How to Build a Virtual Queue from Two Leaky Buckets (and why one is not enough)

Bob Briscoe* Gabriele Corliano

Ben Strulo

BT Innovate & Design, Adastral Park, Martlesham Heath, Ipswich, IP5 3RE, UK

04 Oct 2017

Abstract

A virtual queue can be used to predict when a real queue is about to grow. This memo explains why using a leaky bucket to implement a virtual queue is not useful, despite a superficial similarity. However, it is possible to implement a useful virtual queue using *two* leaky buckets. It requires a simple trick that can typically be implemented with existing hardware.

1 Introduction

A virtual queue imitates how a real queue would behave if it were feeding a line of less capacity than the real line. The length of the virtual queue can then be used to feed notifications into the control loop that regulates the load on the real queue. This can keep the real queue extremely short.

A virtual queue does not actually hold any data; it is merely a number that is incremented as packets arrive and decremented continuously in order to model packets being sent to an imaginary link slower than the real one.[1]

The goal of this paper is to briefly describe a neat trick for implementing a virtual queue using existing hardware. It merely involves switching round an 'if' and 'else' clause in a single rate three colour marker [HG99], which is a linked pair of leaky buckets widely available in existing networking chipsets. In the chipset we chose to modify, this involved simply flipping a bit in a data table.

That in itself would require just a two-page memo (§4 & Appx. A). However, it takes longer to explain how *not* to implement a virtual queue. We argue against the assumption that a leaky bucket is sufficient to implement a useful virtual queue.

A leaky bucket was originally conceived as a way to limit traffic under the assumption of an open-loop control model. It was not originally designed for signalling information into a closed-loop control system. For that, we argue that the trick with two leaky buckets is needed—one is not enough.

Conclusive evidence to back up our argument will require further work. In this brief memo, we outline the intuition and point to incidental evidence in existing empirical studies to support our argument.

2 Related Work

Virtual Switch: Coucoubetis and Weber first proposed the virtual queue in an ATM setting as a way to rapidly estimate a very low loss probability. It was devised so that timely decisions could be made on flow admission or re-routing, without having to wait for very rare loss events [CW96, §5]. They derived the asymptotic loss probability for N superposed traffic streams as $N \to \infty$ in a switch where the service rate and buffer size both increase linearly with N. Because they found that the loss probability scales $O(exp^{-N})$, they suggested that very low loss probabilities could rapidly be estimated by feeding a sample of $1/k$ of the streams into what they called a 'virtual switch' with $1/k$ of the line rate and $1/k$ of the buffer size of the actual switch. For instance, they claimed that one can estimate an actual loss probability of 10^{-10} without waiting a few tens of billions of cells for the extremely rare actual loss events. Instead one only needs to wait a few hundred cells to measure the much higher 10^{-2} (1%) loss probability of a virtual switch that is scaled down by a factor of 1/5

*bob.briscoe@bt.com,

[1] The term virtual queue is also used for a queue on an ingress interface that tracks the state of the queue on an egress interface of a router or switch. This is not the sense intended in this paper.

(i.e. $k = 5$). This virtual switch then models the actual loss probability of $10^{-2k} = 10^{-10}$.

GKVQ: Gibbens and Kelly's virtual queue [GK99] built on this idea, but moved away from scaling down the traffic, only scaling down the line rate and buffer size. Without having scaled the traffic, they recognised that the predictive power of a virtual queue is best when its capacity and buffer size are only slightly less than the real buffer. With a scaling factor just under one, the algorithm gives a reasonably accurate prediction of the real queue without the complexity of sampling to scale down the traffic. The gain in simplicity comes at the expense of losing a fraction of line utilisation.

AVQ: Kunniyur & Srikant [KS01] propose a virtual queue with only the drain rate scaled down, not the buffer size or load. Unlike the schemes above, the scaling factor is not fixed, but a target utilisation is fixed and the scaling factor adapts slowly downwards if arriving traffic exceeds this fixed utilisation, hence they called it an adaptive virtual queue (AVQ).[2] For the present paper, AVQ is an excellent example of how ineffective a virtual queue becomes when it is implemented with a leaky bucket (see §3.2).

PCN: The IETF uses the term pre-congestion notification (PCN) for marking driven by a virtual queue rather than the real queue. It has standardised two virtual queue algorithms [Ear09]:

ETM: Excess traffic marking, meaning only marking the packets that exceed a configured bit-rate (we call this 'tail-marking' in the present memo).

ThM: Threshold marking, meaning marking all packets when the virtual queue exceeds a threshold.

The IETF's PCN algorithms are for marking inelastic traffic in a prioritised scheduling class. The idea is to detect whether the higher priority traffic has exceeded a preconfigured fraction of the line-rate, and consequently trigger flow admission control or even flow termination to protect the service offered to the remaining traffic. Initially, a ramp function was proposed to trigger marking because it was thought that a step function might cause instability. However, wide-ranging simulations at different institutions showed that a step was sufficient (e.g. Zhang's PhD thesis [Zha09] and the report of Menth & Lehreider [ML07]). Although the PCN standard algorithms were targeted at controlling inelastic traffic, now that they are implemented it is possible a step function might be applicable for control of elastic traffic too (see §5 on applications of virtual queues).

[2] Unfortunately, the scaling factor also slowly adapts upwards (with a cap at 1) when utilisation is less than the fixed target. Thus, after a relatively short period of low utilisation, the scaling factor will rise to 1. At the start of the next busy period, just when the predictive power of the virtual queue is most needed, it will not grow any sooner than the real queue. By the time it has adjusted its drain rate, the busy period might well have ended. This effect was never observed in [KS01], because all the simulations included infinite flows, therefore the busy period continued indefinitely.

3 One Bucket is not Enough

3.1 Terminology

Throughout this paper the term leaky bucket should be considered to also imply the token bucket alternative. An implementation of either can always serve the purpose of the other, by simple rearrangement of the inputs and outputs. A leaky bucket fills with tokens that represent the size of each arriving packet and it drains at a configured constant virtual rate. A token bucket fills at the constant virtual rate and drains tokens that represent the size of each arriving packet (Figure 1).

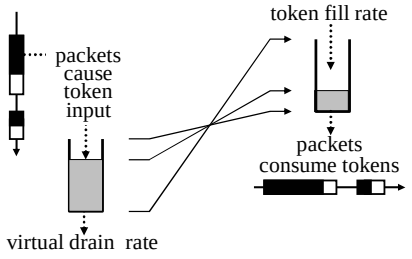

Figure 1: Equivalence of token bucket (right) and leaky bucket (left)

3.2 Leaky Bucket ≡ Virtual Queue?

When it comes to implementing virtual queue algorithms, superficially it seems that a leaky bucket would suffice. Network kit often provides hardware leaky buckets. Therefore, as long as a leaky bucket can be configured to mark rather than drop packets when it has filled, it seems that it is already possible to implement a virtual queue with current hardware.

However, we will now use Figure 2 to explain why a virtual queue implemented with a leaky bucket loses too much information to be generally useful for congestion marking. We will compare a virtual queue that is implemented using

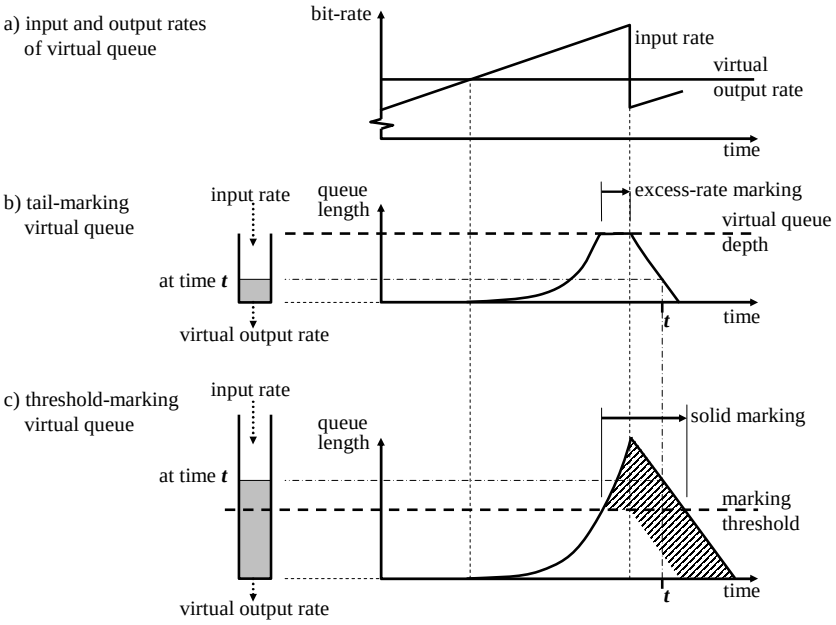

Figure 2: Comparison of virtual queues with b) tail-marking and c) threshold marking, both with the same input and output rates (a)

a leaky bucket, with one designed specifically for congestion-marking. They happen to be represented by the two types of virtual queue standardised by the IETF, which is not surprising given one was standardised because it could be implemented with current leaky bucket hardware, while the other represented the preferred target design:

Tail-marking (Figure 2b): This algorithm marks packets only when the queue is full which leads to excess traffic marking. That is, only the packets that exceed the configured bit-rate are marked;

Threshold-marking (Figure 2c): This algorithm marks all packets when the queue exceeds a threshold part-way between full and empty. Unlike tail-marking, threshold-marking cannot be implemented with existing leaky bucket hardware, which invariably does not have the facility to mark when only part-full.

To make the two comparable, we imagine that the threshold is set at the same depth as the full depth of the tail-marking variant. We also consider that both variants are subjected to the same pattern of input rate and the same constant virtual output rate, as shown in Figure 2a).

From the point when the input rate first crosses the virtual output rate, both virtual queues grow identically, until the tail-marking variant has filled and the threshold marking variant reaches its threshold. Then both virtual queues start marking at the same instant.

From this point, the behaviour of the two variants diverges. With the input rate continuing to exceed the output, the threshold-marking variant continues to fill beyond the threshold, while the tail-marking variant (conceptually) overflows. This causes their marking behaviour to differ. The threshold-marking queue marks all packets, while the tail-marking queue marks only the fraction of packets that exceed the virtual output rate. The latter is because, every time a packet is marked, the virtual queue cannot grow further, but it still empties at the virtual output rate. So the virtual queue continually drains just enough to fit in a proportion of unmarked packets that matches its output rate.

For illustration, we imagine that some external load reduces its input rate sharply in response to onset of congestion signals, which results in the downward sawtooth of the input rate at the queue shown in Figure 2a). Therefore, a round-trip after marking starts, the effect of the source backing down below the virtual output rate reaches the input of the queue. This causes both variants of virtual queue to start to empty. The tail-marking variant is no longer full therefore it immediately stops marking. Whereas the threshold marking variant continues marking because it is still well above its threshold. By time t, which is the instant illustrated in the left hand diagrams, the tail-marking virtual queue is nearly empty while the threshold-marking variant is still filled above the threshold and therefore still marking.

A short while after time t the tail-marking virtual queue becomes completely empty, while the

threshold-marking variant is still marking. A little later it stops marking but it still has the rest of the virtual queue to drain before it finally empties some time later.

It may seem a virtue that the tail-marking queue stops marking as soon as the input rate drops below the output. However, in the next section (§3.3) we will discuss why it is not necessarily desirable to curtail signalling.

The length of a queue represents the sum (or equivalently the integral, in a fluid model) of the excess of arrivals over departures. So, for instance, if the input rate increases linearly as in Figure 2a), the queue length will trace the integral of a linear function: a square law curve, illustrated by the convex hockey-stick shape in Figure 2c). This convexity gives the system inherent stability without having to do anything more complex than making congestion signalling depend on queue length. Even if the input rate is only slightly above the output rate, the queue and therefore the amount of congestion signalling will continually increase.

In contrast, the tail-marking virtual queue in Figure 2b) merely marks linearly—in proportion to excess rate. Once it starts marking, it signals no differently whether the excess of load over output has been persistent or transient. In Figure 2c) the shaded area represents the information that the tail-marking queue forgets, while the threshold marking queue retains it.

AVQ provides an excellent illustration of the knock-on problems that arise when a tail-marking queue is used to generate congestion marks. Recall that AVQ adapts the drain rate of the virtual queue in order to aim at a target utilisation; set to 98% in the AVQ simulations [KS01]. When using a virtual queue for congestion control, the aim is to track movements of the real queue as faithfully as possible, by setting the virtual capacity just below the real capacity, hence AVQ's choice of 98%. However, even in AVQ's simplest simulation scenarios with just long-running TCP file transfers, the drain rate of AVQ's virtual queue adapts downwards until it hovers below 49% of actual capacity. It seems that AVQ has to push the virtual drain rate of the virtual queue down this far in order to get it to generate enough marks to get the load to reduce to 98% of capacity. This is because a tail-marking queue fails to retain information about packet arrivals once it is overflowing. With such a wide discrepancy between the virtual and the real drain rates, this tail-marking queue will no longer be useful for predicting the movements of the real queue.[3]

[3]We will need to test whether AVQ with threshold-marking would adapt the virtual drain rate to about 98%.

3.3 A Design Principle

Gibbens & Kelly argue that the number of packets marked[4] should be the same as the number that contribute to the queue exceeding its marking threshold (sample path shadow pricing or SPSP [GK99]).[5] Intuitively one can think of this as a design principle that recommends 'conservation of congestion information'.

However, it is inadvisable to mark packets that contributed to the initial growth of the queue (because one cannot tell whether an increasing queue will just fall away of its own accord, or continue until it hits the threshold, at which point most packets that contributed to this queue growth will have already departed). Therefore Gibbens & Kelly argue that packets should continue to be marked even after the queue starts to shrink; in order to compensate for those that were not marked before the threshold was reached. This is why the GKVQ algorithm continues to mark until the queue is empty and the busy period is over.

Wischik assesses how various marking algorithms (RED, REM, PI, BLUE) compare with the ideal but unattainable SPSP of Gibbens & Kelly. It compares how many packets each marks, and whether some designs are more prone to some flows being unfairly marked when other flows caused the congestion. Wischik also proposes a new algorithm called ROSE that improves on them all [Wis99, §5].

It can be seen from the above that the tail marking algorithm is very poor at conserving congestion information. It cannot hold a count of how many bits arrive after the threshold is exceeded, which is why a leaky bucket is unlikely to be effective for signalling congestion information in a closed loop system. In contrast, the threshold-marking virtual queue is better with respect to 'conservation of congestion information'. It is not perfect, for instance it does not continue marking until the queue is empty. But at least it continues to mark while the queue is emptying down to the threshold.

As we shall see in the next section, there is a simple way to implement a threshold marking virtual queue using existing hardware. This tempts us to wonder whether a threshold marking virtual queue might be good enough as a general purpose AQM algorithm. Undoubtedly threshold marking will not be as good as ROSE. But it only has to be *good enough*, not necessarily perfect.

[4]Strictly the number of bytes in marked packets.

[5]Standard TCP does not take note of the amount of congestion signalling within a round trip—it only distinguishes between none or some. However, modern transports are being produced that do—to excellent effect, e.g. Relentless TCP [Mat09] and Data Center TCP [A+10]. So, it is worth worrying about how many marks an algorithm generates.

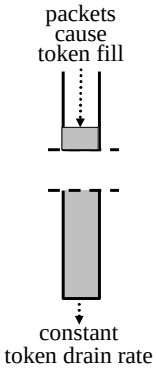

Figure 3: Threshold-marking leaky bucket conceptually split into two leaky buckets

4 Building a Virtual Queue from Two Leaky Buckets

We now present a trivially simple way to implement a threshold marking virtual queue using existing hardware.

A threshold-marking virtual queue can intuitively be thought of as two leaky buckets stuck one on top of the other so that the join between them forms the marking threshold (Figure 3). What we need is hardware that can mark packets if the bottom bucket is full and that implements logic to join the two buckets together.

The standard single rate three colour marker (srTCM [HG99] see Figure 4) seems close to what we want, but it's not quite right. It consists of two leaky buckets, C for the committed burst size and E for the excess burst size, plus token filling and consuming algorithms. It can mark packets if one of the buckets is full and it has logic to join together the two buckets, but not quite the right logic for our purposes.

The consuming algorithm of an srTCM will not consume tokens from bucket E unless it has already emptied bucket C. And similarly the filling algorithm will not fill bucket E unless it has already filled bucket C. In other words, it consumes tokens from bucket C preferentially and fills bucket C preferentially as well.

For our purposes, we certainly need to fill C preferentially but we want to consume from E preferentially, not C. This slight change is the only substantial difference between the standard srTCM in Figure 4 and the threshold marking virtual queue in Figure 5 (carefully compare the text at the bottom beside each consuming algorithm).

Appendix A uses pseudocode to show how simply switching round one 'if-else' clause turns an srTCM into a threshold marking virtual queue.

Three colour markers are a standard feature available in most modern networking chipsets. They are used to colour the IP-Diffserv, Ethernet 802.1p or MPLS Traffic Class codepoints in packets [NBFB98, IEE97, LFE+02] depending on whether the rate exceeds a committed burst size or an excess burst size. Specifically, a TCM marks packets 'green' if neither bucket is full, 'amber' if the committed bucket is full and 'red' if both buckets are full, where green, amber and red are three different codepoints that indicate traffic profiles respectively in contract, out of contract but within an excess allowance, and fully out of contract.

A threshold marking leaky bucket would mark the ECN field whenever the equivalent three colour marker would have re-coloured the class of service to amber or red. This is an easy change, because existing three colour markers can always be reconfigured to map colours to codepoints in headers.

Nonetheless, although colour re-configuration is easy, the consuming algorithm still needs the slight change above. Wide availability of srTCMs that need to be slightly changed would be irrelevant if this slight change were an impossible change.

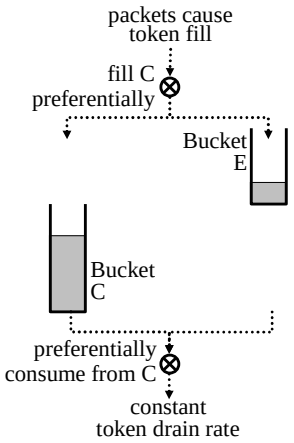

Figure 4: The standard single rate three colour marker (srTCM)

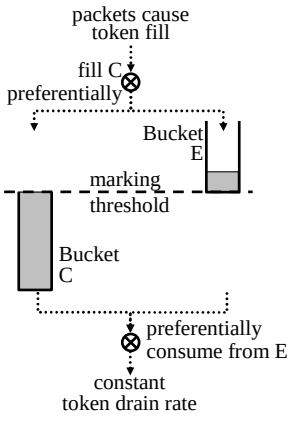

Figure 5: How to build a threshold marking leaky bucket from two leaky buckets

Happily, our colleagues from Broadcom found that it was straightforward to switch round the filling logic of the srTCM in their chipsets. The original TCM standards define different variants of TCM. Therefore Broadcom's TCMs use common hardware for the pair of buckets, while the particular variant of logic that links the buckets together depends on the setting of a data table. A new structure only needs a new setting in this table.

Using this technique, a threshold marking virtual queue has been available in several Broadcom switch devices since 2009. We hope this paper encourages other chipset developers to investigate whether a similar trick can create threshold marking virtual queues from existing hardware TCMs.

5 Applications of Threshold-Marking Virtual Queues

As pointed out under Related Work, the threshold-marking virtual queue has been standardised by the IETF as a way to signal whether inelastic traffic should be admission controlled [Ear09].

Of greater current interest is the potential use of a virtual queue as the active queue management (AQM) mechanism for elastic traffic—which could involve widespread deployment in buffers throughout the Internet (at layer 3 and below).

Data centre TCP (DCTCP [A+10]) already uses a simple step threshold function very successfully in practice. DCTCP uses the fact that a step is a degenerate case of the RED ramp. It turns RED's queue averaging time down to zero and it turns RED's ramp function into a step threshold by setting the minimum and maximum of the ramp to the same value. In DCTCP, the threshold is already set very shallow, so a threshold-marking virtual queue would seem to be a natural evolution direction—aiming to sacrifice a little utilisation for predictable and ultra-low queueing delay.

An initial concern is whether a step threshold will lead to synchronisation and/or oscillations, which is why random early detection (RED [FJ93]) uses a sloping ramp function and its marking is randomised. No instability problems have been reported with DCTCP. However further research will be needed to rule out the possibility in the wide range of scenarios possible in public networks.

6 Conclusions & Further Work

This paper has deprecated a single leaky bucket as a way to implement a virtual queue for closed-loop control of elastic traffic, because it fails to remember the traffic that arrives while the queue is overloaded. This could lead to pathological effects, including the possibility of not properly clearing a standing queue.

Nonetheless, we have proposed a better way to implement a virtual queue using existing hardware—using *two* leaky buckets (or equivalently, two token buckets). It simply involves switching round the order of an if-else clause in a single rate three colour marker (srTCM), which is a linked pair of leaky buckets. Given wide availability of srTCMs in networking chipsets, this opens the possibility of widespread virtual queue deployment.

This results in an instantaneous step function, in contrast to the smoothed gradual functions of RED and other AQM algorithms. Data Centre TCP (DCTCP) is an existence proof that end-system algorithms can be tailored to such a rudimentary AQM in the network. DCTCP has been evaluated with a virtual queue, but it does not provide the natural packet pacing that a real queue introduces, therefore pacing was added in the sender's network interface hardware [AKE+12]. The robustness of virtual queues for elastic traffic control was also assessed via simulation [LBS05], but not with such an unsmoothed step function. It will be necessary to map out a way in which virtual queues could be incrementally deployed in public networks with a mix of ECN and non-ECN transports.

Acknowledgements

We are grateful to the following Broadcom staff who explained their hardware and designed and implemented the proposed changes: Mohan Kalkunte, Bruce Kwan, Puneet Agarwal, Ashvin Lakshmikantha, Martin Weetman and Jim McKeon. We are also grateful to Michael Menth of Tübingen University for his useful review comments.

A Pseudocode

A.1 Single Rate Three Colour Marker

Token fill and packet marking algorithm

```
foreach packet
    if (TC < C-B) {
        mark packet green
        TC += B
    } else if (TE < E-B) {
        mark packet yellow
        TE += B
    } else
        mark packet red
```

References

[A+10] Mohammad Alizadeh et al. Data Center TCP (DCTCP). *Proc. ACM SIGCOMM'10, Computer Communication Review*, 40(4):63–74, October 2010.

[AKE+12] Mohammad Alizadeh, Abdul Kabbani, Tom Edsall, Balaji Prabhakar, Amin Vahdat, and Masato Yasuda. Less Is More: Trading a Little Bandwidth for Ultra-Low Latency in the Data Center. In *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI'12)*, April 2012.

[CW96] Costas Courcoubetis and Richard Weber. Buffer Overflow Asymptotics for a Switch Handling Many Traffic Sources. *Journal Applied Probability*, 33:886–903, 1996.

[Ear09] Philip Eardley. Metering and Marking Behaviour of PCN-Nodes. Request for Comments 5670, RFC Editor, November 2009.

[FJ93] Sally Floyd and Van Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993.

[GK99] Richard J. Gibbens and Frank P. Kelly. Resource pricing and the evolution of congestion control. *Automatica*, 35(12):1969–1985, December 1999.

[HG99] Juha Heinanen and Roch Guerin. A Single Rate Three Color Marker. Request for Comments 2697, RFC Editor, September 1999.

[IEE97] Draft Standard for Traffic Class and Dynamic Multicast Filtering Services in Bridged Local Area Networks (Draft Supplement to 802.1D). Draft standard 802.1p, IEEE, 1997.

[KS01] S. Kunniyur and R. Srikant. Analysis and Design of an Adaptive Virtual Queue (AVQ) Algorithm for Active Queue Management. *Proc. ACM SIGCOMM'01, Computer Communication Review*, 31(4), October 2001.

[LBS05] Ashvin Lakshmikantha, Carolyn L. Beck, and R. Srikant. Robustness of real and virtual queue-based active queue management schemes . *IEEE/ACM Transactions on Networking*, 13(1):81–93, February 2005.

[LFE+02] Francois Le Faucheur (Ed) et al. Multi-Protocol Label Switching (MPLS) Support of Differentiated Services. Request for Comments 3270, RFC Editor, May 2002.

[Mat09] Matt Mathis. Relentless Congestion Control. In *Proc. Int'l Wkshp on Protocols for Future, Large-scale & Diverse Network Transports (PFLDNeT'09)*, May 2009.

[ML07] Michael Menth and Frank Lehrieder. Comparison of Marking Algorithms for PCN-Based Admission Control. Technical Report 437, University of Würzburg, October 2007.

[NBFB98] K. Nichols, S. Blake, F.Baker, and D. Black. Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. Request for Comments 2474, RFC Editor, December 1998.

[Wis99] Damon Wischik. *Large Deviations and Internet Congestion.* PhD dissertation, University of Cambridge, September 1999.

[Zha09] Xinyang Zhang. *Performance Evaluation of Pre-Congestion Notification.* PhD dissertation, Cornell University, January 2009.

where `B` is the packet size, `C` & `E` are the sizes of each bucket and `TC` & `TE` are their respective token levels.

In practice a token fill algorithm will also handle cases where only part of a packet will fit into the bucket by filling the bucket then if necessary adding the remainder to the other bucket. Such detail has been omitted to emphasise the primary intent of the algorithm.

Token draining algorithm

```
foreach packet
    t_now = now()
    F = R(t_now - t_previous)
    if (TC > 0)
        TC -= F
    else if (TE > 0)
        TE -= F
    t_previous = now()
```

where `R` is the committed information rate.

Equivalently, instead of running the above algorithm on every packet arrival, the if-else logic block can be invoked repeatedly every `F/R` seconds.

A.2 Threshold Marking

Token fill and packet marking algorithm To implement a threshold-marking leaky bucket the only change required to an srTCM is to reverse the order of the if-else logic in the draining algorithm as follows:

```
if (TE > 0)
    TE -= F
else if (TC > 0)
    TC -= F
```

Document history

Version	Date	Author	Details of change
00A	18 Jul 2011	Bob Briscoe	First Draft
00B	19 Jul 2011	Bob Briscoe	Completed Apps & Conclusions
01	23 Jul 2011	Bob Briscoe	Clarifications throughout. Distinguished from leaky vs. token bucket debate.
01A	04 Oct 2011	Bob Briscoe	Focused on Leaky rather than Token Bucket for clarity. Highlighted theoretical nature of "one is not enough" argument.
01B	04 Oct 2011	Bob Briscoe	Tidied up.
01C	03 Nov 2011	Bob Briscoe	Corrected implementation details and added Acknowledgements.
02	18 Nov 2011	Bob Briscoe	Issued.
03	16 Mar 2012	Bob Briscoe	Reversed Fig 5 & description.
04	21 Apr 2012	Bob Briscoe	Corrected Typos
05	04 Oct 2017	Bob Briscoe	Corrected statements about standing queues