Internet Draft R. Pan
Network Working Group P. Natarajan, C. Piglione,M. Prabhu
Intended Status: Informational V. Subramanian, F. Baker, B. V. Steeg
Cisco Systems

Expires: June 2, 2013 December 10, 2012

PIE: A Lightweight Control Scheme To Address the Bufferbloat Problem

draft-pan-tsvwg-pie-00

Abstract

Bufferbloat is a phenomenon where excess buffers in the network cause high latency and jitter. As more and more interactive applications (e.g. voice over IP, real time video streaming and financial transactions) run in the Internet, high latency and jitter degrade application performance. There is a pressing need to design intelligent queue management schemes that can control latency and jitter; and hence provide desirable quality of service to users.

We present here a lightweight design, PIE(Proportional Integral controller Enhanced) that can effectively control the average queueing latency to a target value. Simulation results, theoretical analysis and Linux testbed results have shown that PIE can ensure low latency and achieve high link utilization under various congestion situations. The design does not require per-packet timestamp, so it incurs very small overhead and is simple enough to implement in both hardware and software.

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/1id-abstracts.html

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

Table of Contents

1. Introduction 3
2. Terminology 4
3. Design Goals 4
4. The PIE Scheme 5
4.1 Random Dropping 5
4.2 Drop Probability Calculation 6
4.3 Departure Rate Estimation 7
4.4 Handling Bursts 8
5. Comments and Discussions 9
6. Incremental Deployment 10
7. IANA Considerations 10
8. References 10
8.1 Normative References . . . . . .

8.2 Informative References 10
8.3 Other References 10
Authors' Addresses 11

1. Introduction

The explosion of smart phones, tablets and video traffic in the Internet brings about a unique set of challenges for congestion control. To avoid packet drops, many service providers or data center operators require vendors to put in as much buffer as possible. With rapid decrease in memory chip prices, these requests are easily accommodated to keep customers happy. However, the above solution of large buffer fails to take into account the nature of the TCP protocol, the dominant transport protocol running in the Internet. The TCP protocol continuously increases its sending rate and causes network buffers to fill up. TCP cuts its rate only when it receives a packet drop or mark that is interpreted as a congestion signal. However, drops and marks usually occur when network buffers are full or almost full. As a result, excess buffers, initially designed to avoid packet drops, would lead to highly elevated queueing latency and jitter. It is a delicate balancing act to design a queue management scheme that not only allows short-term burst to smoothly pass, but also controls the average latency when long-term congestion persists.

Active queue management (AQM) schemes, such as Random Early Discard (RED), have been around for well over a decade. AQM schemes could potentially solve the aforementioned problem. RFC 2309[RFC2309] strongly recommends the adoption of AQM schemes in the network to improve the performance of the Internet. RED is implemented in a wide variety of network devices, both in hardware and software. Unfortunately, due to the fact that RED needs careful tuning of its parameters for various network conditions, most network operators don't turn RED on. In addition, RED is designed to control the queue length which would affect delay implicitly. It does not control latency directly. Hence, the Internet today still lacks an effective design that can control buffer latency to improve the quality of experience to latency-sensitive applications.

Recently, a new AQM scheme, CoDel[CoDel], was proposed to control the latency directly to address the bufferbloat problem. CoDel requires per packet timestamps. Also, packets are dropped at the dequeue function after they have been enqueued for a while. Both of these requirements consume excessive processing and infrastructure resources. This consumption will make CoDel expensive to implement and operate, especially in hardware.

PIE aims to combine the benefits of both RED and CoDel: easy to implement like RED and directly control latency like CoDel. Similar to RED, PIE randomly drops a packet at the onset of the congestion. The congestion detection, however, is based on the queueing latency like CoDel instead of the queue length like RED. Furthermore, PIE

also uses the latency moving trends: latency increasing or decreasing, to help determine congestion levels. The design parameters of PIE are chosen via stability analysis. While these parameters can be fixed to work in various traffic conditions, they could be made self-tuning to optimize system performance.

In addition, we assume any delay-based AQM scheme would be applied over a Fair Queueing (FQ) structure or its approximate design, Class Based Queueing (CBQ). FQ is one of the most studied scheduling algorithms since it was first proposed in 1985 [RFC970]. CBQ has been a standard feature in most network devices today[CBQ]. These designs help flows/classes achieve max-min fairness and help mitigate bias against long flows with long round trip times(RTT). Any AQM scheme that is built on top of FQ or CBQ could benefit from these advantages. Furthermore, we believe that these advantages such as per flow/class fairness are orthogonal to the AQM design whose primary goal is to control latency for a given queue. For flows that are classified into the same class and put into the same queue, we need to ensure their latency is better controlled and their fairness is not worse than those under the standard DropTail or RED design.

This draft describes the overall design goals, system elements and implementation details of PIE. We will also discuss various design considerations, including how auto-tuning can be done.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

3. Design Goals

We explore a queue management framework where we aim to improve the performance of interactive and delay-sensitive applications. The design of our scheme follows a few basic criteria.

* First, we directly control queueing latency instead of controlling queue length. Queue sizes change with queue draining rates and various flows' round trip times. Delay bloat is the real issue that we need to address as it impairs real time applications. If latency can be controlled, bufferbloat is not an issue. As a matter of fact, we would allow more buffers for sporadic bursts as long as the latency is under control.

* Secondly, we aim to attain high link utilization. The goal of low latency shall be achieved without suffering link under-utilization or losing network efficiency. An early congestion signal could cause TCP to back off and avoid queue building up. On the other hand, however, TCP's rate reduction could result in link udner-utilization. There is a delicate balance between achieving high link utilization and low latency.

* Furthermore, the scheme should be simple to implement and easily scalable in both hardware and software. The wide adoption of RED over a variety of network devices is a testament to the power of simple random early dropping/marking. We strive to maintain similar design simplicity.

* Finally, the scheme should ensure system stability for various network topologies and scale well with arbitrary number streams. Design parameters shall be set automatically. Users only need to set performance-related parameters such as target queue delay, not design parameters.

In the following, we will elaborate on the design of PIE and its operation.

4. The PIE Scheme

As illustrated in Fig. 1, our scheme comprises three simple components: a) random dropping at enqueuing; b) periodic drop probability update; c) dequeuing rate estimation.

The following sections describe these components in further detail, and explain how they interact with each other. At the end of this section, we will discuss how the scheme can be easily augmented to precisely control bursts.

4.1 Random Dropping

Like any state-of-the-art AQM scheme, PIE would drop packets randomly according to a drop probability, p, that is obtained from the drop-probability-calculation component:

* upon a packet arrival

  randomly drop a packet with a probability p.

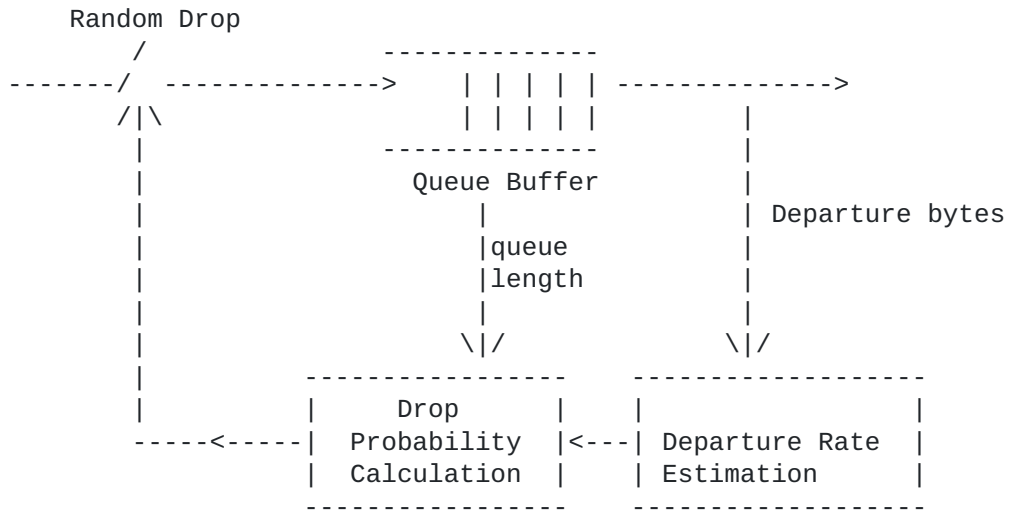

Figure 1. The PIE Structure

4.2 Drop Probability Calculation

The PIE algorithm periodically updates the drop probability as follows:

* estimate current queueing delay using Little's law:

```
est_del = qlen/depart_rate;
```

* calculate drop probability p as:

```
p = p + alpha*(est_del-target_del) + beta*(est_del-
est_del_old);

est_del_old = est_del.
```

Here, the current queue length is denoted by qlen. The draining rate of the queue, depart_rate, is obtained from the departure-rate-estimation block. Variables, est_del and est_del_old, represent the current and previous estimation of the queueing delay. The target latency value is expressed in target_del. The update interval is denoted as Tupdate.

Note that the calculation of drop probability is based not only on the current estimation of the queueing delay, but also on the direction

where the delay is moving, i.e., whether the delay is getting longer or shorter. This direction can simply be measured as the difference between
est_del and est_del_old. This is the classic Proportional Integral controller design that is adopted here for controlling queueing latency.
The controller parameters, in the unit of hz, are designed using feedback loop analysis where TCP's behaviors are modeled using the results from well-studied prior art[TCP-Models].

We would like to point out that this type of controller has been studied
before for controlling the queue length [PI, QCN]. PIE adopts the Proportional Integral controller for controlling delay and makes the scheme auto-tuning. The theoretical analysis of PIE is under paper submission and its reference will be included in this draft once it becomes available. Nonetheless, we will discuss the intuitions for these
parameters in Section 5.

4.3 Departure Rate Estimation

The draining rate of a queue in the network often varies either because other queues are sharing the same link, or the link capacity fluctuates.
Rate fluctuation is particularly common in wireless networks. Hence, we decide to measure the departure rate directly as follows.

* we are in a measurement cycle if we have enough data in the queue:

  qlen > deq_threshold

* if in a measurement cycle:

  upon a packet departure

  dq_count = dq_count + deque_pkt_size;

* if dq_count > deq_threshold then

  depart_rate = dq_count/(now-start);

  dq_count = 0;

  start = now;

We only measure the departure rate when there are sufficient data in the

buffer, i.e., when the queue length is over a certain threshold, dq_threshold. Short, non-persistent bursts of packets result in empty queues from time to time, this would make the measurement less accurate.
The parameter, dq_count, represents the number of bytes departed since the last measurement. Once dq_count is over a certain threshold, deq_threshold, we obtain a measurement sample. The threshold is recommended to be set to 10KB assuming a typical packet size of around 1KB or 1.5KB. This threshold would allow us a long enough period to obtain an average draining rate but also fast enough to reflect sudden changes in the draining rate. Note that this threshold is not crucial for the system's stability.

4.4 Handling Bursts

The above three components form the basis of the PIE algorithm. Although
we aim to control the average latency of a congested queue, the scheme should allow short term bursts to pass through the system without hurting them. We would like to discuss how PIE manages bursts in this section.

Bursts are well tolerated in the basic scheme for the following reasons:
first, the drop probability is updated periodically. Any short term burst that occurs within this period could pass through without incurring extra drops as it would not trigger a new drop probability calculation. Secondly, PIE's drop probability calculation is done incrementally. A single update would only lead to a small incremental change in the probability. So if it happens that a burst does occur at the exact instant that the probability is being calculated, the incremental nature of the calculation would ensure its impact is kept small.

Nonetheless, we would like to give users a precise control of the burst.
We introduce a parameter, max_burst, that is similar to the burst tolerance in the token bucket design. By default, the parameter is set to be 100ms. Users can certainly modify it according to their application scenarios. The burst allowance is added into the basic PIE design as follows:

* if p == 0 and est_del < del_ref and est_del_old < del_ref

    burst_allowance = max_burst;

* upon packet arrival

    if burst_allowance > 0 enqueue packet;

```
    * upon probability update

        burst_allowance = burst_allowance - Tupdate;
```

The burst allowance, noted by burst_allowance, is initialized to max_burst. As long as burst_allowance is above zero, an incoming packet will be enqueued bypassing the random drop process. During each update instance, the value of burst_allowance is decremented by the update period, Tupdate. When the congestion goes away, defined by us as p equals to 0 and both the current and previous samples of estimated delay
are less than target_del, we reset burst_allowance to max_burst.

5. Comments and Discussions

While the formal analysis will be included later, we would like to discuss the intuitions regarding how to determine the key parameters. Although the PIE algorithm would set them automatically, they are not meant to be magic numbers. We hope to give enough explanations here to help demystify them so that users can experiment and explore on their own.

As it is obvious from the above, the crucial equation in the PIE algorithm is

```
     p = p + alpha*(est_del-target_del) + beta*(est_del-est_del_old).
```

The value of alpha determines how the deviation of current latency from the target value affects the drop probability. The beta term exerts additional adjustments depending on whether the latency is trending up or down. Note that the drop probability is reached incrementally, not through a single step. To avoid big swings in adjustments which often leads to instability, we would like to tune p in small increments. Suppose that p is in the range of 1%. Then we would want the value of alpha and beta to be small enough, say 0.1%, adjustment in each step. If
p is in the higher range, say above 10%, then the situation would warrant a higher single step tuning, for example 1%. Finally, the drop probability would only be stabilized when the latency is stable, i.e. est_del equals est_del_old; and the value of the latency is equal to target_del. The relative weight between alpha and beta determines the final balance between latency offset and latency jitter.

The update interval, Tupdate, also plays a key role in stability. Given the same alpha and beta values, the faster the update is, the higher the
loop gain will be. As it is not showing explicitly in the above equation, it can become an oversight. Notice also that alpha and beta have a unit of hz.

As a further extension, we could introduce weights for flows that are classified into the same queue to achieve differential dropping. For example, the dropping probability for flow i could be p(i) = p/weight(i). Flows with higher weights would receive proportionally less
drops; and vice versa. Adding FQ on top, FQ_PIE, is another alternative.

Also, we have discussed congestion notification via the form of packet drops. The algorithm can be easily applied to networks codes where Early
Congestion Notification (ECN) is enabled. The drop probability, p, above
would become marking probability.

6. Incremental Deployment

One nice property of the AQM design is that it can be independently designed and operated without the requirement of being inter-operable.

Although all network nodes can not be changed altogether to adopt latency-based AQM schemes, we envision a gradual adoption which would eventually lead to end-to-end low latency service for real time applications.

7. IANA Considerations

There are no actions for IANA.

8. References

8.1 Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

8.2 Informative References

[RFC970] Nagle, J., "On Packet Switches With Infinite Storage",RFC970, December 1985.

8.3 Other References

[CoDel] Nichols, K., Jacobson, V., "Controlling Queue Delay", ACM
Queue. ACM Publishing. doi:10.1145/2209249.22W.09264.

[CBQ] Cisco White Paper, "http://www.cisco.com/en/US/docs/ 12_0t

/12_0tfeature/guide/cbwfq.html".

[TCP-Models] Misra, V., Gong, W., and Towsley, D., "Fluid-based Analysis of a Network of AQM Routers Supporting TCP Flows with an Application to RED", SIGCOMM 2000.

[PI] Hollot, C.V., Misra, V., Towsley, D. and Gong, W., "On Designing Improved Controller for AQM Routers Supporting TCP Flows", Infocom 2001.

[QCN] "Data Center Bridging - Congestion Notification", http://www.ieee802.org/1/pages/802.1au.html.

Authors' Addresses

Rong Pan
Cisco Systems
510 McCarthy Blvd,
Milpitas, CA 95134, USA
Email: ropan@cisco.com

Preethi Natarajan,
Cisco Systems
510 McCarthy Blvd,
Milpitas, CA 95134, USA
Email: prenatar@cisco.com

Chiara Piglione
Cisco Systems
510 McCarthy Blvd,
Milpitas, CA 95134, USA
Email: cpiglion@cisco.com

Mythili Prabhu
Cisco Systems
510 McCarthy Blvd,
Milpitas, CA 95134, USA
Email: mysuryan@cisco.com

Vijay Subramanian
Cisco Systems
510 McCarthy Blvd,
Milpitas, CA 95134, USA
Email: vijaynsu@cisco.com

Fred Baker
Cisco Systems

510 McCarthy Blvd,
Milpitas, CA 95134, USA
Email: fred@cisco.com

Bill Ver Steeg
Cisco Systems
5030 Sugarloaf Parkway
Lawrenceville, GA, 30044, USA
Email: versteb@cisco.com