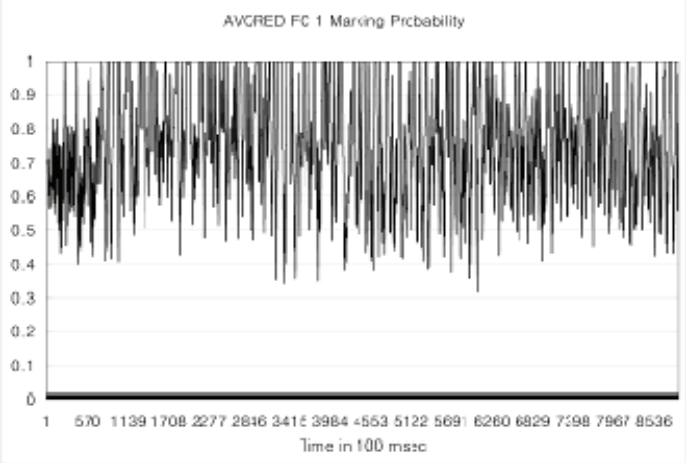

Figure 6.25: AVQRED FC 1 marking probability

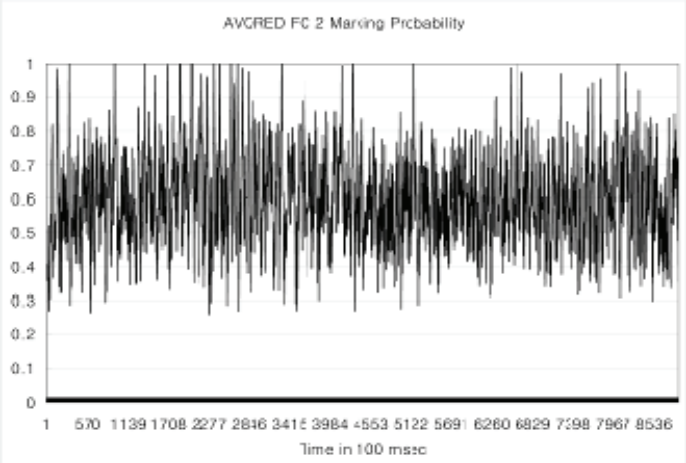

Figure 6.26: AVQRED FC 2 marking probability

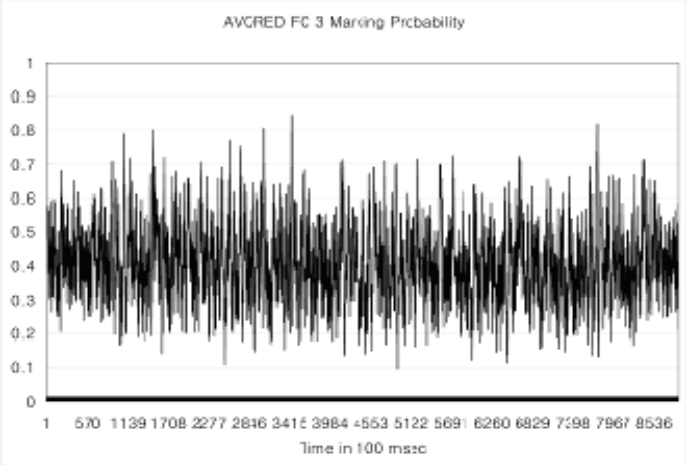

Figure 6.27: AVQRED FC 3 marking probability

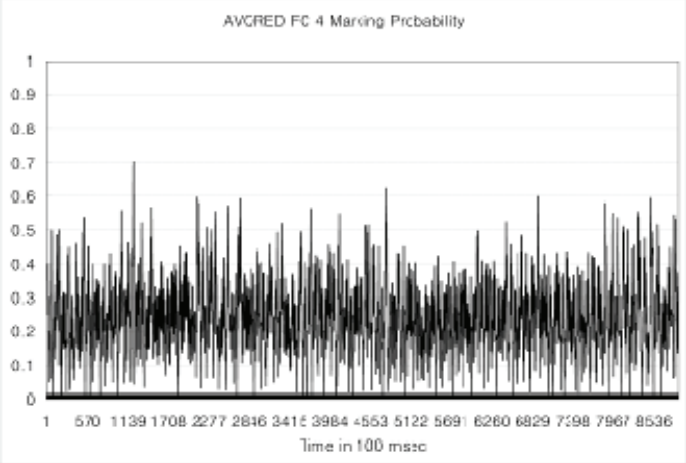

Figure 6.28: AVQRED FC 4 marking probability

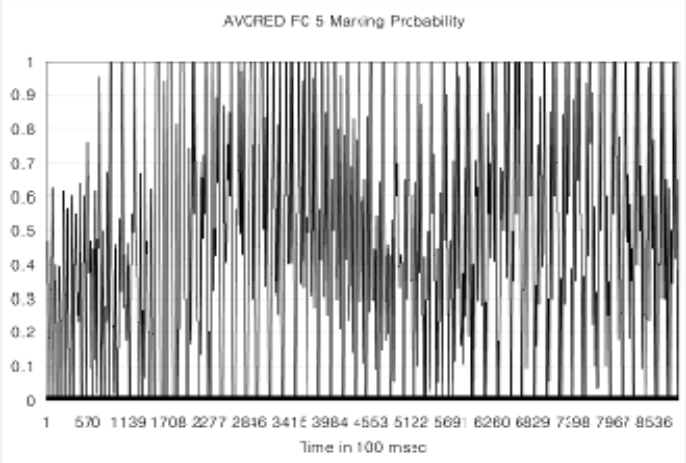

Figure 6.29: AVQRED FC 5 marking probability

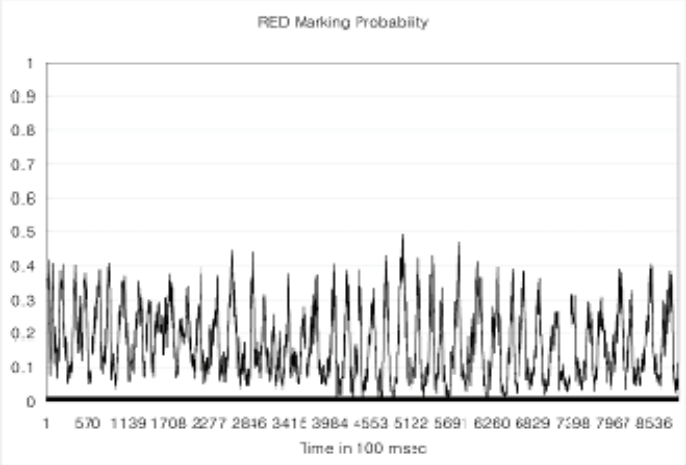

Figure 6.30: RED marking probability

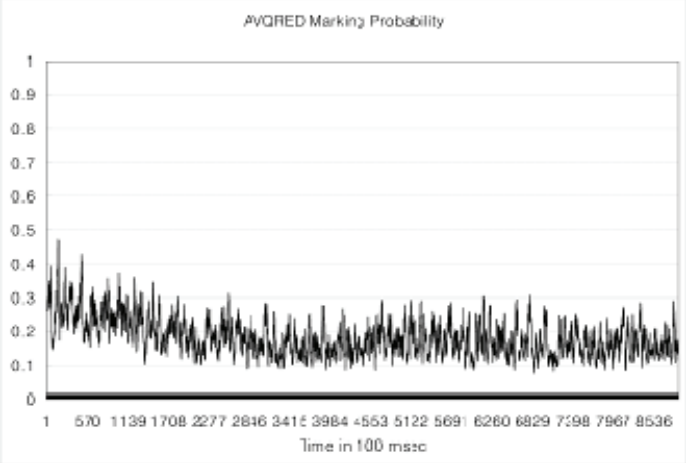

Figure 6.31: AVQRED marking probability

In summary, the emulation results confirmed that the marking probability of the buffer flow control AQMs is oscillatory and is higher than the packet drop AQMs. Unstable and high marking probability implies that the senders are not flow controlled effectively via buffer adjustments at the gateway.

6.4. Mathematical Model

The model used in section 4.4 is slightly modified to illustrate the buffer flow control instability problem mathematically. The differences between these two models are:

1. Model for the buffer flow control problem accepts packets instead of dropping them.
2. Model for the buffer flow control problem has new connections with full TCP receive windows.

$$\frac{dn}{dt} = \upsilon(t) \cdot m \cdot MaxRcvWin - n(t) \tag{6.1}$$

$$\frac{d\lambda}{dt} = (1 - p(t)) \times \frac{(1-\upsilon(t)) \cdot m}{R^2} - p(t) \times decr \times \frac{\lambda^2}{(1-\upsilon(t)) \cdot m} + \frac{dn}{dt} \tag{6.2}$$

$$\frac{dq}{dt} = -\mu + \omega(t - d(t)) \times \lambda(t - d(t)) \tag{6.3}$$

$$\frac{dv}{dt} = -\tau \times \mu + \omega(t) \times \lambda(t) \tag{6.4}$$

$$\frac{dw}{dt} = \frac{\log(1-B)}{\delta} \cdot w(t) - \frac{\log(1-B)}{\delta} \cdot q(t) \tag{6.5}$$

$$\frac{dp}{dt} = \begin{cases} -1.0 \times p(t) & \text{if } w(\text{or } v) < \min_{th} \\ \dfrac{p_{\max}}{(\max_{th} - \min_{th})} \times \dfrac{dw}{dt} & \text{if } \min_{th} \le w \le \max_{th} \text{ for RED} \\ \dfrac{p_{\max}}{(\max_{th} - \min_{th})} \times \dfrac{dv}{dt} & \text{if } \min_{th} \le v \le \max_{th} \text{ for AVQRED} \\ 1.0 - p(t) & \text{else} \end{cases} \tag{6.6}$$

(6.1) is the ODE of the arrival rate of new connections (that have passed the slow start phase) where *MaxRcvWin* is our configured maximum TCP receive window (16,000 bytes/10 to convert 1 second unit to 100 msec unit). *m* is the number of TCP connections, 500. υ(t) is the fourier series for new connections captured for 50 seconds during an actual emulation without AQM (shown in Figure 6.32). υ(t) can range from 0 to 1, and represent the proportion of the total TCP connections that are new.

(6.2) is the ODE of the arrival rate of the offered load which is similar to (4.1). The difference is that only the old enough connections, $(1- \upsilon(t))\cdot m$, participate in the TCP evolution, and the new connections always transmit at the full TCP receive window (by adding *dn/dt* to the equation). *decr* is the decrement factor that replaces the TCP halving factor, 1/2, in equation (4.1).

(6.3) is the ODE of the transmit queue size which is similar to (4.2). The difference is that the received packets are never dropped by removing the $(1- p(t-d(t)))$ term.

(6.4) is the ODE of the virtual queue size which is similar to (4.3). The difference is that the receive packets are never dropped by removing the $(1 - p(t))$ term.

(6.5) is the ODE of the weighted average transmit queue size which is the same as (4.4).

(6.6) is the ODE of the marking probability which is the same as (4.5).

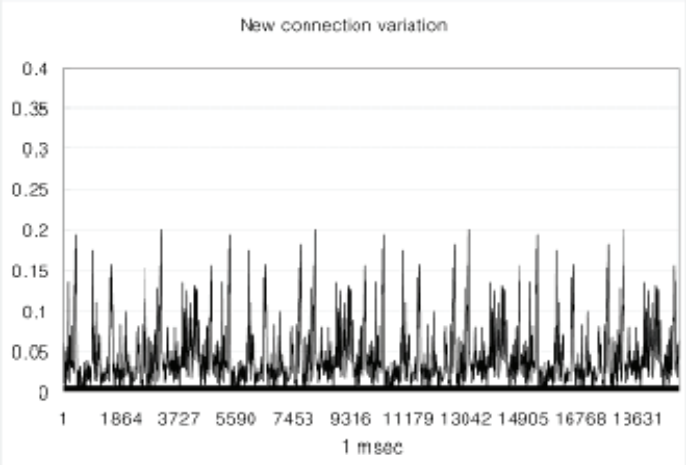

Figure 6.32: New connection variation

6.5. MATLAB Results

RED FC 2 and AVQRED FC 2 were fed to MATLAB, and the results are shown in this section. RED FC 2 and AVQRED FC 2 were selected as they are the most stable ones amongst the ones that have close to the full link utilization. MATLAB results for RED and AVQRED are not presented again because they are already presented in section 4.5.

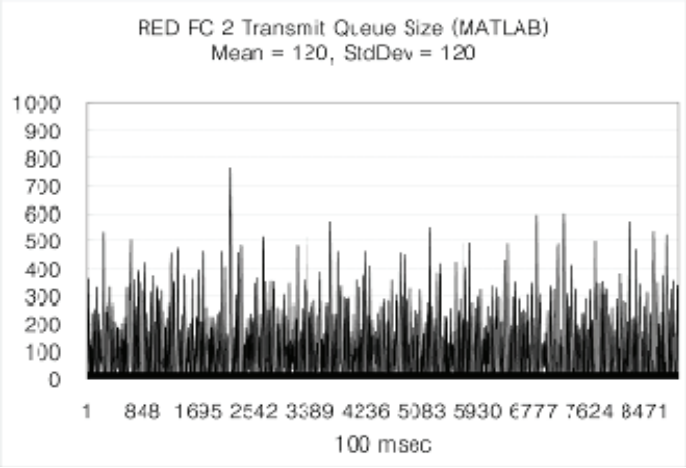

Figure 6.33: RED FC 2 transmit queue size (MATLAB)

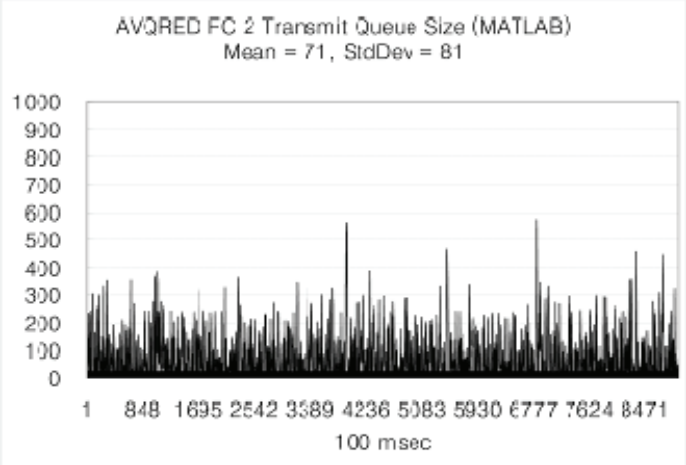

Figure 6.34: AVQRED FC 2 transmit queue size (MATLAB)

Compared to Figure 6.9, Figure 6.33 is little more oscillatory because the PEP buffering variation for the buffer flow control based AQMs is not the same as the fou-

rier series used in the model. During the experimentation, it was found that the amplitude and the period of PEP buffering variation are smaller with buffer flow control based AQMs. Analyzing the cause of this different behavior in PEP buffering variation is left for future study if necessary. However, it is important to note that the queueing behavior of Figure 6.34 resembles very closely to the behavior of Figure 6.14 because AVQRED is much less sensitive to PEP buffering as discussed in Chapter 4. Figure 6.35 and Figure 6.36 summarize the above results.

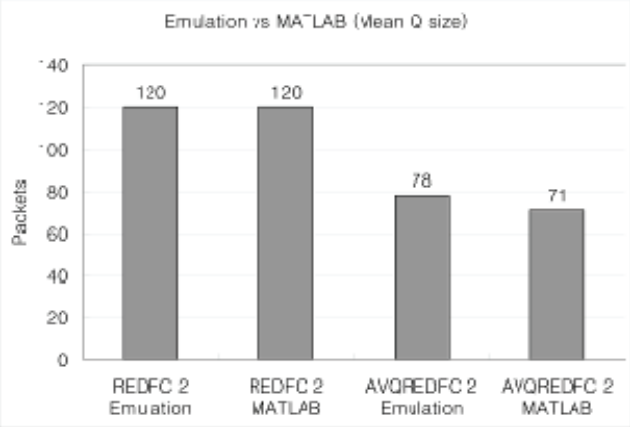

Figure 6.35: Emulation vs. MATLAB (Mean)

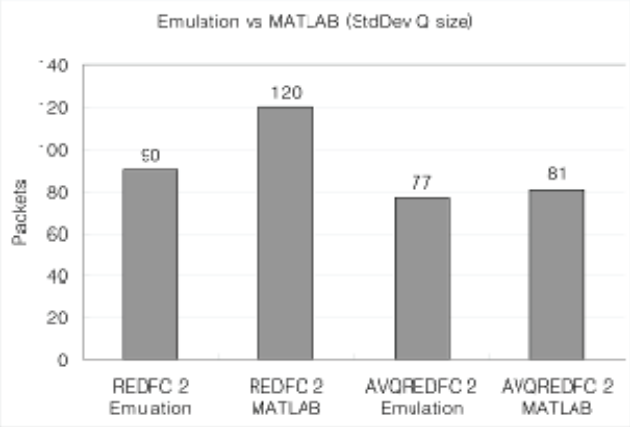

Figure 6.36: Emulation vs. MATLAB (Standard deviation)

To show how much of the instability is caused by the new connections and how much is caused by accepting packets during congestion, we used $\upsilon(t) = 0$ and re-ran MATLAB. The results are shown in Figure 6.37 and Figure 6.38. There are some improvements by not allowing new connections to burst in but the core of the problem still remains.

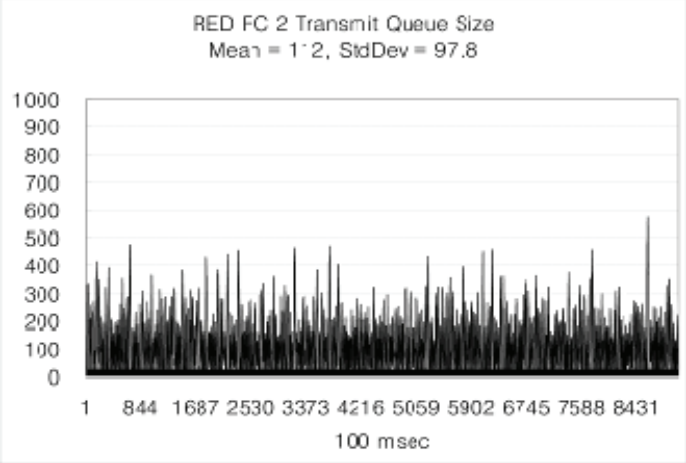

Figure 6.37: RED FC 2 transmit queue size without new connections

The purpose of this experimentation is not to generalize how much of the improvement we get by not allowing the new connection bursts, but to show which of the two (not dropping packets or allowing bursty new connections) is the main cause of the queueing instability. Figure 6.39 summarizes the improvement by not allowing bursty new connections. It is clear that accepting marked packets is more responsible for the queueing instability problem.

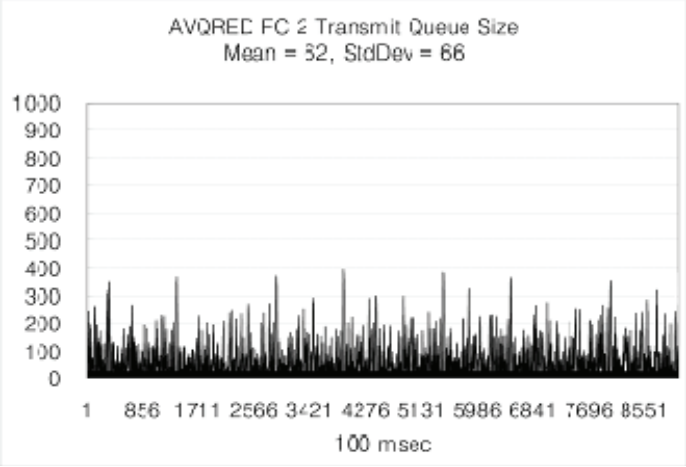

Figure 6.38: AVQRED FC 2 transmit queue size without new connections

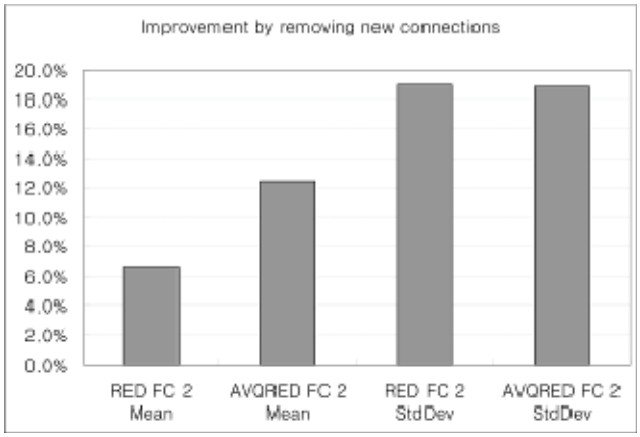

Figure 6.39: Queueing stability Improvement by removing new connections

6.6. Summary

We have applied two classes of AQM to satellite networks. One is buffer flow control AQM and the other is packet drop AQM. Buffer flow control adjusts the senders' transmission rate by adjusting the receive buffer limit. This method is quite applicable and attractive to satellite networks due to their inevitable PEP buffers. Microscopic examinations of buffer flow control predicted that it would be ineffective in flow con-

trolling the senders by allowing packets to enter the congested queue and by allowing bursts from new connections. Not being able to flow control the senders effectively would then result in queueing instability.

AVQRED with packet drop marking was proposed as the solution, and both the problem and the solution were validated with emulations and a mathematical model. The MATLAB results further revealed that allowing packets to enter the congested queue plays the major role in the instability problem.

Chapter 7

Conclusions

In this dissertation, we have examined utilizing existing active queue management (AQM) methods in satellite communication networks, found problems with the existing AQM methods, and provided the solutions for the problems. In these investigations, we have discovered three problems which we analyzed and resolved. The first problem is queueing instability (asynchronous queueing) due to high PEP buffering and global synchronization from virtual-queue based AQMs. The second problem is unfair bandwidth sharing. The third problem is queueing instability caused by allowing packets to enter the congested queue and by allowing new connections to burst into the queue.

First, we have looked at 8 existing AQM methods, selected 3 that are suitable for our experimentations and provided the reasons for the selections. The AQMs that were selected are RED, SRED and AVQ.

Secondly, we have constructed a realistic emulation environment with the actual gateway software used in *Hughes Network Systems*' *HughesNet*® networks and a traffic generator called *Spirent*. To add the satellite delays, we have created a bi-directional delay simulator software between the gateway and simulated remote terminals. 400 HTTP connections were emulated, and link utilization, queue size and packet drop were collected as the performance metrics. 100 HTTP connections and 3

UDP connections were added, and per-flow throughputs were collected in addition to the three metrics for the fair bandwidth sharing problem.

Third, we have looked at the first problem in depth: asynchronous queueing and global synchronization. Asynchronous queueing occurs when a real-queue-based AQM is used due to its inevitable high delays between the monitoring queue and the marking queue. Global synchronization occurs when a virtual-queue-based AQM is used due to its tail-drop nature. We have proposed the solution, AVQRED, to fix both asynchronous queueing and global synchronization. AVQRED essentially moves the monitoring queue to the marking queue by creating a virtual queue and avoids global synchronization by adapting the RED algorithm. We have constructed a mathematical model to provide intuitive illustrations of the problem and the solution. Both emulation and MATLAB results confirmed the problem and the solution.

Fourth, we have looked at the second problem in depth: fair bandwidth sharing. The fairness problem arises because existing AQMs do not have an accurate way of measuring fairness in sharing. The reason for just approximating the fair share is to avoid complexity added to the AQM algorithms. Our study found that the complexity does not change much when a fully per-flow aware algorithm is implemented except that the memory usage becomes higher. However, few extra kilo-bytes of memory usage are not an issue with gateways these days especially when the return is significant. We have enhanced AVQRED to be fully per-flow aware (called PFAVQRED) to solve the fairness problem. RED, SRED, AVQ, AVQRED and PFAVQRED were emulated with 500 HTTP and 3 UDP flows. The emulation results showed that

PFAVQRED provides close to the optimal bandwidth share to each flow while the per-flow bandwidth usage of the other AQMs varies at least twice as high.

Fifth, we have looked at the third problem in depth: queueing instability with buffer flow control. The problem is caused by allowing packets to enter the congested queue and by allowing new connections to burst into the queue. We have proposed AVQRED with packet drop marking as the solution. We have constructed a mathematical model to provide intuitive illustrations of the problem and the solution. Both emulation and MATLAB results confirmed the problem and the solution.

In an effort to improve the gateway performance of satellite networks, we have discovered three problems in existing AQMs and found the solutions for them. The problems and solutions are based on satellite networks where congested gateways reside on the ground station. However, if the gateways are placed on the satellite, the nature of congestion control changes significantly because there is another layer of delays (between the satellite and the ground station) to synchronize. Because mesh satellite networks most likely demand such a configuration, our future research will focus on solving problems in congestion control caused by the delays between the satellite and the ground station in mesh satellite networks.

Bibliography

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

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

[3] K. Ramakrishnan and S. Floyd, "A proposal to add explicit congestion notification (ECN) to IP," RFC 2481, Jan. 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, "AVQRED in satellite networks," to appear in *Wireless Technology: Applications, Management, and Security*, Springer, 2008.

[8] 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.

[9] D. Byun and J. Baras, "Per-flow adaptive virtual queue random early detection in satellite networks," submitted to *International Journal of Satellite Communications and Networking*, April, 2007.

[10] 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.

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

[12] 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.

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

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

[15] X. Lin, X. Chang and J. Muppala, "VQ-RED: An efficient virtual queue management approach to improve fairness in infrastructure WLAN," in *Proceedings of IEEE Local Computer Networks*, November 2005.

[16] Y. Shang, M. Hadjitheodosiou and J. Baras, "Flow control and active queue management for integrated services in an aeronautical satellite network," in *Proceedings of 24th American Institute of Aeronautics and Astronautics*, June 2006.

[17] W. Feng, D. Kandlur, D. Saha and K. Shin, "Blue: A new class of active queue management algorithms," Tech. Report, UM CSE-TR-387-99, 1999.

[18] C. Hollot, V. Misra, D. Towsley and W. Gong, "On designing improved controllers for AQM routers supporting TCP flows," in *Proceedings of IEEE/INFOCOM*, April 2001.

[19] 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.

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

[21] V. Jacobson, "Congestion Avoidance and Control," Computer Communication Review, vol. 18, no. 4, pp. 314-329, Aug. 1988.

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

[23] 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.

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

[25] 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.

[26] D. Byun and J. Baras, "Buffer flow control in satellite networks," submitted to *International Journal of Satellite Communications and Networking*, May, 2007.