Review: Proportional Integral controller Enhanced (PIE) Active Queue Management (AQM) [PNB+15]

Bob Briscoe*

09 May 2015

Main Concerns

The following summary of my concerns, can be used as a table of contents for the rest of the review:

*ietf@bobbriscoe.net, BT Research & Technology, B54/77, Adastral Park, Martlesham Heath, Ipswich, IP5 3RE, UK

My technical points run to 15 pages. The review ends with a 7 page tailpiece of editorial nits, references, etc.

The PIE draft ends with two assertions in the Discussion section:
“PIE is simple to implement”
“PIE does not require any user configuration”

I am afraid I have to say that I do not believe either statement is warranted any more. PIE has lost its way a bit. The implementation has not retained the elegance of the theory. The performance benefit from so-called ‘enhancements’ is questionable or non-existent, whereas the added complexity is very apparent. Also PIE now contains a large number of hard-coded constants (I counted 20) that ought to be scenario-dependent configuration variables.

1 Proposed Standard, but No Normative Language

See email thread “PIE (and CoDel) drafts: proposed standard vs informational?” (It seems like it would not be a big issue to make the PIE draft informational)

If normative language were included, we’d need to consider, “What is the essence of PIE?” My suggestions:

- MUST calculate drop probability as $p = p + \alpha(\dots) + \beta(\dots)$
- MAY `est_del = qlen/depart_rate`
  Alternative calculations are discussed:
  - solely at enqueue
  - using timestamps at dequeue
- MUST turn on & off
- MUST autotune α & β
- MUST handle bursts for drop, by default SHOULD NOT for ECN
- MAY derandomize, but only for per-flow queuing (see later)

And, in each case, what is the outcome that MUST be achieved, not just the RECOMMENDED formula for the the reference implementation?

2 Justify Each Enhancement

Each enhancement is (mostly) justified theoretically, which is good. But has PIE been separately tested with and without each enhancement to validate the theory? For instance:

- with and without derandomization?
- with different burst sizes?
- with and without autotuning α & β?

3 AQM Recommendations

Check whether PIE satisfies all the requirements in the Recommendations [BF15], and if it doesn’t comply with the SHOULDs, say why. I noticed a few specific “non-compliances”:

- Recommendation 4.2.1. “AQM and ECN”
- Recommendation 4.3 Autotuning
- Recommendation 4.5 Transport-Independent?

3.1 Recommendation 4.2.1. "AQM and ECN"

"Network devices SHOULD use an AQM algorithm that marks ECN-capable traffic"

In the PIE draft, there are three mentions of marking or dropping in general sections, but only dropping is described in the specific sections on the PIE algorithm.

The AQM Recommendations draft [BF15] also says that ECN parameters SHOULD be separately configurable, so the PIE draft ought to say which specific parameters would be separately configurable for ECN. IMO, at least burst size should be separately configurable for ECN (irrespective of whether ECN and non-ECN flows are queued separately or together).

3.2 Recommendation 4.3 Autotuning

"AQM algorithm deployment SHOULD NOT require operational tuning"

However, the PIE draft has not attempted this. Indeed, when the Introduction discusses PIE parameters (`target_del` and `max_burst`?), it only says could, not should:

```
"While these parameters can be fixed to work
 in various traffic conditions, they could be made self-tuning to
 optimize system performance.
"
```

3.3 Recommendation 4.5 Transport-Independent?

"AQM algorithms SHOULD NOT be dependent on specific transport protocol behaviours"

I believe the autotuning factors in the lookup table in the appendix of the IETF draft (`status_update()` block) are specific to TCP Reno. Whether they are or not, the draft needs to say how they were derived. And how they would be derived if conditions change, e.g. Cubic (or even rmcat!) becomes dominant instead of Reno.

Most importantly, we need to know whether these values can be hard-coded for ever, or whether they must be configurable.

These factors were originally derived using the theory in the HPSR'13 paper on PIE [PPP+13]. However, as shown in the table below, the factors in the pseudocode are different from the paper (which used Misra *et al*'s model of TCP Reno to derive them). The draft still says it is based on the same model of TCP Reno, so why are the factors different? If the analysis has been generalised since the HPSR'13 paper, we need to know what logic these new choices are based on.

p	factor in I-D [PNB+15]	factor in original HPSR paper [PPP+13]
$< 0.1\%$	1/128	
$< 1\%$	1/16	1/8
$< 10\%$	1/2	1/2
`else`	1	1

BTW, there's a mistake in the theoretical analysis section of the HPSR'13 paper on PIE [PPP+13], so the effect on the tuning factors in the draft will need to be checked:

CURRENT:

$$\kappa = \alpha R_o/(p_o T)$$

SHOULD BE:

$$\kappa = 2\alpha R_o/(p_o T)$$

BTW2, the control law in CoDel is also specific to TCP Reno.

4 Articulate the Rationale for a PI Controller

The draft doesn't actually say that a PI controller is chosen to keep the average latency constant even under high load. The 3rd para of Intro says that, unlike RED's use of queue length, PIE uses latency as the metric. However, that alone could just mean that the latency still increases with load.

All the rationale in section 4.2 doesn't state this fundamental reasoning either, except by reference, citing the original paper on PI [HMTG01].

5 Technical Flaws/Concerns

5.1 Turning PIE Off

My concern is about buffers that are more often completely idle than active (e.g. in most access networks). PIE sets itself to `inactive` when `qdelay` has been less than `QDELAY_REF` for long enough that `drop_prob` has reduced to zero.

But it seems that PIE will not reduce `drop_prob` unless there is at least a low rate of packet arrivals. I think PIE is blind to any period when there are no packets at all, and just picks up where it left off the next time a packet does arrive.

I am unsure, because it is not clear from the pseudocode where the `status_update()` routine is called from. Is it a parallel background process, or triggered by packet arrivals? At the start of `status_update()`, it tests whether `T_UPDATE` has been reached, which implies that it is called on each packet arrival (because this test would not be necessary if it were triggered by a timer interrupt).

If the timer check in `status_update` is meant to be triggered by packet arrivals, when traffic just stops at the end of user activity, PIE will stop updating `drop_prob`. Then when some time later the user becomes active again, PIE tests whether it should be in the active state by whether `drop_prob` is zero. But nothing has triggered updates to `drop_prob` in the meantime. So PIE will fool itself into thinking it is in the active state and pick up all the variables where it left off, perhaps hours ago.

If this is corrected by making `status_update()` into a timer-triggered process, then care will be needed to ensure it doesn't keep a machine awake that would otherwise go into an energy-saving sleep mode when it has been inactive for a while.

5.2 'Autotuning' α & β Parameters

The following lookup table is abbreviated from that in the `status_update()` code block:

```
if (p < 0.1%) {
     p += [alpha*(qdelay - QDELAY_REF) + beta*(qdelay-PIE->qdelay_old_)]/128;
} else if (p < 1%) {
     p += [alpha*(qdelay - QDELAY_REF) + beta*(qdelay-PIE->qdelay_old_)]/16;
} else if (p < 10%) {
     p += [alpha*(qdelay - QDELAY_REF) + beta*(qdelay-PIE->qdelay_old_)]/2;
} else {
     p +=  alpha*(qdelay - QDELAY_REF) + beta*(qdelay-PIE->qdelay_old_);
}
```

I am concerned that using a look-up table is not 'autotuning'. It does not tune beyond the bounds of the table. I.e. everything below 0.1% (the lowest value) will have to make do with division by 128 (see Fig 1). To properly cover the operational range of values of p would require far too many "`else ifs`" in sequence.

For instance a single Cubic flow with RTT of 100 ms bottlenecked in a link at the rates below causes the `drop_prob` shown.

6 Mb/s	0.1%	
12 Mb/s	0.04%	←the table in the draft stops about here!
24 Mb/s	0.016%	
48 Mb/s	0.0065%	
96 Mb/s	0.0025%	
192 Mb/s	0.0010%	
384 Mb/s	0.00040%	
768 Mb/s	0.00016%	

So the table in the PIE code only covers up to about 12 Mb/s, which might represent some upstream access links today, but it is missing nearly two orders of magnitude for even the normal residential downstream rates sold today (e.g. 80 Mb/s).

1 Gb/s access links are fairly normal today for private networks, and PIE should be applicable on aggregated links too, as well as in data centres where 10 Gb/s is becoming the norm.

Further, equipment implemented today will probably sit in the network for 10–20 years. Access link rates have typically doubled every 1.6 years. So, in 10–20 years, link rates might be 75×–6000× their rates today.

Anyway, that look-up table is surely unnecessary. Below I've repeated the look-up table of p against 'factor' from the draft and added more columns to show that the factors in the look-up table approximate to a linear relationship with p (compare p_{mid} and `factor`/16). This is not surprising because the draft says that the factors were chosen so that the changes in p were proportionate to the size of p before the change.

p	p_{mid}	p_{mid}	`factor`	`factor`/16
<0.1%	0.03%	1/3333	1/128	1/2048
<1%	0.3%	1/333	1/16	1/256
<10%	3%	1/33	1/2	1/32
`else`	30%	1/3	1	1/16

where:

p	:	`PIE->drop_prob`
p_{mid}	:	the (geometric) mid-points of each range, as a percentage and as a fraction
`factor`	:	The factor in the "`if else`" look-up table in the pseudocode, repeated at the start of this section.

So, why not just set

$$\texttt{factor} = p * \texttt{ALPHA\_DEFAULT}$$

with `ALPHA_DEFAULT` = 16? It's a decent approximation of the look-up table of factors in the draft, particularly given the draft says that the exact values of the factors are not critically sensitive. Importantly, it continues indefinitely as p gets smaller, whereas the look-up table approach always stops somewhere (see Fig 1).

Previously, each factor was chosen as a power of two, so multiplying it with the "$\alpha(...)+\beta(...)$") expression could be done with just adds and bit-shifts. Even though '`factor`' now seems to have to be a double, this multiplication can still be done with just adds and bit-shifts (see "Implementation Suggestions" later).

Note: My colleague Koen de Schepper reckons the factors in the pseudocode reduce too fast (see my earlier question about why the factors in the draft reduce faster than those in the HPSR'13 paper). If he's right, whether we use the factors in the draft or my approximation to them, PIE will be too sluggish at high bit-rates. If the draft authors agree that the factors should reduce in smaller steps, then the approach I propose above will not be applicable. However, there would be a different solution.

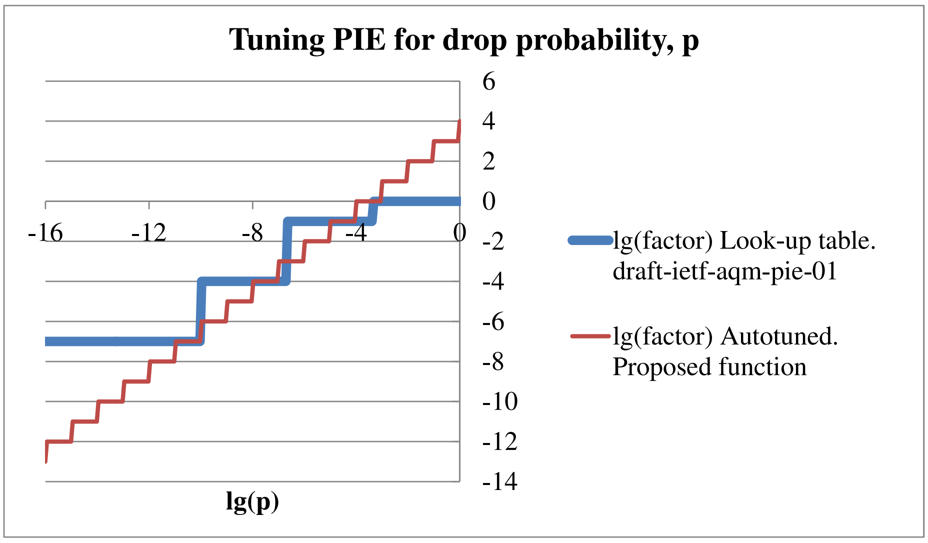


Figure 1: Comparison of Proposed Autotuning Function with Look-Up Table Approach

5.3 Averaging

5.3.1 Queue Sampling Rate

The queue is only sampled and p is only updated once every `T_UPDATE` (= 16ms by default). This might be appropriate for properly ACK-clocked or paced long-running flows. However, more bursty traffic is often the norm, with lots of short flows arriving at irregular intervals. In such bursty scenarios, the queue will tend to vary rapidly, so numerous samples will be needed to characterise the average queue. By spacing samples so widely apart, it seems likely to take a very long time for p to reach an appropriate value, by which time it will probably no longer be appropriate.

PIE's sampling rate might be appropriate to minimise processing on cost-reduced equipment like a residential gateway for a link rate of say 8 Mb/s (it then equates to about one in ten 1500 B packets). But, even then, it seems unnecessarily infrequent. The draft should explain the tradeoffs that were made before settling on this sampling rate, and whether evaluations have shown that a shorter sampling time (if feasible) would be beneficial?

The draft says "Given modern high speed links, this period translates into once every tens, hundreds or even thousands of packets." However, it seems that the benefit of reducing background processing beyond tens of packets becomes insignificant compared to the sloppiness of such infrequent updates to p.

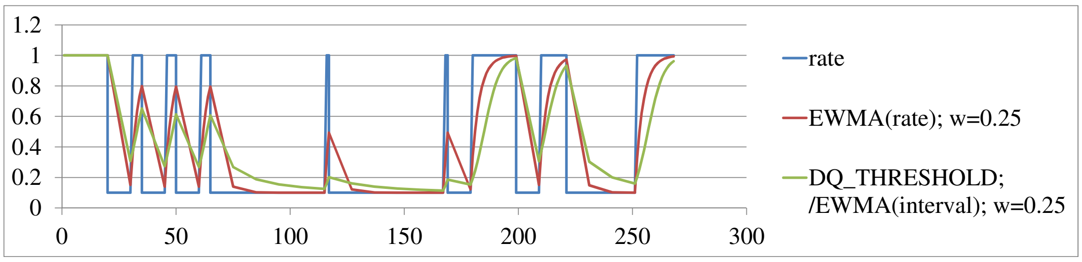
5.3.2 Averaging is Not Exponentially Weighted

In the `set_status()` block of the pseudocode (or Section 4.2 of the body of the draft), the queuing delay is determined as follows:

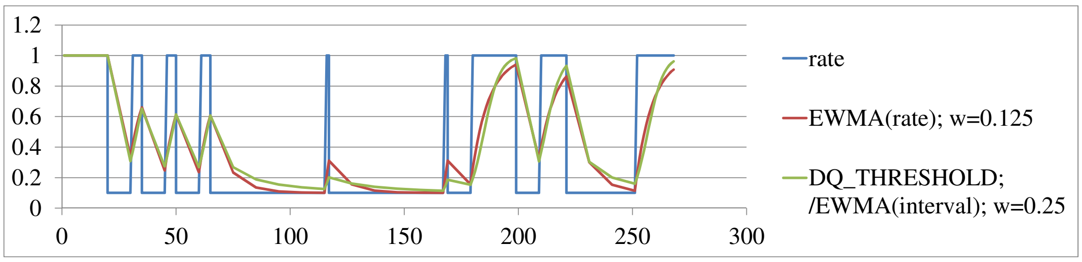
```
  qdelay = queue_.byte_length() * avg_dq_time_/DQ_THRESHOLD;
```

which is explained (in Section 4.2) as `qlen / depart_rate`, where

```
   depart_rate = DQ_THRESHOLD/avg_dq_time_
```



(a) Equal weights



(b) EWMA weight adjusted to attempt to match the non-EWMA

Figure 2: Exponentially Weighted Moving Average vs. PIE's Non-EWMA Approach for an Example Sequence of Rate Variations

However, taking an exponentially weighted moving average (EWMA) of the dequeue times in the denominator is not the same as taking the EWMA of the rates. The pseudocode continually measures the sequence of dequeue times, $t_1, t_2, t_3, \ldots$ that it takes to transmit constant amounts of bytes (`DQ_THRESHOLD`). Then the exponentially weighted moving average (EWMA) of the rate should be:

$$\begin{aligned}\mathrm{ewma}(\texttt{depart\_rate}) &= \texttt{DQ\_THRESHOLD} * \mathrm{ewma}(1/t_1, 1/t_2, 1/t_3, \ldots)\\ &\neq \texttt{DQ\_THRESHOLD}/\mathrm{ewma}(t_1, t_2, t_3, \ldots)\end{aligned}$$

Fig 2 compares a correctly calculated EWMA with the non-EWMA in the PIE draft. They are both shown averaging the departure rate of a link that varies between 1 unit and 0.1 units for the same sequence of example durations. For the record, the formula for the correct EWMA is:

$$\texttt{avg\_depart\_rate} \leftarrow [1 - (1-a)^{1/\texttt{dq\_time}}] * \texttt{dq\_time} + (1-a)^{1/\texttt{dq\_time}} * \texttt{avg\_depart\_rate},$$

where a is the weighting constant of the EWMA. This correct EWMA formula has solely been used to compare it with the PIE approach; it is not intended to be used in the PIE code, because it is too complex—more complex than a typical EWMA, because the times between each measurement event depend on the measurement.

The top chart (2a) shows that the PIE approach is considerably slower to converge than the correct EWMA with the same weight (1/4 in both cases). In the bottom chart (2b), the weight of the correct EWMA has been scaled down to 1/8 in an attempt to match the outcome of the PIE approach. Looking closely it can be seen that the PIE approach is faster than exponential at first, then slower than exponential later. This explains why, whatever constant scaling is chosen, even if one part of the trace can be made to match, other parts will fail to match.

It would be hard to implement a correct EWMA unless constant time periods were used instead of constant amounts of bytes. It might still be acceptable to use the PIE formula as an approximation. However, it should be considered whether there might be pathological scenarios where the error would be unacceptable.

(See also the comment at the end of §6 about how the EWMA constant should be set relative to `DQ_THRESHOLD`.)

5.4 Burst allowance unnecessary?

The recommended `BURST_ALLOWANCE` of 150 ms seems both very large and unnecessary. For typical interactions with CDNs placed about 10 ms away, this will suppress any response from PIE for about 15 round trips. PIE's 'silence period' occurs every time PIE starts after an application-limited or idle period, which is a very common occurrence in access links dedicated to single customers.

A burst allowance is intended to ensure that bursts that appear and disappear of their own accord do not get hit unnecessarily by a loss. Bursts that "disappear of their own accord" means bursts due to coincidences in arrivals from multiple flows. It does not mean bursts that TCP removes after it receives round trip feedback. The AQM is meant to respond faster than TCP, because TCP is meant to respond to the signals from the AQM. If the AQM responds more slowly than TCP's round trip time response, TCP will have to either drive up queuing delay or drive the queue to drop-tail in order to get a signal to respond to.

As the draft says, the PIE algorithm without a burst allowance already naturally filters its response to bursts, for two reasons: i) it misses bursts within the queue sampling period `T_UPDATE`; ii) its drop probability only evolves in small steps in response to changing queue length.

This seems the best way to filter bursts, because the larger and longer the burst, the less the response to it will be filtered out. Introducing a fixed period of 150 ms during which PIE is completely unable to send any signals, no matter how large the burst, seems very wrong, especially given PIE already has a good filtering mechanism that self-tunes to the traffic.

Given the draft recommends setting `BURST_ALLOWANCE` to 150 ms, it should at least point to experiments that explain why this choice gives good performance compared to: i) a smaller burst allowance; ii) no burst allowance; or iii) no burst allowance and a less aggressive (lower) value for `beta` (and perhaps also for `alpha`). This will at least allow operators to judge whether the experiments are appropriate to the deployment in question (e.g. appropriate round trip times and appropriate idle periods in the traffic).

5.5 Needs a Large Delay to Make the Delay Small

The departure rate measurement only works if the queue grows larger than 2^{14} B. For even a medium rate broadband uplink (8 Mb/s), this represents 16 ms delay. So, the queue has to reach 16 ms of delay for the algorithm to start working. And unless the target delay is set as high as 16 ms, PIE cannot maintain a queue large enough to update its own rate estimation. Therefore, it seems that the constraints of rate estimation set a lower bound on the target delay, which is not a healthy reason for choosing a target delay value.

For a 2 Mb/s link (the lower bound of the definition of broadband), PIE has to build a 66 ms queue in order to estimate its departure rate.

5.6 Derandomization

I've said this before on this list about the derandomization in RED, but perhaps it was not understood...

Unless there is only one flow in the queue (e.g. per-flow queuing), derandomizing the spacing between drops just wastes cycles. So, if PIE is in a shared FIFO queue, the derandomization routine might as well be called `heat_the_room()`.

There's a formal proof of this in my PhD dissertation [Bri09, §7.7.1], but here's the intuition. Imagine 2 equal but desynchronized flows A & B are feeding the queue. Even if PIE makes the spacing between drops regular, some drops will hit flow A and some will hit flow B. As long as the flows are not synchronized, PIE will not drop alternately from A,B,A,B,A,B, etc. The flow that is hit by each drop will be randomized. So the spacing between drops within each flow will no longer be regular, even though the spacing between drops in the whole aggregate is regular.

If this is still not obvious, here's a numerical example, with 4 desynchronized but equal flows A,B,C&D. For illustration, let's make the derandomization perfect and I will show that it becomes nearly random again when one considers the spacing in each flow rather than in the aggregate.

Let's say drop probability = 1% and derandomization is so perfect that it drops packet number 50, 150, 250, etc. Over such a distance between drops, it is reasonable to assume that there will be an equal chance of hitting any of the flows.

The drop at packet#50 hits flow A with chance 1 in 4, and misses it with chance 3 in 4.
The drop at packet#150 hits flow A with chance 1 in 4, and misses it with chance 3 in 4.
The drop at packet#250 hits flow A with chance 1 in 4, and misses it with chance 3 in 4.
etc.

Let's assume one of these drops has hit flow A at packet#X.
Then the chance of a drop from flow A:

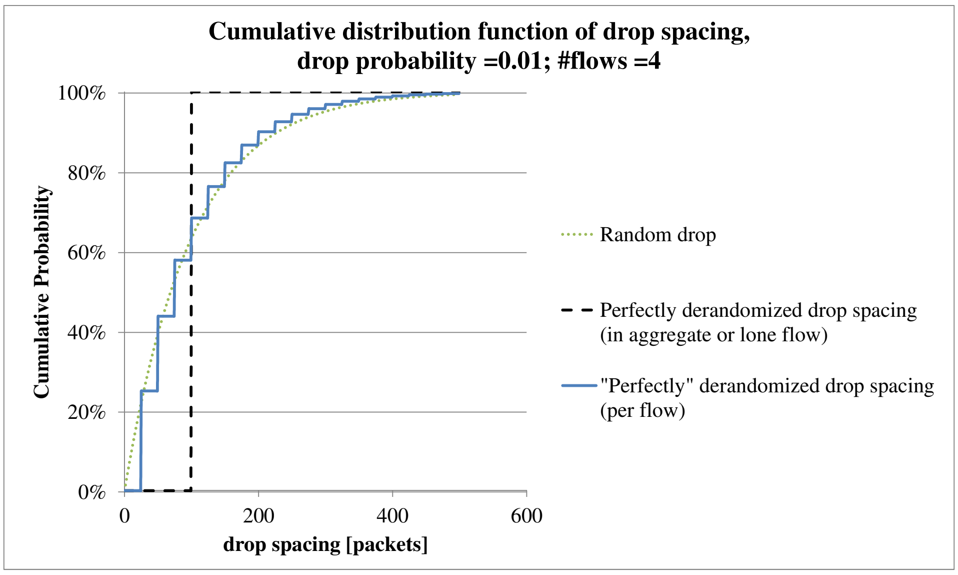
at #X	AND #X+100 =		1/4 = 25.00%
at #X AND NOT #X+100	AND #X+200 =	$(3/4)^1$ *	1/4 = 18.75%
at #X AND NOT #X+100 AND NOT #X+200	AND #X+300 =	$(3/4)^2$ *	1/4 = 14.06%
at #X AND NOT #X+100 AND NOT #X+200 AND NOT #X+300	AND #X+400 =	$(3/4)^3$ *	1/4 = 10.55%

But, for flow A, there are only *about* 100/4 = 25 packets between #50 and #150, because the other ~ 75 packets are in the other flows.

So, in general, the chance of about $25 * n$ packets between drops in one of the flows is

$$1/4 * (3/4)^{(n-1)}$$

So, this so-called 'perfect' derandomization gives the following chance for each of the following spacings between drops in flows A (and equivalently in the other flows):



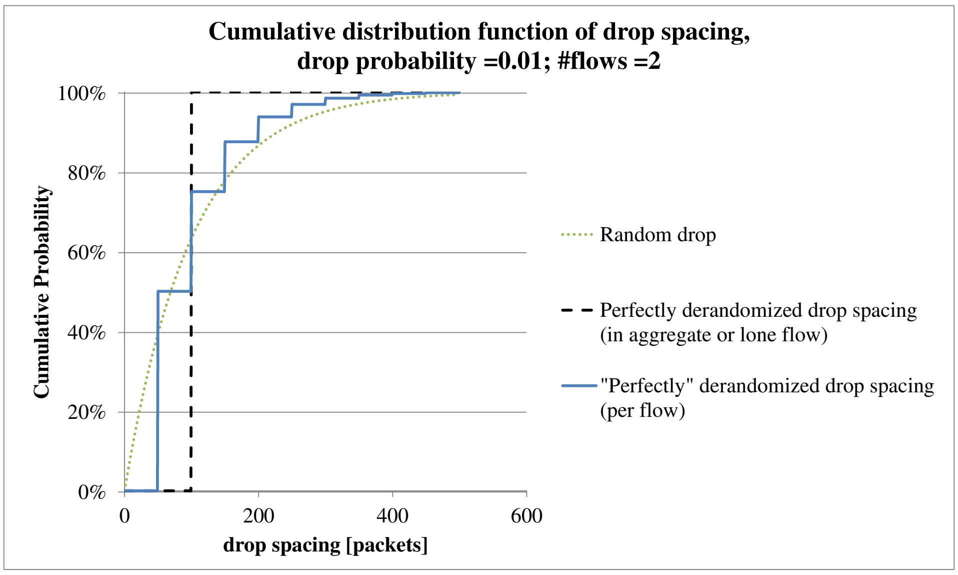


Figure 3: Rerandomization of Derandomized Drop Spacing

Drop Spacing	Probability	
~ 25	25.00%	
~ 50	18.75%	
~ 75	14.06%	
~ 100	10.55%	$\leftarrow$intended spacing
~ 125	7.91%	
~ 150	5.93%	
~ 175	4.45%	
~ 200	3.34%	
etc.		

In fact, for any number of flows greater than one, if the cumulative distribution function of spacings with derandomization is laid on top of the CDF without, they follow nearly the same curve, except the derandomized one is stepped (see Fig 3). Derandomization does avoid bunching of drops, but it does not prevent some drops arriving very far apart, as the figure shows. This is true no matter how few flows there are, unless there is just one (see the lower figure for 2 flows).

In practice, the outcome of derandomization will be even more random than shown in the plots, for two reasons:

- These plots are for a fluid flow. In practice TCP's rate varies considerably (the classic saw-tooth), so the steps would be less-pronounced. IOW, the futile attempt to derandomize drop spacing would be 'rerandomized' even more than shown.
- These plots assume perfect derandomization. The derandomisation code in PIE deliberately leaves some randomness in the spacings. Once these more random spacing are spread randomly over different flows, there would hardly be any derandomization left.

Admittedly, if per-flow queuing is used (with one PIE per flow), PIE's derandomization will make the drop spacing more regular. But then, what benefit does derandomizing spacing give anyway?

With random spacing, sometimes one flow gets hit more than once per RTT, and sometimes another flow gets lucky and doesn't get hit at all. But surely this small benefit from more regular drops would only be noticeable if all flows were long-running, smooth and stable. In normal traffic with lots of short flows as well as large flows coming and going, I doubt the difference of regular drop spacings would be noticeable amongst all the other noise.

Also, if p has been high, so that `accu_prob` has approached 8.5, then some flows depart or the link goes idle, p will reduce, but the artificial limit at 8.5 could well cause a drop when actually p is now very low and a drop is not warranted. IOW, derandomization makes drop depend on the history of p, not just the current state of PIE (which already depends on history in a deliberate way, but should not depend on history in this accidental way as well).

In summary:

- Is it worth including the derandomization code if multiple flows share the queue? No.
- Is it worth including the derandomization code with per-flow queuing? I'm not convinced.

That's why I say derandomization merely contributes to the heat death of the Universe. Ironic.

5.7 Bound drop probability below 100%?

This code fragment has been added to status_update():

```
        //bound drop probability
        if (PIE->drop_prob_ < 0)
                PIE->drop_prob_ = 0
        if (PIE->drop_prob_ > 1)
                PIE->drop_prob_ = 1
```

The lower bound of zero is fine. I believe the upper bound is broken. I think that allowing the possibility of 100% drop in the code creates a potential DoS vulnerability. The early RED implementations suffered from this problem, but they were fixed later.

Explanation: If the arriving load is persistently say 200% of link capacity, without an AQM the queue would grow to infinity, but tail-drop would drop 50% and utilization of the output link would be 100%. With an AQM it still only needs to drop 50%. If an AQM is programmed to drop 100%, it will give absolutely zero throughput — the output will completely dry up — 0% utilization.

I know PIE has been tested with unresponsive traffic and it was good (while CoDel was not). I assume that's because the PIE controller stabilized on a `drop_prob` that was just right to still achieve 100% utilization at the output, and keep average delay at the target. But I think that would only happen if the unresponsive traffic was stable.

If a botnet used a flooding attack that continually increased, I think it could push `drop_prob` up towards 100% by exploiting the `beta` term in the PIE controller. I guess it could pulse this attack every time the botnet hit its sending limit (or the limit of an upstream link). Under such an attack, I think PIE would keep latency low, but utilization would drop to near-zero.

My hypothesis would need to be tested. But if this attack were possible, it would be best to switch to tail drop above a certain `drop_prob`.

This was the way various RED implementations were fixed some years ago. For instance, search for "forced" in the Apple derivative of the OpenBSD RED code here: <http://www.opensource.apple.com/source/xnu/xnu-2050.48.11/bsd/net/classq/classq_red.c> I recall that Fred Baker said he did something similar in the Cisco RED code.

5.8 Lock-out Bias against Large Packets

If PIE has to fall back to tail drop, at least it should always stop enqueuing packets when there is < 1MTU (max transmission unit) free at the tail. This prevents the bias against large packets that tail drop otherwise suffers from, which is a benefit of AQM that does not need to be abandoned even when the system is driven to tail drop.

In other words, in the enque() block in the pseudocode:

```
CURRENT
        if (queue_.is_full()) {
         drop(packet);
        }
SUGGESTED:
        if (queue_.byte_length() > (buffer_size - MTU) ) {
         drop(packet);
        }
```

With suitable definitions of the two new variables, `buffer_size` and `MTU`.

6 Numerous Magic Numbers

I counted 20 magic numbers in the pseudocode. Below, I have tried to reduce this by showing how some of them should depend on others.

However, my first gripe is that they should all be called out in the header of the pseudocode. I found 13 of them hard-coded within the pseudocode.

Configurable Parameters:

- `QDELAY_REF`. AQM Latency Target (default: 16 [ms])
- `BURST_ALLOWANCE`. AQM Latency Target (default: 150 [ms])

Internal Parameters:

- Weights in the drop probability calculation:
  - `alpha` (default: 1/8 [Hz])
  - `beta` (default: 1 + 1/4 [Hz])
  - the look-up table of factors:

`drop_prob`	`factor`
$< 0.1\%$	1/128
$< 1\%$	1/16
$< 10\%$	1/2
`else`	1

- `DQ_THRESHOLD` (default: 2^{14} [B])
- `T_UPDATE`: a period to calculate drop probability (default: 16 [ms])
- `QUEUE_SMALL` = (1/3) * Buffer limit in bytes
- `accu_prob`
  - min (default 0.85)
  - max (default 8.5)
- uncongested queue test#1 (default: `QDELAY_REF`/2)
- uncongested queue test#2 (default: `drop_prob_` < 20%)
- uncongested queue test#2 (default: `queue_.byte_length()` <= 2 * `MEAN_PKTSIZE`))
- EWMA constant for `avg_dq_time_` (default: 1/4)

Not all these parameters are scenario-independent. So there should be a category between 'configurable' and 'internal' called perhaps 'scenario-dependent'.

The spec needs to state how to set these scenario-dependent parameters for different environments. And if one internal parameter depends on another, it should be set relative to the other, not as a separate constant. Specifically:

- The look-up table of factors to scale alpha and beta might need to be changed if the dominant transport changes (see earlier).
- `DQ_THRESHOLD` = 2^{14}B is equivalent to about 10–11 Ethernet frames. In buffers carrying large numbers of flows, the average queue can be kept about this large, or smaller, without losing utilisaation. So, if 2^{14} were not configurable, PIE's rate measurement might only be triggered rarely in these highly aggregated buffers.
- `T_UPDATE` = 16ms would not be appropriate in short RTT scenarios (e.g. data centres or private networks), where it could lead to the same drop probability being used for many RTTs.

- It would be better to initialise `T_UPDATE` as a number of bytes divided by the nominal link rate, rather than a fixed amount of time. Then, without having to change the code, it will tend to scale with the available processing capacity, which tends to be sized per packet. E.g.
  `B_UPDATE` = 96kB
  `T_UPDATE = B_UPDATE / NOMINAL_LINK_RATE` // = 16ms for a 48Mb/s link
- `alpha` and `beta` should be specified in terms of `T_UPDATE` in the code. E.g.
  `ALPHA_DEFAULT = T_UPDATE`/1ms
  `BETA_DEFAULT = ALPHA_DEFAULT` * $10^{\{\mathrm{Note1}\}}$
  Then if `T_UPDATE` = 16 ms, `ALPHA_DEFAULT` = 16, `BETA_DEFAULT` = 10*16=160
  And if `T_UPDATE` is ever changed to say 4 ms, `ALPHA_DEFAULT` and `BETA_DEFAULT` will correctly scale to 4 and 40 respectively.
- `QUEUE_SMALL`. Why 1/3 of the buffer? Why depend on the buffer size at all? This makes PIE depend on a sensible buffer size setting, while one of the reasons for introducing AQM is to decouple queue size from buffer size. Whatever, 1/3 seems like it was chosen assuming a buffer for a small number of flows. `QUEUE_SMALL` should surely be different for a highly aggregated link?
- `accu_prob`. Why limits of 0.85 and 8.5? Have it been tested whether these values are sufficient to prevent periods of synchronisation with TCP? Might these values need to be reconfigured if we find synchronisation occurs in certain scenarios?
- The three uncongested queue tests seem specific to a buffer intended for a small number of flows. Might they need to be reconfigured for a highly aggregated link?
  `drop_prob_` < 20% seems a very high threshold for 'uncongested'.
  Might this have to be reconfigured after operational experience?
- I think the EWMA-constant for `avg_dq_time_` (default: 1/4) should depend on `DQ_THRESHOLD`.
  Specifically, EWMA-constant = `DQ_THRESHOLD`/2^{16}.
  Because the lower that `DQ_THRESHOLD` is set, the more often there is a reading for `dq_time`, so the contribution of each reading to the average needs to be reduced. And this still gives EWMA-constant = 1/4 with the recommended value of `DQ_THRESHOLD` = 2^{14}.

7 Implementation Suggestions

Earlier, I promised that the recalculation of p could be implemented using only adds and bit-shifts, even though I have suggested that the look-up table can also be removed, which seems to require two doubles to be multipled together:

```
p = p + p * [ALPHA_DEFAULT*(qdelay - QDELAY_REF) + BETA_DEFAULT*(qdelay - PIE->qdelay_old_)].
```

Here's how:

```
p = p + ( ((qdelay - QDELAY_REF) + ((qdelay - PIE->qdelay_old_) << LG_BA_RATIO))
          << (bsr(p) + LG_ALPHA_DEFAULT) )
```

This single expression consisting of adds and shifts replaces the whole look-up table in the pseudocode, and scales to indefinitely small p (down to the resolution of the number representation in the machine, to be picky).

where

```
ALPHA_DEFAULT = T\_UPDATE}/1ms              // = 16
LG_ALPHA_DEFAULT = lg(ALPHA_DEFAULT)        // = 4
BA_RATIO = 8                                // {Note 1}
BETA_DEFAULT = ALPHA_DEFAULT * BA_RATIO     // = 128
LG_BA_RATIO = lg(BA_RATIO)                  // = 3
```

> bsr(p) means the integer log (base 2) of p, which is simply the position of the highest bit set to 1. On Intel Architecture this is implemented in assembler as the bsr instruction, which consumes a single op. This is OK, because alpha and beta only need to be scaled by the order of magnitude of p, not its precise value.

{Note 1}: For simplicity, I've used `BA_RATIO` = 8, rather than the value 10 suggested in the draft.

8 Nits

All Sections

s/user/operator/g
The use of 'we' and 'our' throughout is not really good style for an RFC.

Abstract

Consider adding 'Web' to the list of latency-sensitive apps.

```
s/The design does not require per-packet timestamp/
 /The design does not need to add per-packet timestamps/
```

Rationale: Otherwise readers not familiar with the WG discussions might think this means timestamps added by hosts.

1. Introduction

```
CURRENT:
   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.
SUGGESTED:
   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 in the presence of long-
   running greedy flows.
```

Rationale: 'long-term persistent congestion' could be ambiguously interpreted as pathologically severe congestion.

```
CURRENT:
   Separately, we assume any delay-based AQM scheme would be applied
   over a Fair Queueing (FQ) structure or its approximate design, Class
   Based Queueing (CBQ).
SUGGESTED:
   Separately, any delay-based AQM scheme could be applied
   within a Fair Queueing (FQ) structure or its approximate design, Class
   Based Queueing (CBQ).
```

Rationale: A few sentences later the draft says "...advantages such as per-flow/class fairness are orthogonal...," so it is clearly unnecessary to assume that any delay-based AQM would be applied over FQ.

If no-one can argue against the subsequent paragraph (my own words from [Bri07]), please remove the implication that max-min is a desirable goal in the following sentence.

```
   [FQ or CBQ]
   help flows/classes achieve max-min fairness and help mitigate bias
   against long flows with long round trip times(RTT).
```

"In 1997, Kelly demonstrated [Kel97] that . . . Users would only have chosen max-min if they valued bit rate with an unrealistically extreme set of utility functions that were all identical and that all valued low bit rate infinitesimally less than high bit rate. To spell Kellys result out even more bluntly, max-min fair rate allocation would only be considered fair if everyone valued bit rate in a really weird way: that is, they they all valued very low bit rate hardly any less than very high bit rate and they all valued bit rate exactly the same as each other."

s/standard DropTail/regular drop tail/

```
s/In October 2013, [...] DOCSIS 3.1 [...] mandates/
 /In October 2013, [...] DOCSIS 3.1 [...] mandated/
```

"The previous draft of PIE. . . " Which previous draft?

I suggest the following sentence is stated earlier. Even tho I think the draft means that this is a delta since the 'previous draft' it is more important than just a delta. Perhaps it would be most relevant after ". . . they could be made self-tuning to optimize system performance."

```
   We also discusses a pure enque-based design where all the
   operations can be triggered by a packet arrival.
```

3. Design Goals

```
s/we directly control queueing latency instead of controlling queue length/
 /we indirectly control queueing latency instead of controlling queue length/
```

Rationale: If PIE used time-stamps it would control queueing latency directly. PIE only controls queueing latency as accurately as its rate estimate allows for the conversion of queue length to latency.

```
s/As a matter of fact, we would allow more buffers for
        sporadic bursts as long as the latency is under control./
 /In fact, once latency is under control it frees up buffers for
        sporadic bursts./
```

s/On the other hand, however, /However, /

s/number streams/numbers of streams/

4.1 Random Dropping

```
s/Like any state-of-the-art AQM scheme, PIE would drop packets randomly/
 /PIE drops packets randomly/
```

Rationale: The other scheme with a claim to be state-of-the-art doesn't (CoDel).

I would agree if the draft had said "A state of the art scheme *should* introduce randomness into packet dropping in order to desynchronize flows," but maybe it was decided not to introduce such underhand criticism of CoDel. Whatever, the draft needs to be careful about evangelising random drop, given it attempts to derandomize later.

4.2 Drop Probability Calculation

```
s/the direction where the delay is moving/
 /the direction in which the delay is moving/
```

```
s/We would like to point out that this type of controller has been studied before/
 /This type of controller has been studied before/
```

```
"The theoretical analysis of PIE is under paper
 submission and its reference will be included in this draft once it
 becomes available.
"
```

Does this refer to a second paper, or was this written before the HPSR'13 paper was accepted?

4.3 Departure Rate Estimation

```
   qlen > dq_threshold
```

add

```
   start = now
```

```
s/deq_threshold/
 /dq_threshold/
```

5. Design Enhancement

Redundant sentence: "For clarity purpose, we include them here in this section."

5.2 Auto-tuning of PIE's control parameters

```
s/discuss the intuitions regarding/
 /discuss the intuition regarding/
```

```
s/big swings [...] often leads to/
 /big swings [...] often lead to/
```

```
s/There are could be/
s/There could be/
```

```
several regions of these tuning, extendable all the way to 0.001% if
needed.
```

This sentence may need reviewing if my proposed autotuning in §5.2 is adopted. Anyway, why is 0.001% the smallest possible number anyone might know (see the table of p for various Cubic flow rates in §5.2)?

```
As it is not showing explicitly in the above
equation, it can become an oversight.
```

Well, show it explicitly then (see my suggestion in §6). Also, s/showing/shown/

s/hz/Hz/

5.3 Handling Bursts

```
s/We would like to discuss how PIE manages bursts in this section
when it is active./
 /In this section, we discuss how PIE manages bursts
when it is active./
```

```
s/will be enqueued bypassing the random drop process./
 /will be enqueued, bypassing the random drop process./
```

```
s/defined by us as p/
 /defined as p/
```

5. De-randomization

```
s/cause real drop percentage to deviate/
 /cause real drop percentage to temporarily deviate/
```

New para before "We keep a parameter called `accu_prob`"

```
s/This avoids packets are dropped/
 /This avoids packets being dropped/
```

6. Implementation and Discussions

```
Hence, a drop at enqueueing can be readily retrofitted into existing
hardware or software implementations.
```

Yes, but please also sate what would be recommended for new hardware.
And s/enqueueing/enqueuing/

```
The state requirement is only two variables
per queue: est_del and est_del_old.
```

It is true that state per queue is low, but there is no need to omit `accu_prob`, `burst_allowance`, `dq_count`.

```
departure rate, which can be implemented using a multiplication
```

This sentence might need to be reviewed (see "Should Use Harmonic Averaging" in §??).

```
s/keep track an interface's/
 /keep track of an interface's/
```

```
s/independent to each other/
 /independent of each other/
```

```
s/the number of extra packets in queue/
 /the number of remaining packets in the queue/
```

```
s/PIE is simple to be implemented./
 /PIE is simple to implement./
```

Incidentally, IMO this statement is not warranted (see "Main Concerns" at the start).

New para before "SFQ can be combined with PIE…".

Future Research

```
s/What is presented in this document is the design of the PIE algorithm, which effectively/
 /The design of the PIE algorithm is is presented in this document. It effectively/
```

I suggest each research item in this first para is bulleted.

Add:

1. Autotuning of target delay without losing utilisation;
2. Autotuning for ave RTT of traffic;
3. ECN (unless in future it is included in the body).

```
s/and control path. If/
 /and control path, if/
```

Incremental Deployment

```
s/One nice property of the AQM design/
 /One nice property of AQM design/
```

s/can not/cannot/

```
s/low latency service for real-time applications/
 /low latency service for all applications/
```

Rationale: many latency-sensitive apps are not real-time. And AQM provides (fairly) low latency for all traffic, unlike EF which only provides low latency for an identified and limited small proportion of capacity.

Pseudocode Appendix (Section 10) It is sort-of clear that there are two entry points into the pseudocode blocks:

- `enque()` is called on each packet arrival
- `deque()` is called on each packet departure

But it would be useful to state this explicitly, and to state that this is the variant shown in Fig 1, not Fig 2 (that is only called on enqueue).

In the `drop_early()` block, I assume ENQU and DROP are meant to be defined as FALSE and TRUE.

"4.3 Departure Rate Estimation"
CURRENT:

```
if dq_count > dq_threshold then
    depart_rate = dq_count/(now-start);
    dq_count = 0;
    start = now;
```

SUGGESTED:

```
if dq_count > dq_threshold then
    depart_rate = dq_count/(now-start);
    dq_count -= dq_threshold;
    start = now;
```

RATIONALE:
The suggested code carries over the remainder of `dq_count_` to the next cycle with no extra operations. The current code loses the remainder in every measurement cycle, so the estimated `depart_rate` is always biased on the low side.

References

[BF15] Fred Baker and Gorry Fairhurst. IETF Recommendations Regarding Active Queue Management. Internet Draft draft-ietf-aqm-recommendation-11, Internet Engineering Task Force, February 2015. (Work in Progress).

[Bri07] Bob Briscoe. Flow Rate Fairness: Dismantling a Religion. *ACM SIGCOMM Computer Communication Review*, 37(2):63–74, April 2007.

[Bri09] Robert (Bob) Briscoe. *Re-feedback: Freedom with Accountability for Causing Congestion in a Connectionless Internetwork*. PhD thesis, UC London, 2009.

[HMTG01] C. V. Hollot, Vishal Misra, Donald F. Towsley, and Weibo Gong. On Designing Improved Controllers for AQM Routers Supporting TCP Flows. In *INFOCOM*, pages 1726–1734, 2001.

[Kel97] Frank P. Kelly. Charging and Rate Control for Elastic Traffic. *European Transactions on Telecommunications*, 8:33–37, 1997. A version with a correction by Ramesh Johari and Frank Kelly to distinguish flows with zero weight and using a better structure of proof is available from URL: http://www.statslab.cam.ac.uk/~frank/elastic.html.

[PNB+15] Rong Pan, Preethi Natarajan, Fred Baker, Bill Ver Steeg, Mythili Prabhu, Chiara Piglione, Vijay Subramanian, and Greg White. PIE: A Lightweight Control Scheme To Address the Bufferbloat Problem. Internet Draft draft-ietf-aqm-pie-01, Internet Engineering Task Force, March 2015. (Work in progress).

[PPP+13] Rong Pan, Preethi Natarajan Chiara Piglione, Mythili Prabhu, Vijay Subramanian, Fred Baker, and Bill Ver Steeg. PIE: A Lightweight Control Scheme To Address the Bufferbloat Problem. In *High Performance Switching and Routing (HPSR'13)*. IEEE, 2013.

Document history

Version	Date	Author	Details of change
00A	01-May-15	Bob Briscoe	First Draft;
00B	01-May-15	Bob Briscoe	Converted from txt to Tech Report, added figures.
00C	06-May-15	Bob Briscoe	Added critiques of averaging, burst allowance, rate estimation, large-packet lock-out.
01	08-May-15	Bob Briscoe	Issued without further change.
01A	09 May 2015	Bob Briscoe	Clarified large packet lock-out (§5.8) and integer log (§7)