Internet Engineering Task Force (IETF)
Request for Comments: 7806
Category: Informational
ISSN: 2070-1721

F. Baker
R. Pan
Cisco Systems

April 2016

On Queuing, Marking, and Dropping

Abstract

This note discusses queuing and marking/dropping algorithms. While these algorithms may be implemented in a coupled manner, this note argues that specifications, measurements, and comparisons should decouple the different algorithms and their contributions to system behavior.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Not all documents approved by the IESG are a candidate for any level of Internet Standard; see Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc7806.

Table of Contents

1. Introduction 2
2. Fair Queuing: Algorithms and History 3
2.1. Generalized Processor Sharing 3
2.1.1. GPS Comparisons: Transmission Quanta 4
2.1.2. GPS Comparisons: Flow Definition 4
2.1.3. GPS Comparisons: Unit of Measurement 5
2.2. GPS Approximations 5
2.2.1. Definition of a Queuing Algorithm 5
2.2.2. Round-Robin Models 6
2.2.3. Calendar Queue Models 7
2.2.4. Work-Conserving Models and Stochastic Fairness Queuing 9
2.2.5. Non-Work-Conserving Models and Virtual Clock 9
3. Queuing, Marking, and Dropping 10
3.1. Queuing with Tail Mark/Drop 11
3.2. Queuing with CoDel Mark/Drop 11
3.3. Queuing with RED or PIE Mark/Drop 11
4. Conclusion 12
5. Security Considerations 13
6. References 13
6.1. Normative References 13
6.2. Informative References 13
Acknowledgements 15
Authors' Addresses 16

1. Introduction

In the discussion of Active Queue Management (AQM), there has been discussion of the coupling of queue management algorithms such as Stochastic Fairness Queuing [SFQ], Virtual Clock [VirtualClock], or Deficit Round Robin [DRR] with mark/drop algorithms such as Controlled Delay (CoDel) [DELAY-AQM] or Proportional Integral controller Enhanced (PIE) [AQM-PIE]. In the interest of clarifying the discussion, we document possible implementation approaches to that and analyze the possible effects and side effects. The language and model derive from the Architecture for Differentiated Services [RFC2475].

This note is informational and is intended to describe reasonable possibilities without constraining outcomes. This is not so much about "right" or "wrong" as it is "what might be reasonable" and discusses several possible implementation strategies. Also, while queuing might be implemented in almost any layer, the note specifically addresses queues that might be used in the Differentiated Services Architecture and are therefore at or below the IP layer.

2. Fair Queuing: Algorithms and History

There is extensive history in the set of algorithms collectively referred to as "fair queuing". The model was initially discussed in [RFC970], which proposed it hypothetically as a solution to the TCP Silly Window Syndrome issue in BSD 4.1. The problem was that, due to a TCP implementation bug, some senders would settle into sending a long stream of very short segments, which unnecessarily consumed bandwidth on TCP and IP headers and occupied short packet buffers, thereby disrupting competing sessions. Nagle suggested that if packet streams were sorted by their source address and the sources treated in a round-robin fashion, a sender's effect on end-to-end latency and increased loss rate would primarily affect only itself. This touched off perhaps a decade of work by various researchers on what was and is termed "fair queuing", philosophical discussions of the meaning of the word "fair", operational reasons that one might want a "weighted" or "predictably unfair" queuing algorithm, and so on.

2.1. Generalized Processor Sharing

Conceptually, any fair queuing algorithm attempts to implement some approximation to the Generalized Processor Sharing [GPS] model.

The GPS model, in its essence, presumes that a set of identified data streams, called "flows", pass through an interface. Each flow has a rate when measured over a period of time; a voice session might, for example, require 64 kbit/s plus whatever overhead is necessary to deliver it, and a TCP session might have variable throughput depending on where it is in its evolution. The premise of Generalized Processor Sharing is that on all time scales, the flow occupies a predictable bit rate so that if there is enough bandwidth for the flow in the long term, it also lacks nothing in the short term. "All time scales" is obviously untenable in a packet network -- and even in a traditional Time-Division Multiplexer (TDM) circuit switch network -- because a timescale shorter than the duration of a packet will only see one packet at a time. However, it provides an ideal for other models to be compared against.

There are a number of attributes of approximations to the GPS model that bear operational consideration, including at least the transmission quanta, the definition of a "flow", and the unit of measurement. Implementation approaches have different practical impacts as well.

2.1.1. GPS Comparisons: Transmission Quanta

The most obvious comparison between the GPS model and common approximations to it is that real world data is not delivered uniformly, but in some quantum. The smallest quantum in a packet network is a packet. But quanta can be larger; for example, in video applications, it is common to describe data flow in frames per second, where a frame describes a picture on a screen or the changes made from a previous one. A single video frame is commonly on the order of tens of packets. If a codec is delivering thirty frames per second, it is conceivable that the packets comprising a frame might be sent as thirty bursts per second, with each burst sent at the interface rate of the camera or other sender. Similarly, TCP exchanges have an initial window (common values of which include 1, 2, 3, 4 [RFC3390], and 10 [RFC6928]), and there are also reports of bursts of 64 KB at the relevant Maximum Segment Size (MSS), which is to say about 45 packets in one burst, presumably coming from TCP Segment Offload ((TSO) also called TCP Offload Engine (TOE)) engines (at least one implementation is known to be able to send a burst of 256 KB). After that initial burst, TCP senders commonly send pairs of packets but may send either smaller or larger bursts [RFC5690].

2.1.2. GPS Comparisons: Flow Definition

An important engineering trade-off relevant to GPS is the definition of a "flow". A flow is, by definition, a defined data stream. Common definitions include:

- packets in a single transport layer session ("microflow"), identified by a five-tuple [RFC2990];
- packets between a single pair of addresses, identified by a source and destination address or prefix;
- packets from a single source address or prefix [RFC970];
- packets to a single destination address or prefix; and
- packets to or from a single subscriber, customer, or peer [RFC6057]. In Service Provider operations, this might be a neighboring Autonomous System; in broadband, this might be a residential customer.

The difference should be apparent. Consider a comparison between sorting by source address or destination address, to pick two examples, in the case that a given router interface has N application sessions going through it between N/2 local destinations and N remote sources. Sorting by source, or in this case by source/destination

pair, would give each remote peer an upper-bound guarantee of 1/N of the available capacity, which might be distributed very unevenly among the local destinations. Sorting by destination would give each local destination an upper-bound guarantee of 2/N of the available capacity, which might be distributed very unevenly among the remote systems and correlated sessions. Who is one fair to? In both cases, they deliver equal service by their definition, but that might not be someone else's definition.

Flow fairness, and the implications of TCP's congestion avoidance algorithms, is discussed extensively in [NoFair].

2.1.3. GPS Comparisons: Unit of Measurement

And finally, there is the question of what is measured for rate. If the only objective is to force packet streams to not dominate each other, it is sufficient to count packets. However, if the issue is the bit rate of a Service Level Agreement (SLA), one must consider the sizes of the packets (the aggregate throughput of a flow measured in bits or bytes). If predictable unfairness is a consideration, the value must be weighted accordingly.

[RFC7141] discusses measurement.

2.2. GPS Approximations

Carrying the matter further, a queuing algorithm may also be termed "work conserving" or "non work conserving". A queue in a work-conserving algorithm, by definition, is either empty, in which case no attempt is being made to dequeue data from it, or contains something, in which case the algorithm continuously tries to empty the queue. A work-conserving queue that contains queued data at an interface with a given rate will deliver data at that rate until it empties. A non-work-conserving queue might stop delivering even though it still contains data. A common reason for doing this is to impose an artificial upper bound on a class of traffic that is lower than the rate of the underlying physical interface.

2.2.1. Definition of a Queuing Algorithm

In the discussion following, we assume a basic definition of a queuing algorithm. A queuing algorithm has, at minimum:

- some form of internal storage for the elements kept in the queue;

- if it has multiple internal classifications, then it has
  - a method for classifying elements and
  - additional storage for the classifier and implied classes;
- potentially, a method for creating the queue;
- potentially, a method for destroying the queue;
- an enqueuing method for placing packets into the queue or queuing system; and
- a dequeuing method for removing packets from the queue or queuing system.

There may also be other information or methods, such as the ability to inspect the queue. It also often has inspectable external attributes, such as the total volume of packets or bytes in queue, and may have limit thresholds, such as a maximum number of packets or bytes the queue might hold.

For example, a simple FIFO queue has a linear data structure, enqueues packets at the tail, and dequeues packets from the head. It might have a maximum queue depth and a current queue depth maintained in packets or bytes.

2.2.2. Round-Robin Models

One class of implementation approaches, generically referred to as "Weighted Round Robin" (WRR), implements the structure of the queue as an array or ring of subqueues associated with flows for whatever definition of a flow is important.

The arriving packet must, of course, first be classified. If a hash is used as a classifier, the hash result might be used as an array index, selecting the subqueue that the packet will go into. One can imagine other classifiers, such as using a Differentiated Services Code Point (DSCP) value as an index into an array containing the queue number for a flow, or more complex access list implementations.

In any event, a subqueue contains the traffic for a flow, and data is sent from each subqueue in succession.

Upon entering the queue, the enqueue method places a classified packet into a simple FIFO subqueue.

On dequeue, the subqueues are searched in round-robin order, and when a subqueue is identified that contains data, the dequeue method removes a specified quantum of data from it. That quantum is at minimum a packet, but it may be more. If the system is intended to maintain a byte rate, there will be memory between searches of the excess previously dequeued.

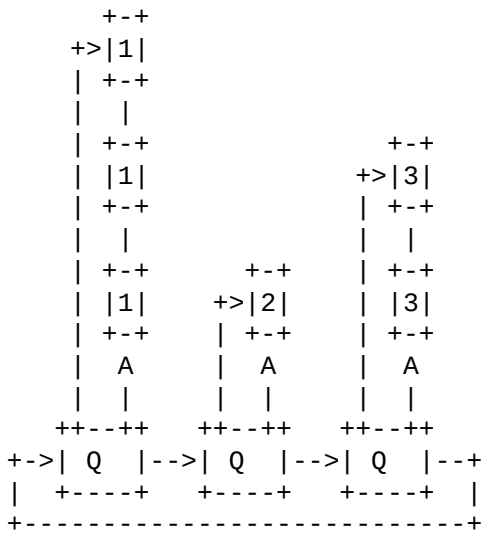

Figure 1: Round-Robin Queues

2.2.3. Calendar Queue Models

Another class of implementation approaches, generically referred to as Calendar Queue Implementations [CalendarQueue], implements the structure of the queue as an array or ring of subqueues (often called "buckets") associated with time or sequence; each bucket contains the set of packets, which may be null, intended to be sent at a certain time or following the emptying of the previous bucket. The queue structure includes a look-aside table that indicates the current depth (which is to say, the next bucket) of any given class of traffic, which might similarly be identified using a hash, a DSCP, an access list, or any other classifier. Conceptually, the queues each contain zero or more packets from each class of traffic. One is the queue being emptied "now"; the rest are associated with some time or sequence in the future. The characteristics under "load" have been investigated in [Deadline].

Upon entering the queue, the enqueue method, considering a classified packet, determines the current depth of that class with a view to scheduling it for transmission at some time or sequence in the future. If the unit of scheduling is a packet and the queuing

quantum is one packet per subqueue, a burst of packets arrives in a given flow, and if at the start the flow has no queued data, the first packet goes into the "next" queue, the second into its successor, and so on. If there was some data in the class, the first packet in the burst would go into the bucket pointed to by the look-aside table. If the unit of scheduling is time, the explanation in Section 2.2.5 might be simplest to follow, but the bucket selected will be the bucket corresponding to a given transmission time in the future. A necessary side effect, memory being finite, is that there exist a finite number of "future" buckets. If enough traffic arrives to cause a class to wrap, one is forced to drop something (tail drop).

On dequeue, the buckets are searched at their stated times or in their stated sequence, and when a bucket is identified that contains data, the dequeue method removes a specified quantum of data from it and, by extension, from the associated traffic classes. A single bucket might contain data from a number of classes simultaneously.

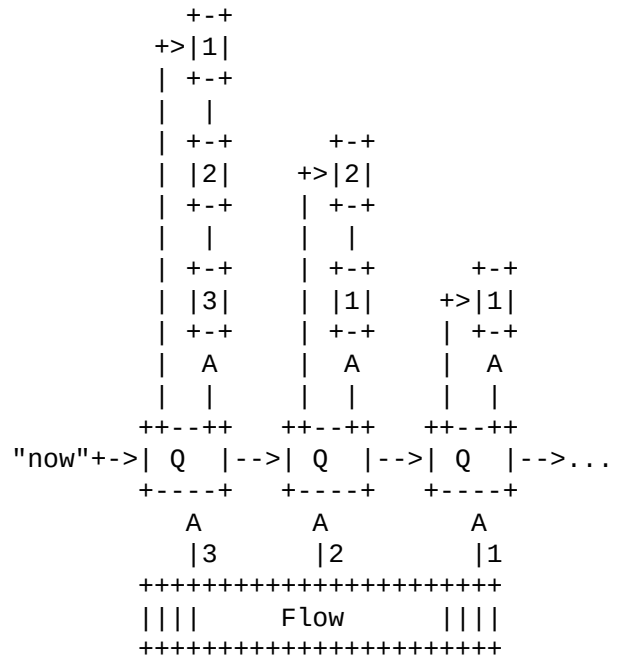

Figure 2: Calendar Queue

In any event, a subqueue contains the traffic for a point in time or a point in sequence, and data is sent from each subqueue in succession. If subqueues are associated with time, an interesting end case develops: if the system is draining a given subqueue and the time of the next subqueue arrives, what should the system do? One

potentially valid line of reasoning would have it continue delivering the data in the present queue on the assumption that it will likely trade off for time in the next. Another potentially valid line of reasoning would have it discard any waiting data in the present queue and move to the next.

2.2.4. Work-Conserving Models and Stochastic Fairness Queuing

Stochastic Fairness Queuing [SFQ] is an example of a work-conserving algorithm. This algorithm measures packets and considers a "flow" to be an equivalence class of traffic defined by a hashing algorithm over the source and destination IPv4 addresses. As packets arrive, the enqueue method performs the indicated hash and places the packet into the indicated subqueue. The dequeue method operates as described in Section 2.2.2; subqueues are inspected in round-robin sequence and a packet is removed if they contain one or more packets.

The Deficit Round Robin [DRR] model modifies the quanta to bytes and deals with variable length packets. A subqueue descriptor contains a waiting quantum (the amount intended to be dequeued on the previous dequeue attempt that was not satisfied), a per-round quantum (the subqueue is intended to dequeue a certain number of bytes each round), and a maximum to permit (some multiple of the MTU). In each dequeue attempt, the dequeue method sets the waiting quantum to the smaller of the maximum quantum and the sum of the waiting and incremental quantum. It then dequeues up to the waiting quantum (in bytes) of packets in the queue and reduces the waiting quantum by the number of bytes dequeued. Since packets will not normally be exactly the size of the quantum, some dequeue attempts will dequeue more than others, but they will over time average the incremental quantum per round if there is data present.

[SFQ] and [DRR] could be implemented as described in Section 2.2.3. The weakness of a classical WRR approach is the search time expended inspecting and not choosing sub-queues that contain no data or not enough to trigger a transmission from them.

2.2.5. Non-Work-Conserving Models and Virtual Clock

Virtual Clock [VirtualClock] is an example of a non-work-conserving algorithm. It is trivially implemented as described in Section 2.2.3. It associates buckets with intervals in time that have durations on the order of microseconds to tens of milliseconds. Each flow is assigned a rate in bytes per interval. The flow entry maintains a point in time the "next" packet in the flow should be scheduled.

On enqueue, the method determines whether the "next schedule" time is "in the past"; if so, the packet is scheduled "now", and if not, the packet is scheduled at that time. It then calculates the new "next schedule" time as the current "next schedule" time plus the length of the packet divided by the rate. If the resulting time is also in the past, the "next schedule" time is set to "now"; otherwise, it is set to the calculated time. As noted in Section 2.2.3, there is an interesting point regarding "too much time in the future"; if a packet is scheduled too far into the future, it may be marked or dropped in the AQM procedure, and if it runs beyond the end of the queuing system, may be defensively tail dropped.

On dequeue, the bucket associated with the time "now" is inspected. If it contains a packet, the packet is dequeued and transmitted. If the bucket is empty and the time for the next bucket has not arrived, the system waits, even if there is a packet in the next bucket. As noted in Section 2.2.3, there is an interesting point regarding the queue associated with "now". If a subsequent bucket, even if it is actually empty, would be delayed by the transmission of a packet, one could imagine marking the packet Explicit Congestion Notification - Congestion Experienced (ECN-CE) [RFC3168] [RFC6679] or dropping the packet.

3. Queuing, Marking, and Dropping

Queuing, marking, and dropping are integrated in any system that has a queue. If nothing else, as memory is finite, a system has to drop as discussed in Sections 2.2.3 and 2.2.5 in order to protect itself. However, host transports interpret drops as signals, so AQM algorithms use that as a mechanism to signal.

It is useful to think of the effects of queuing as a signal as well. The receiver sends acknowledgements as data is received, so the arrival of acknowledgements at the sender paces the sender at approximately the average rate it is able to achieve through the network. This is true even if the sender keeps an arbitrarily large amount of data stored in network queues and is the basis for delay-based congestion control algorithms. So, delaying a packet momentarily in order to permit another session to improve its operation has the effect of signaling a slightly lower capacity to the sender.

3.1. Queuing with Tail Mark/Drop

In the default case in which a FIFO queue is used with defensive tail drop only, the effect is to signal to the sender in two ways:

- Ack clocking, which involves pacing the sender to send at approximately the rate it can deliver data to the receiver; and
- Defensive loss, which is when a sender sends faster than available capacity (such as by probing network capacity when fully utilizing that capacity) and overburdens a queue.

3.2. Queuing with CoDel Mark/Drop

In any case wherein a queuing algorithm is used along with CoDel [DELAY-AQM], the sequence of events is that a packet is time stamped, enqueued, dequeued, compared to a subsequent reading of the clock, and then acted on, whether by dropping it, marking and forwarding it, or simply forwarding it. This is to say that the only drop algorithm inherent in queuing is the defensive drop when the queue's resources are overrun. However, the intention of marking or dropping is to signal to the sender much earlier when a certain amount of delay has been observed. In a FIFO+CoDel, Virtual Clock+CoDel, or FlowQueue-Codel [FQ-CODEL] implementation, the queuing algorithm is completely separate from the AQM algorithm. Using them in series results in four signals to the sender:

- Ack clocking, which involves pacing the sender to send at approximately the rate it can deliver data to the receiver through a queue;
- Lossless signaling that a certain delay threshold has been reached, if ECN [RFC3168] [RFC6679] is in use;
- Intentional signaling via loss that a certain delay threshold has been reached, if ECN is not in use; and
- Defensive loss, which is when a sender sends faster than available capacity (such as by probing network capacity when fully utilizing that capacity) and overburdens a queue.

3.3. Queuing with RED or PIE Mark/Drop

In any case wherein a queuing algorithm is used along with PIE [AQM-PIE], Random Early Detection (RED) [RFC7567], or other such algorithms, the sequence of events is that a queue is inspected, a packet is dropped, marked, or left unchanged, enqueued, dequeued, compared to a subsequent reading of the clock, and then forwarded on.

This is to say that the AQM Mark/Drop Algorithm precedes enqueue; if it has not been effective and as a result the queue is out of resources anyway, the defensive drop algorithm steps in, and failing that, the queue operates in whatever way it does. Hence, in a FIFO+PIE, SFQ+PIE, or Virtual Clock+PIE implementation, the queuing algorithm is again completely separate from the AQM algorithm. Using them in series results in four signals to the sender:

- Ack clocking, which involves pacing the sender to send at approximately the rate it can deliver data to the receiver through a queue;

- Lossless signaling that a queue depth that corresponds to a certain delay threshold has been reached, if ECN is in use;

- Intentional signaling via loss that a queue depth that corresponds to a certain delay threshold has been reached, if ECN is not in use; and

- Defensive loss, which is when a sender sends faster than available capacity (such as by probing network capacity when fully utilizing that capacity) and overburdens a queue.

4. Conclusion

To summarize, in Section 2, implementation approaches for several classes of queuing algorithms were explored. Queuing algorithms such as SFQ, Virtual Clock, and FlowQueue-Codel [FQ-CODEL] have value in the network in that they delay packets to enforce a rate upper bound or to permit competing flows to compete more effectively. ECN marking and loss are also useful signals if used in a manner that enhances TCP / Steam Control Transmission Protocol (SCTP) operation or restrains unmanaged UDP data flows.

Conceptually, queuing algorithms and mark/drop algorithms operate in series (as discussed in Section 3), not as a single algorithm. The observed effects differ: defensive loss protects the intermediate system and provides a signal, AQM mark/drop works to reduce mean latency, and the scheduling of flows works to modify flow interleave and acknowledgement pacing. Certain features like flow isolation are provided by fair-queuing-related designs but are not the effect of the mark/drop algorithm.

There is value in implementing and coupling the operation of both queuing algorithms and queue management algorithms, and there is definitely interesting research in this area, but specifications, measurements, and comparisons should decouple the different algorithms and their contributions to system behavior.

5. Security Considerations

This memo adds no new security issues; it observes implementation strategies for Diffserv implementation.

6. References

6.1. Normative References

[RFC2475] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., and W. Weiss, "An Architecture for Differentiated Services", RFC 2475, DOI 10.17487/RFC2475, December 1998, <http://www.rfc-editor.org/info/rfc2475>.

6.2. Informative References

[AQM-PIE] Pan, R., Natarajan, P., and F. Baker, "PIE: A Lightweight Control Scheme To Address the Bufferbloat Problem", Work in Progress, draft-ietf-aqm-pie-06, April 2016.

[CalendarQueue] Brown, R., "Calendar queues: a fast 0(1) priority queue implementation for the simulation event set problem", Communications of the ACM Volume 21, Issue 10, pp. 1220-1227, DOI 10.1145/63039.63045, October 1988, <http://dl.acm.org/citation.cfm?id=63045>.

[Deadline] Kruk, L., Lohoczky, J., Ramanan, K., and S. Shreve, "Heavy Traffic Analysis For EDF Queues With Reneging", The Annals of Applied Probability Volume 21, Issue No. 2, pp. 484-545, DOI 10.1214/10-AAP681, 2011, <http://www.math.cmu.edu/users/shreve/Reneging.pdf>.

[DELAY-AQM] Nichols, K., Jacobson, V., McGregor, A., and J. Iyengar, "Controlled Delay Active Queue Management", Work in Progress, draft-ietf-aqm-codel-03, March 2016.

[DRR] Shreedhar, M. and G. Varghese, "Efficient fair queuing using deficit round-robin", IEEE/ACM Transactions on Networking Volume 4, Issue 3, pp. 375-385, DOI 10.1109/90.502236, June 1996, <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=502236>.

[FQ-CODEL] Hoeiland-Joergensen, T., McKenney, P., Taht, D., Gettys, J., and E. Dumazet, "The FlowQueue-CoDel Packet Scheduler and Active Queue Management Algorithm", Work in Progress, draft-ietf-aqm-fq-codel-06, March 2016.

[GPS] Demers, A., University of California, Berkeley, and Xerox PARC, "Analysis and Simulation of a Fair Queueing Algorithm", ACM SIGCOMM Computer Communication Review, Volume 19, Issue 4, pp. 1-12, DOI 10.1145/75247.75248, September 1989, <http://blizzard.cs.uwaterloo.ca/keshav/home/Papers/data/89/fq.pdf>.

[NoFair] Briscoe, B., "Flow rate fairness: dismantling a religion", ACM SIGCOMM Computer Communication Review, Volume 37, Issue 2, pp. 63-74, DOI 10.1145/1232919.1232926, April 2007, <http://dl.acm.org/citation.cfm?id=1232926>.

[RFC970] Nagle, J., "On Packet Switches With Infinite Storage", RFC 970, DOI 10.17487/RFC0970, December 1985, <http://www.rfc-editor.org/info/rfc970>.

[RFC2990] Huston, G., "Next Steps for the IP QoS Architecture", RFC 2990, DOI 10.17487/RFC2990, November 2000, <http://www.rfc-editor.org/info/rfc2990>.

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <http://www.rfc-editor.org/info/rfc3168>.

[RFC3390] Allman, M., Floyd, S., and C. Partridge, "Increasing TCP's Initial Window", RFC 3390, DOI 10.17487/RFC3390, October 2002, <http://www.rfc-editor.org/info/rfc3390>.

[RFC5690] Floyd, S., Arcia, A., Ros, D., and J. Iyengar, "Adding Acknowledgement Congestion Control to TCP", RFC 5690, DOI 10.17487/RFC5690, February 2010, <http://www.rfc-editor.org/info/rfc5690>.

[RFC6057] Bastian, C., Klieber, T., Livingood, J., Mills, J., and R. Woundy, "Comcast's Protocol-Agnostic Congestion Management System", RFC 6057, DOI 10.17487/RFC6057, December 2010, <http://www.rfc-editor.org/info/rfc6057>.

[RFC6679] Westerlund, M., Johansson, I., Perkins, C., O'Hanlon, P., and K. Carlberg, "Explicit Congestion Notification (ECN) for RTP over UDP", RFC 6679, DOI 10.17487/RFC6679, August 2012, <http://www.rfc-editor.org/info/rfc6679>.

[RFC6928] Chu, J., Dukkipati, N., Cheng, Y., and M. Mathis, "Increasing TCP's Initial Window", RFC 6928, DOI 10.17487/RFC6928, April 2013, <http://www.rfc-editor.org/info/rfc6928>.

[RFC7141] Briscoe, B. and J. Manner, "Byte and Packet Congestion Notification", BCP 41, RFC 7141, DOI 10.17487/RFC7141, February 2014, <http://www.rfc-editor.org/info/rfc7141>.

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

[SFQ] Mckenney, P., "Stochastic Fairness Queuing", Proceedings of IEEE INFOCOM '90, Volume 2, pp. 733-740, DOI 10.1109/INFCOM.1990.91316, June 1990, <http://www2.rdrop.com/~paulmck/scalability/paper/sfq.2002.06.04.pdf>.

[VirtualClock] Zhang, L., "VirtualClock: A New Traffic Control Algorithm for Packet Switching Networks", Proceedings of the ACM Symposium on Communications Architectures and Protocols, Volume 20, DOI 10.1145/99508.99525, September 1990, <http://dl.acm.org/citation.cfm?id=99508.99525>.

Acknowledgements

This note grew out of, and is in response to, mailing list discussions in AQM, in which some have pushed an algorithm to compare to AQM marking and dropping algorithms, but which includes flow queuing.

Authors' Addresses

Fred Baker
Cisco Systems
Santa Barbara, California 93117
United States

Email: fred@cisco.com

Rong Pan
Cisco Systems
Milpitas, California 95035
United States

Email: ropan@cisco.com