Internet Engineering Task Force (IETF)
Request for Comments: 8034
Category: Informational
ISSN: 2070-1721
February 2017

G. White
CableLabs
R. Pan
Cisco Systems

Active Queue Management (AQM) Based on Proportional Integral Controller Enhanced (PIE) for Data-Over-Cable Service Interface Specifications (DOCSIS) Cable Modems

Abstract

Cable modems based on Data-Over-Cable Service Interface Specifications (DOCSIS) provide broadband Internet access to over one hundred million users worldwide. In some cases, the cable modem connection is the bottleneck (lowest speed) link between the customer and the Internet. As a result, the impact of buffering and bufferbloat in the cable modem can have a significant effect on user experience. The CableLabs DOCSIS 3.1 specification introduces requirements for cable modems to support an Active Queue Management (AQM) algorithm that is intended to alleviate the impact that buffering has on latency-sensitive traffic, while preserving bulk throughput performance. In addition, the CableLabs DOCSIS 3.0 specifications have also been amended to contain similar requirements. This document describes the requirements on AQM that apply to DOCSIS equipment, including a description of the "DOCSIS-PIE" algorithm that is required on DOCSIS 3.1 cable modems.

Status of This Memo

Table of Contents

1. Introduction

A recent resurgence of interest in active queue management, arising from a recognition of the inadequacies of drop-tail queuing in the presence of loss-based congestion control algorithms, has resulted in the development of new algorithms that appear to provide very good congestion feedback to current TCP algorithms, while also having operational simplicity and low complexity. One of these algorithms has been selected as a requirement for cable modems built according to the DOCSIS 3.1 specification [DOCSIS_3.1]. The Data-Over-Cable Service Interface Specifications (DOCSIS) define the broadband technology deployed worldwide for Ethernet and IP service over hybrid fiber-coaxial cable systems. The most recent revision of the DOCSIS technology, version 3.1, was originally published in October 2013 and provides support for up to 10 Gbps downstream (toward the customer) and 1 Gbps upstream (from the customer) capacity over existing cable networks. Previous versions of the DOCSIS technology did not contain requirements for AQM. This document outlines the high-level AQM requirements for DOCSIS systems, discusses some of the salient features of the DOCSIS Media Access Control (MAC) layer, and describes the DOCSIS-PIE algorithm -- largely by comparing it to its progenitor, the PIE algorithm [RFC8033].

2. Overview of DOCSIS AQM Requirements

CableLabs' DOCSIS 3.1 specification [DOCSIS_3.1] mandates that cable modems implement a specific variant of the Proportional Integral controller Enhanced (PIE) AQM algorithm [RFC8033]. This specific variant is provided for reference in Appendix A, and simulation results comparing it to drop-tail queuing and other AQM options are given in [CommMag] and [DOCSIS-AQM]. In addition, CableLabs' DOCSIS 3.0 specification [DOCSIS_3.0] has been amended to recommend that cable modems implement the same algorithm. Both specifications allow that cable modems can optionally implement additional algorithms that can then be selected for use by the operator via the modem's configuration file.

These requirements on the cable modem apply to upstream transmissions (i.e., from the customer to the Internet).

Both specifications also include requirements (mandatory in DOCSIS 3.1 and recommended in DOCSIS 3.0) that the Cable Modem Termination System (CMTS) implement AQM for downstream traffic; however, no specific algorithm is defined for downstream use.

3. The DOCSIS MAC Layer and Service Flows

The DOCSIS Media Access Control (sub-)layer provides tools for configuring differentiated Quality of Service (QoS) for different applications by the use of Packet Classifiers and Service Flows.

Each Service Flow has an associated QoS parameter set that defines the treatment of the packets that traverse the Service Flow. These parameters include, for example, Minimum Reserved Traffic Rate, Maximum Sustained Traffic Rate, Peak Traffic Rate, Maximum Traffic Burst, and Traffic Priority. Each upstream Service Flow corresponds to a queue in the cable modem, and each downstream Service Flow corresponds to a queue in the CMTS. The DOCSIS AQM requirements mandate that the CM and CMTS implement the AQM algorithm (and allow it to be disabled, if needed) on each Service Flow queue independently.

Packet Classifiers can match packets based upon several fields in the packet/frame headers including the Ethernet header, IP header, and TCP/UDP header. Matched packets are then queued in the associated Service Flow queue.

Each cable modem can be configured with multiple Packet Classifiers and Service Flows. The maximum number of such entities that a cable modem supports is an implementation decision for the manufacturer, but modems typically support 16 or 32 upstream Service Flows and at least that many Packet Classifiers. Similarly, the CMTS supports multiple downstream Service Flows and multiple Packet Classifiers per cable modem.

It is typical that upstream and downstream Service Flows used for broadband Internet access are configured with a Maximum Sustained Traffic Rate. This QoS parameter rate-shapes the traffic onto the DOCSIS link and is the main parameter that defines the service offering. Additionally, it is common that upstream and downstream Service Flows are configured with a Maximum Traffic Burst and a Peak Traffic Rate. These parameters allow the service to burst at a higher (sometimes significantly higher) rate than is defined in the Maximum Sustained Traffic Rate for the amount of bytes configured in Maximum Traffic Burst, as long as the long-term average data rate remains at or below the Maximum Sustained Traffic Rate.

Mathematically, what is enforced is that the traffic placed on the DOCSIS link in the time interval (t1,t2) complies with the following rate-shaping equations:

```
TxBytes(t1,t2) <= (t2-t1)*R/8 + B

TxBytes(t1,t2) <= (t2-t1)*P/8 + 1522
```

for all values t2>t1, where:

R = Maximum Sustained Traffic Rate (bps)

P = Peak Traffic Rate (bps)

B = Maximum Traffic Burst (bytes)

The result of this configuration is that the link rate available to the Service Flow varies based on the pattern of load. If the load that the Service Flow places on the link is less than the Maximum Sustained Traffic Rate, the Service Flow "earns" credit that it can then use (should the load increase) to burst at the Peak Traffic Rate. This dynamic is important since these rate changes (particularly the decrease in data rate once the traffic burst credit is exhausted) can induce a step increase in buffering latency.

4. DOCSIS-PIE vs. PIE

There are a number of differences between the version of the PIE algorithm that is mandated for cable modems in the DOCSIS specifications and the version described in [RFC8033]. These differences are described in the following subsections.

4.1. Latency Target

The latency target (a.k.a. delay reference) is a key parameter that affects, among other things, the trade-off in performance between latency-sensitive applications and bulk TCP applications. Via simulation studies, a value of 10 ms was identified as providing a good balance of performance. However, it is recognized that there may be service offerings for which this value doesn’t provide the best performance balance. As a result, this is provided as a configuration parameter that the operator can set independently on each upstream Service Flow. If not explicitly set by the operator, the modem will use 10 ms as the default value.

4.2. Departure Rate Estimation

The PIE algorithm utilizes a departure rate estimator to track fluctuations in the egress rate for the queue and to generate a smoothed estimate of this rate for use in the drop probability calculation. This estimator may be well suited to many link technologies but is not ideal for DOCSIS upstream links for a number of reasons.

First, the bursty nature of the upstream transmissions, in which the queue drains at line rate (up to ~100 Mbps for DOCSIS 3.0 and ~1 Gbps for DOCSIS 3.1) and then is blocked until the next transmit opportunity, results in the potential for inaccuracy in measurement, given that the PIE departure rate estimator starts each measurement during a transmission burst and ends each measurement during a (possibly different) transmission burst. For example, in the case where the start and end of measurement occur within a single burst, the PIE estimator will calculate the egress rate to be equal to the line rate, rather than the average rate available to the modem.

Second, the latency introduced by the DOCSIS request-grant mechanism can result in some further inaccuracy. In typical conditions, the request-grant mechanism can add between ~4 ms and ~8 ms of latency to the forwarding of upstream traffic. Within that range, the amount of additional latency that affects any individual data burst is effectively random, being influenced by the arrival time of the burst relative to the next request transmit opportunity, among other factors.

Third, in the significant majority of cases, the departure rate, while variable, is controlled by the modem itself via the pair of token bucket rate-shaping equations described in Section 3. Together, these two equations enforce a Maximum Sustained Traffic Rate, a Peak Traffic Rate, and a Maximum Traffic Burst size for the modem’s requested bandwidth. The implication of this is that the modem, in the significant majority of cases, will know precisely what the departure rate will be and can predict exactly when transitions between the Peak Traffic Rate and Maximum Sustained Traffic Rate will occur. Compare this to the PIE estimator, which would be simply reacting to (and smoothing its estimate of) those rate transitions after the fact.

Finally, since the modem is already implementing the dual-token bucket traffic shaper, it contains enough internal state to calculate predicted queuing delay with a minimum of computations. Furthermore, these computations only need to be run at every drop probability update interval, as opposed to the PIE estimator, which runs a similar number of computations on each packet dequeue event.

For these reasons, the DOCSIS-PIE algorithm utilizes the configuration and state of the dual-token bucket traffic shaper to translate queue depth into predicted queuing delay, rather than implementing the departure rate estimator defined in PIE.

4.3. Enhanced Burst Protection

The PIE algorithm [RFC8033] has two states: INACTIVE and ACTIVE. During the INACTIVE state, AQM packet drops are suppressed. The algorithm transitions to the ACTIVE state when the queue exceeds 1/3 of the buffer size. Upon transition to the ACTIVE state, PIE includes a burst protection feature in which the AQM packet drops are suppressed for the first 150 ms. Since DOCSIS-PIE is predominantly deployed on consumer broadband connections, a more sophisticated burst protection was developed to provide better performance in the presence of a single TCP session.

Where the PIE algorithm has two states, DOCSIS-PIE has three. The INACTIVE and ACTIVE states in DOCSIS-PIE are identical to those states in PIE. The QUIESCENT state is a transitional state between INACTIVE and ACTIVE. The DOCSIS-PIE algorithm transitions from INACTIVE to QUIESCENT when the queue exceeds 1/3 of the buffer size. In the QUIESCENT state, packet drops are immediately enabled, and upon the first packet drop, the algorithm transitions to the ACTIVE state (where drop probability is reset to zero for the 150 ms duration of the burst protection as in PIE). From the ACTIVE state, the algorithm transitions to QUIESCENT if the drop probability has decayed to zero and the queuing latency has been less than half of the LATENCY_TARGET for two update intervals. The algorithm then fully resets to the INACTIVE state if this "quiet" condition exists for the duration of the BURST_RESET_TIMEOUT (1 second). One end result of the addition of the QUIESCENT state is that a single packet drop can occur relatively early on during an initial burst, whereas all drops would be suppressed for at least 150 ms of the burst duration in PIE. The other end result is that if traffic stops and then resumes within 1 second, DOCSIS-PIE can directly drop a single packet and then re-enter burst protection, whereas PIE would require that the buffer exceed 1/3 full.

4.4. Expanded Auto-Tuning Range

The PIE algorithm scales the Proportional and Integral coefficients based on the current drop probability. The DOCSIS-PIE algorithm extends this scaling to cover values of drop probability greater than 1, which can occur as a result of the drop probability scaling function described in Section 4.6. As an example, if a flood of non-responsive 64-byte packets were to arrive at a rate that is twice the departure rate, the DOCSIS-PIE steady-state condition would be to drop 50% of these packets, which implies that drop probability would have the value of 8.00.

4.5. Trigger for Exponential Decay

The PIE algorithm includes a mechanism by which the drop probability is allowed to decay exponentially (rather than linearly) when it is detected that the buffer is empty. In the DOCSIS case, recently arrived packets may reside in the buffer due to the request-grant latency even if the link is effectively idle. As a result, the buffer may not be identically empty in the situations for which the exponential decay is intended. To compensate for this, we trigger exponential decay when the buffer occupancy is less than 5 ms * Peak Traffic Rate.

4.6. Drop Probability Scaling

The DOCSIS-PIE algorithm scales the calculated drop probability based on the ratio of the packet size to a constant value of 1024 bytes (representing approximate average packet size). While [RFC7567] in general recommends against this type of scaling, we note that DOCSIS-PIE is expected to be used predominantly to manage upstream queues in residential broadband deployments, where we believe the benefits outweigh the disadvantages. As a safeguard to prevent a flood of small packets from starving flows that use larger packets, DOCSIS-PIE limits the scaled probability to a defined maximum value of 0.85.

4.7. Support for Explicit Congestion Notification

DOCSIS-PIE does not include support for Explicit Congestion Notification (ECN). Cable modems are essentially IEEE 802.1d Ethernet bridges and so are not designed to modify IP header fields. Additionally, the packet-processing pipeline in a cable modem is commonly implemented in hardware. As a result, introducing support for ECN would engender a significant redesign of cable modem data path hardware, and would be difficult or impossible to modify in the future. At the time of the development of DOCSIS-PIE, which coincided with the development of modem chip designs, the benefits of ECN marking relative to packet drop were considered to be relatively minor; there was considerable discussion about differential treatment of ECN-capable packets in the AQM drop/mark decision, and there were some initial suggestions that a new ECN approach was needed. Due to this uncertainty, we chose not to include support for ECN.

5. Implementation Guidance

The AQM space is an evolving one, and it is expected that continued research in this field may result in improved algorithms in the future.

As part of defining the DOCSIS-PIE algorithm, we split the pseudocode definition into two components: a "data path" component and a "control path" component. The control path component contains the packet drop probability update functionality, whereas the data path component contains the per-packet operations, including the drop decision logic.

It is understood that some aspects of the cable modem implementation may be done in hardware, particularly functions that handle packet processing.

While the DOCSIS specifications don’t mandate the internal implementation details of the cable modem, modem implementers are strongly advised against implementing the control path functionality in hardware. The intent of this advice is to retain the possibility that future improvements in AQM algorithms can be accommodated via software updates to deployed devices.

6. Security Considerations

This document describes an active queue management algorithm based on [RFC8033] for implementation in DOCSIS cable modem devices. This algorithm introduces no specific security exposures.

7. References

7.1. Normative References

[RFC8033] Pan, R., Natarajan, P., Baker, F., and G. White, "Proportional Integral Controller Enhanced (PIE): A Lightweight Control Scheme to Address the Bufferbloat Problem", RFC 8033, DOI 10.17487/RFC8033, February 2017, <http://www.rfc-editor.org/info/rfc8033>.

7.2. Informative References

[CommMag] White, G., "Active queue management in DOCSIS 3.1 networks", IEEE Communications Magazine vol. 53, no. 3, pp. 126-132, DOI 10.1109/MCOM.2015.7060493, March 2015.

[DOCSIS-AQM]
White, G., "Active Queue Management in DOCSIS 3.x Cable Modems", May 2014, <http://www.cablelabs.com/wp-content/uploads/2014/06/DOCSIS-AQM_May2014.pdf>.

[DOCSIS_3.0]
CableLabs, "MAC and Upper Layer Protocols Interface Specification", DOCSIS 3.0, January 2017, <https://apps.cablelabs.com/specification/CM-SP-MULPIv3.0>.

[DOCSIS_3.1]
CableLabs, "MAC and Upper Layer Protocols Interface Specification", DOCSIS 3.1, January 2017, <https://apps.cablelabs.com/specification/CM-SP-MULPIv3.1>.

[RFC7567] Baker, F., Ed. and G. Fairhurst, Ed., "IETF Recommendations Regarding Active Queue Management", BCP 197, RFC 7567, DOI 10.17487/RFC7567, July 2015, <http://www.rfc-editor.org/info/rfc7567>.

Appendix A. DOCSIS-PIE Algorithm Definition

PIE defines two functions organized here into two design blocks:

1. Control path block -- a periodically running algorithm that calculates a drop probability based on the estimated queuing latency and queuing latency trend.

2. Data path block, a function that occurs on each packet enqueue that implements a per-packet drop decision based on the drop probability.

It is desirable to have the ability to update the control path block based on operational experience with PIE deployments.

A.1. DOCSIS-PIE AQM Constants and Variables

A.1.1. Configuration Parameters

- o LATENCY_TARGET. AQM Latency Target for this Service Flow

- o PEAK_RATE. Service Flow configured Peak Traffic Rate, expressed in bytes/second

- o MSR. Service Flow configured Maximum Sustained Traffic Rate, expressed in bytes/second

- o BUFFER_SIZE. The size (in bytes) of the buffer for this Service Flow

A.1.2. Constant Values

- o A = 0.25, B = 2.5. Weights in the drop probability calculation

- o INTERVAL = 16 ms. Update interval for drop probability

- o BURST_RESET_TIMEOUT = 1 second

- o MAX_BURST = 142 ms (150 ms - 8 ms (update error))

- o MEAN_PKTSIZE = 1024 bytes

- o MIN_PKTSIZE = 64 bytes

- o PROB_LOW = 0.85

- o PROB_HIGH = 8.5

- LATENCY_LOW = 5 ms
- LATENCY_HIGH = 200 ms

A.1.3. Variables

- drop_prob_. The current packet drop probability
- accu_prob_. Accumulated drop probability since last drop
- qdelay_old_. The previous queue delay estimate
- burst_allowance_. Countdown for burst protection, initialize to 0
- burst_reset_. Counter to reset burst
- aqm_state_. AQM activity state encoding 3 states:

  INACTIVE - Queue staying below 1/3 full, suppress AQM drops

  QUIESCENT - Transition state

  ACTIVE - Normal AQM drops (after burst protection period)
- queue_. Holds the pending packets

A.1.4. Public/System Functions

- drop(packet). Drops/discards a packet
- random(). Returns a uniform random value in the range 0 ~ 1
- queue_.is_full(). Returns true if queue_ is full
- queue_.byte_length(). Returns current queue_ length in bytes, including all MAC PDU bytes without DOCSIS MAC overhead
- queue_.enque(packet). Adds packet to tail of queue_
- msrtokens(). Returns current token credits (in bytes) from the Maximum Sustained Traffic Rate token bucket
- packet.size(). Returns size of packet

A.2. DOCSIS-PIE AQM Control Path

The DOCSIS-PIE control path performs the following:

o Calls control_path_init() at Service Flow creation

o Calls calculate_drop_prob() at a regular INTERVAL (16 ms)

```
================

//  Initialization function
control_path_init() {
    drop_prob_ = 0;
    qdelay_old_ = 0;
    burst_reset_ = 0;
    aqm_state_ = INACTIVE;
}

//  Background update, occurs every INTERVAL
calculate_drop_prob() {

    if (queue_.byte_length() <= msrtokens()) {
        qdelay = queue_.byte_length() / PEAK_RATE;
    } else {
        qdelay = ((queue_.byte_length() - msrtokens()) / MSR \
                  +  msrtokens() / PEAK_RATE);
    }

    if (burst_allowance_ > 0) {
        drop_prob_ = 0;
        burst_allowance_ = max(0, burst_allowance_ - INTERVAL);
    } else {

        p = A * (qdelay - LATENCY_TARGET) + \
            B * (qdelay - qdelay_old_);
        // Since A=0.25 & B=2.5, can be implemented
        // with shift and add

        if (drop_prob_ < 0.000001) {
            p /= 2048;
        } else if (drop_prob_ < 0.00001) {
            p /= 512;
        } else if (drop_prob_ < 0.0001) {
            p /= 128;
        } else if (drop_prob_ < 0.001) {
            p /= 32;
        } else if (drop_prob_ < 0.01) {
            p /= 8;
```

```
           } else if (drop_prob_ < 0.1) {
               p /= 2;
           } else if (drop_prob_ < 1) {
               p /= 0.5;
           } else if (drop_prob_ < 10) {
               p /= 0.125;
           } else {
               p /= 0.03125;
           }

           if ((drop_prob_ >= 0.1) && (p > 0.02)) {
               p = 0.02;
           }
           drop_prob_ += p;

           /* some special cases */
           if (qdelay < LATENCY_LOW && qdelay_old_ < LATENCY_LOW) {
               drop_prob_ *= 0.98;    // exponential decay
           } else if (qdelay > LATENCY_HIGH) {
               drop_prob_ += 0.02;   // ramp up quickly
           }

           drop_prob_ = max(0, drop_prob_);
           drop_prob_ = min(drop_prob_, \
                        PROB_LOW * MEAN_PKTSIZE/MIN_PKTSIZE);
       }

       // Check if all is quiet
       quiet = (qdelay < 0.5 * LATENCY_TARGET)
               && (qdelay_old_ < 0.5 * LATENCY_TARGET)
               && (drop_prob_ == 0)
               && (burst_allowance_ == 0);

       // Update AQM state based on quiet or !quiet
       if ((aqm_state_ == ACTIVE) && quiet) {
           aqm_state_ = QUIESCENT;
           burst_reset_ = 0;
       } else if (aqm_state_ == QUIESCENT) {
           if (quiet) {
               burst_reset_ += INTERVAL ;
               if (burst_reset_ > BURST_RESET_TIMEOUT) {
                   burst_reset_ = 0;
                   aqm_state_ = INACTIVE;
               }
           } else {
               burst_reset_ = 0;
           }
       }
       qdelay_old_ = qdelay;

   }
```

A.3. DOCSIS-PIE AQM Data Path

The DOCSIS-PIE data path performs the following:

- Calls enque() in response to an incoming packet from the CMCI

```
   ================
   enque(packet) {
       if (queue_.is_full()) {
           drop(packet);
           accu_prob_ = 0;
       } else if (drop_early(packet, queue_.byte_length())) {
           drop(packet);
       } else {
           queue_.enque(packet);
       }
   }

   ////////////////
   drop_early(packet, queue_length) {

       // if still in burst protection, suppress AQM drops
       if (burst_allowance_ > 0) {
           return FALSE;
       }

       // if drop_prob_ goes to zero, clear accu_prob_
       if (drop_prob_ == 0) {
           accu_prob_ = 0;
       }

       if (aqm_state_ == INACTIVE) {
           if (queue_.byte_length() < BUFFER_SIZE/3) {
               // if queue is still small, stay in
               // INACTIVE state and suppress AQM drops
               return FALSE;
           } else {
               // otherwise transition to QUIESCENT state
               aqm_state_ = QUIESCENT;
           }
       }
       //The CM can quantize packet.size to 64, 128, 256, 512, 768,
       // 1024, 1280, 1536, 2048 in the calculation below

       p1 = drop_prob_ * packet.size() / MEAN_PKTSIZE;
       p1 = min(p1, PROB_LOW);

       accu_prob_ += p1;

       // Suppress AQM drops in certain situations
       if ( (qdelay_old_ < 0.5 * LATENCY_TARGET && drop_prob_ < 0.2)
             || (queue_.byte_length() <= 2 * MEAN_PKTSIZE) ) {
           return FALSE;
       }

       if (accu_prob_ < PROB_LOW) {  // avoid dropping too fast due
            return FALSE;            // to bad luck of coin tosses...
       } else if (accu_prob_ >= PROB_HIGH) { // ...and avoid dropping
           drop = TRUE;                      // too slowly
       } else {                        //Random drop
           double u = random();        // 0 ~ 1
           if (u > p1)
              return FALSE;
           else
               drop = TRUE;
       }

       // At this point, drop == TRUE, so packet will be dropped.

       // Reset accu_prob_
       accu_prob_ = 0;

       // If in QUIESCENT state, packet drop triggers
       // ACTIVE state and start of burst protection
       if (aqm_state_ == QUIESCENT) {
           aqm_state_ = ACTIVE;
           burst_allowance_ = MAX_BURST;
       }
       return TRUE;
   }
```

Authors' Addresses

Greg White
CableLabs
858 Coal Creek Circle
Louisville, CO 80027-9750
United States of America

Email: g.white@cablelabs.com

Rong Pan
Cisco Systems
510 McCarthy Blvd
Milpitas, CA 95134
United States of America

Email: ropan@cisco.com