< draft-ietf-aqm-fq-implementation.txt

Active Queue Management F. Baker
Internet-Draft R. Pan
Intended status: Informational Cisco Systems
Expires: March 22, 2015 September 18, 2014

On Queuing, Marking, and Dropping
draft-ietf-aqm-fq-implementation-00

Abstract

This note discusses implementation strategies for coupled queuing and mark/drop algorithms.

Status of This Memo

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

skipping to change at *page 1, line 31*

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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

This Internet-Draft will expire on March 22, 2015.

Table of Contents

1. Introduction

In the discussion of Active Queue Management, 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 CoDel [I-D.nichols-tsvwg-codel] or PIE [I-D.pan-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].

2. Fair Queuing: Algorithms and History

draft-ietf-aqm-fq-implementation-01.txt >

Active Queue Management F. Baker
Internet-Draft R. Pan
Intended status: Informational Cisco Systems
Expires: September 12, 2015 March 11, 2015

On Queuing, Marking, and Dropping
draft-ietf-aqm-fq-implementation-01

Abstract

This note discusses implementation strategies for coupled queuing and mark/drop algorithms.

Status of This Memo

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

skipping to change at *page 1, line 31*

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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

This Internet-Draft will expire on September 12, 2015.

Table of Contents

1. Introduction

In the discussion of Active Queue Management, 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 CoDel [I-D.ietf-aqm-codel] or PIE [I-D.ietf-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, intended to describe reasonable possibilities without constraining outc omes. 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, specifically the note 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 [RFC0970], 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

skipping to change at *page 3, line 32*

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 kbps 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 TDM circuit switch network - because a timescale shorter than he duration of a packet will only see one packet at a time. But 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", the unit of measurement. Implementation algorithms have different practical impacts as well.

2.1.1. GPS Comparisons: transmission quanta

skipping to change at *page 4, line 13*

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 65K bytes at the relevant MSS, which is to say about 45 packets in one burst, presumably coming from TCP Segment Offload (TSO, also called TOE) engines. After that initial burst, TCP senders commonly send pairs of packets, but may send either smaller or larger bursts [RFC5690], and the rate at which they send data is governed by the arrival rate of acknowledgements from the receiver.

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],

skipping to change at *page 5, line 5*

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.

2.1.3. GPS Comparisons: unit of measurement

And finally, there is the question of what is measured for rate. If the sole 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 an SLA, one must consider the sizes of the packets (the aggregate throughput of a flow, measured in bits or bytes). And if predictable unfairness is a consideration, the value must be weighted accordingly.

2.2. GPS Approximations

Carrying the matter further, a queuing algorithm may also be termed "Work Conserving" or "Non Work Conserving". 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 it 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 through it still

skipping to change at *page 5, line 39*

2.2.1. Definition of a queuing algorithm

There is extensive history in the set of algorithms collectively referred to as "Fair Queuing". The model was initially discussed in [RFC0970], 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

skipping to change at *page 3, line 40*

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 kbps 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 TDM circuit switch network - because a timescale shorter than the duration of a packet will only see one packet at a time. But 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", the unit of measurement. Implementation algorithms have different practical impacts as well.

2.1.1. GPS Comparisons: transmission quanta

skipping to change at *page 4, line 24*

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 65K bytes at the relevant MSS, which is to say about 45 packets in one burst, presumably coming from TCP Segment Offload (TSO, also called TOE) engines. 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],

skipping to change at *page 5, line 12*

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 sole 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 an SLA, one must consider the sizes of the packets (the aggregate throughput of a flow, measured in bits or bytes). And if predictable unfairness is a consideration, the value must be weighted accordingly.

Briscoe discusses measurement in his paper on Byte and Packet Congestion Notification [RFC7141].

2.2. GPS Approximations

Carrying the matter further, a queuing algorithm may also be termed "Work Conserving" or "Non Work Conserving". 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 it 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 through it still

skipping to change at *page 6, line 4*

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,
  - a method for classifying elements,
  - additional storage for the classifier and implied classes,
- a method for creating the queue,
- a method for destroying the queue,
- a method, called "enqueue", for placing packets into the queue or queuing system
- a method, called "dequeue", 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,

skipping to change at *page 6, line 20*

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", implements the structure of the queue as an array or ring of queues associated with flows, for whatever definition of a flow is important.

On enqueue, the enqueue function classifies a packet and places it into a simple FIFO sub-queue.

On dequeue, the sub-queues are searched in round-robin order, and when a sub-queue is identified that contains data, 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

skipping to change at *page 7, line 19*

array containing the queue number for a flow, or more complex access list implementations.

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

2.2.3. Calendar Queue Models

Another class of implementation approaches, generically referred to as "Weighted Fair Queues" or "Calendar Queue Implementations", implements the structure of the queue as an array or ring of queues (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.

On enqueue, the enqueue function classifies a packet and 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 sub-queue, a burst of packets arrives in a given flow, and 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

skipping to change at *page 10, line 43*

- Ack Clocking, pacing the sender to send at approximately the rate it can deliver data to the receiver, and
- Defensive loss, 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 [I-D.nichols-tsvwg-codel], 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

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,
  - a method for classifying elements,
  - additional storage for the classifier and implied classes,
- potentially, a method for creating the queue,
- potentially, a method for destroying the queue,
- a method, called "enqueue", for placing packets into the queue or queuing system
- a method, called "dequeue", 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,

skipping to change at *page 6, line 31*

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", implements the structure of the queue as an array or ring of sub-queues associated with flows, for whatever definition of a flow is important.

On enqueue, the enqueue function classifies a packet and places it into a simple FIFO sub-queue.

On dequeue, the sub-queues are searched in round-robin order, and when a sub-queue is identified that contains data, 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

skipping to change at *page 7, line 39*

array containing the queue number for a flow, or more complex access list implementations.

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

2.2.3. Calendar Queue Models

Another class of implementation approaches, generically referred to as "Weighted Fair Queues" or "Calendar Queue Implementations", implements the structure of the queue as an array or ring of sub-queues (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.

On enqueue, the enqueue function classifies a packet and 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 sub-queue, a burst of packets arrives in a given flow, and 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

skipping to change at *page 11, line 15*

- Ack Clocking, pacing the sender to send at approximately the rate it can deliver data to the receiver, and
- Defensive loss, 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 [I-D.ietf-aqm-codel], 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

[I-D.hoeiland-joergensen-aqm-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, 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, 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 PIE Mark/Drop

In any case wherein a queuing algorithm is used along with PIE [I-D.pan-aqm-pie], RED, 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, 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, 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 queueing algorithms were explored. Queuing algorithms such as SFQ, Virtual Clock, and FlowQueue-Codel [I-D.hoeiland-joergensen-aqm-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/SCTP operation or restrains unmanaged UDP data flows.

Conceptually, queuing algoritms and a 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 queueing related designs, but are not the effect of the mark/drop algorithm.

The authors think highly of queuing algorithms, which can ensure certain behaviors, but in this context believe that coupling queuing and marking or dropping in an AQM (mark/drop) discussion is unwarranted and masks issues with the mark/drop algorithm in question.

5. IANA Considerations

This memo asks the IANA for no new parameters.

6. Security Considerations

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

skipping to change at *page 13, line 21*

[DRR] Microsoft Corporation and Washington University in St. Louis, "Efficient fair queueing using deficit round robin", ACM SIGCOMM 1995, October 1995, <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=502236>.

[GPS] Xerox PARC, University of California, Berkeley, and Xerox PARC, "Analysis and simulation of a fair queueing algorithm", ACM SIGCOMM 1989, September 1989, <http://blizzard.cs.uwaterloo.ca/keshav/home/Papers/data/89/fq.pdf>.

amount of delay has been observed. In a FIFO+CoDel, Virtual Clock+CoDel, or FlowQueue-Codel [I-D.ietf-aqm-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, 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, 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 [I-D.ietf-aqm-pie], RED [RFC2309], 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, 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, 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 queueing algorithms were explored. Queuing algorithms such as SFQ, Virtual Clock, and FlowQueue-Codel [I-D.ietf-aqm-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/SCTP operation or restrains unmanaged UDP data flows.

Conceptually, queuing algoritms and a 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 queueing related designs, but are not the effect of the mark/drop algorithm.

There is value in implementing and coupling the operation of both queueing 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. IANA Considerations

This memo asks the IANA for no new parameters.

6. Security Considerations

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

skipping to change at *page 13, line 36*

[DRR] Microsoft Corporation and Washington University in St. Louis, "Efficient fair queueing using deficit round robin", ACM SIGCOMM 1995, October 1995, <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=502236>.

[GPS] Xerox PARC, University of California, Berkeley, and Xerox PARC, "Analysis and simulation of a fair queueing algorithm", ACM SIGCOMM 1989, September 1989, <http://blizzard.cs.uwaterloo.ca/keshav/home/Papers/data/89/fq.pdf>.

[I-D.hoeiland-joergensen-aqm-fq-codel]
Hoeiland-Joergensen, T., McKenney, P., Taht, D., Gettys, J., and E. Dumazet, "FlowQueue-Codel", draft-hoeiland-joergensen-aqm-fq-codel-00 (work in progress), March 2014.

[I-D.nichols-tsvwg-codel]
Nichols, K. and V. Jacobson, "Controlled Delay Active Queue Management", draft-nichols-tsvwg-codel-02 (work in progress), March 2014.

[I-D.pan-aqm-pie]
Pan, R., "PIE: A Lightweight Control Scheme To Address the Bufferbloat Problem", draft-pan-aqm-pie-02 (work in progress), September 2014.

[RFC0970] Nagle, J., "On packet switches with infinite storage", RFC 970, December 1985.

[RFC2990] Huston, G., "Next Steps for the IP QoS Architecture", RFC 2990, November 2000.

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001.

[RFC3390] Allman, M., Floyd, S., and C. Partridge, "Increasing TCP's Initial Window", RFC 3390, October 2002.

skipping to change at *page 14, line 20*

Woundy, "Comcast's Protocol-Agnostic Congestion Management System", RFC 6057, December 2010.

[RFC6679] Westerlund, M., Johansson, I., Perkins, C., O'Hanlon, P., and K. Carlberg, "Explicit Congestion Notification (ECN) for RTP over UDP", RFC 6679, August 2012.

[RFC6928] Chu, J., Dukkipati, N., Cheng, Y., and M. Mathis, "Increasing TCP's Initial Window", RFC 6928, April 2013.

[SFQ] SRI International, "Stochastic Fairness Queuing", IEEE Infocom 1990, June 1990, <http://www2.rdrop.com/~paulmck/scalability/paper/sfq.2002.06.04.pdf>.

[VirtualClock]
Xerox PARC, "Virtual Clock", ACM SIGCOMM 1990, September 1990, <http://www.cs.ucla.edu/~lixia/papers/90sigcomm.pdf>.

Appendix A. Change Log

Initial Version: June 2014

[I-D.ietf-aqm-codel]
Nichols, K., Jacobson, V., McGregor, A., and J. Iyengar, "Controlled Delay Active Queue Management", draft-ietf-aqm-codel-00 (work in progress), October 2014.

[I-D.ietf-aqm-fq-codel]
Hoeiland-Joergensen, T., McKenney, P., Taht, D., Gettys, J., and E. Dumazet, "FlowQueue-Codel", draft-ietf-aqm-fq-codel-00 (work in progress), January 2015.

[I-D.ietf-aqm-pie]
Pan, R., Natarajan, P., Baker, F., and G. White, "PIE: A Lightweight Control Scheme To Address the Bufferbloat Problem", draft-ietf-aqm-pie-00 (work in progress), October 2014.

[NoFair] British Telecom, "Flow rate fairness: dismantling a religion", ACM SIGCOMM 2007, April 2007, <http://dl.acm.org/citation.cfm?id=1232926>.

[RFC0970] Nagle, J., "On packet switches with infinite storage", RFC 970, December 1985.

[RFC2309] Braden, B., Clark, D., Crowcroft, J., Davie, B., Deering, S., Estrin, D., Floyd, S., Jacobson, V., Minshall, G., Partridge, C., Peterson, L., Ramakrishnan, K., Shenker, S., Wroclawski, J., and L. Zhang, "Recommendations on Queue Management and Congestion Avoidance in the Internet", RFC 2309, April 1998.

[RFC2990] Huston, G., "Next Steps for the IP QoS Architecture", RFC 2990, November 2000.

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001.

[RFC3390] Allman, M., Floyd, S., and C. Partridge, "Increasing TCP's Initial Window", RFC 3390, October 2002.

skipping to change at *page 14, line 50*

Woundy, "Comcast's Protocol-Agnostic Congestion Management System", RFC 6057, December 2010.

[RFC6679] Westerlund, M., Johansson, I., Perkins, C., O'Hanlon, P., and K. Carlberg, "Explicit Congestion Notification (ECN) for RTP over UDP", RFC 6679, August 2012.

[RFC6928] Chu, J., Dukkipati, N., Cheng, Y., and M. Mathis, "Increasing TCP's Initial Window", RFC 6928, April 2013.

[RFC7141] Briscoe, B. and J. Manner, "Byte and Packet Congestion Notification", BCP 41, RFC 7141, February 2014.

[SFQ] SRI International, "Stochastic Fairness Queuing", IEEE Infocom 1990, June 1990, <http://www2.rdrop.com/~paulmck/scalability/paper/sfq.2002.06.04.pdf>.

[VirtualClock]
Xerox PARC, "Virtual Clock", ACM SIGCOMM 1990, September 1990, <http://www.cs.ucla.edu/~lixia/papers/90sigcomm.pdf>.

Appendix A. Change Log

Initial Version: June 2014

End of changes. 35 change blocks.

77 lines changed or deleted *102 lines changed or added*

X-Generator: pyht 0.35