                    Controlled Delay Active Queue Management
                         draft-nichols-tsvwg-codel-00

Abstract

   The "persistently full buffer" problem has been discussed in the
   IETF community since the early 80's [RFC896]. The IRTF's End-to-End
   Working Group called for the deployment of active queue management
   to solve the problem in 1998 [RFC2309]. Despite the awareness and
   recommendations, the "full buffer" problem has not gone away, but on
   the contrary has become worse as buffers have grown in size and
   proliferated and today's networks proved intractable for available
   AQM approaches. The overall problem is presently known as
   "bufferbloat"[TSVBB2011, BB2011] and has become increasingly
   important, particularly at the consumer edge.

   This document describes a recently developed AQM, Controlled Delay
   (CoDel) algorithm, which was designed to work in modern networking
   environments and can be deployed as a major part of the solution to
   bufferbloat [CODEL2012].

Status of this Memo

Table of Contents

1. Introduction

   The need for queue management has been evident for decades.
   Unfortunately, the development and deployment of effective active
   queue management has been hampered by persistent misconceptions
   about the cause and meaning of queues. Network buffers exist to
   absorb the packet bursts that occur naturally in statistically
   multiplexed networks. Short-term mismatches in traffic arrival and
   departure rates that arise from upstream resource contention,
   transport conversation startup transients and/or changes in the
   number of conversations sharing a link create queues in the buffers.
   Unfortunately, other network behavior can cause queues to fill and
   their effects aren't nearly as benign. Discussion of these issues
   and why the solution isn't just smaller buffers can be found in
   [RFC2309],[VANQ2006],[REDL1998] and [CODEL2012]. It is critical to
   understand the difference between the necessary and useful "good"
   queue and the counterproductive "bad" queue.

   Many approaches to active queue management (AQM) have been developed
   over the past two decades, but none has been widely deployed due to
   performance problems. When designed with the wrong conceptual model
   for queues, AQMs have limited operational range, require a lot of
   configuration tweaking, and frequently impair rather than improve
   performance. Today, the demands on an effective AQM are even
   greater: many network devices must work across a range of
   bandwidths, either due to link variations or due to the mobility of
   the device. CoDel has been designed to meet the following goals:

   o is parameterless — has no knobs for operators, users, or
      implementers to adjust

   o treats "good queue" and "bad queue" differently, that is, keeps
      delay low while permitting necessary bursts of traffic

   o controls delay while insensitive (or nearly so) to round trip
      delays, link rates and traffic loads; this goal is to "do no
      harm" to network traffic while controlling delay

   o adapts to dynamically changing link rates with no negative impact
      on utilization

   o is simple and efficient (can easily span the spectrum from low-
      end, linux-based access points and home routers up to high-end
      commercial router silicon)

   With no changes to parameters, we have found CoDel to work across a
   wide range of conditions, with varying links and the full range of
   terrestrial round trip times. CoDel has been implemented in Linux
   very efficiently and should lend itself to silicon implementation.

Further, CoDel is well-adapted for use in multiple queued devices due to its use of sojourn time.

Since CoDel was published, in late April of this year, a number of talented and enthusiastic implementers and experimenters have been working with CoDel with promising results. Much of this work can be located starting from: http://www.bufferbloat.net/projects/codel.

2. Conventions used in this document

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

In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying RFC-2119 significance.

In this document, the characters ">>" preceding an indented line(s) indicates a compliance requirement statement using the key words listed above. This convention aids reviewers in quickly identifying or finding the explicit compliance requirements of this RFC.

3. The Controlled Delay (CoDel) Approach

CoDel has three major innovations that distinguish it from prior AQMs: use of local queue minimum to track congestion ("bad queue"), use of an efficient single state variable representation of that tracked statistic, and the use of packet sojourn time time as the observed datum, rather than packets, bytes, or rates. The local minimum queue provides an accurate and robust measure of standing queue and has an efficient implementation since it is sufficient to keep a single state variable of how long the minimum has been above or below a target value rather than retaining all the local values to compute the minimum. By tracking the packet sojourn time in the buffer, CoDel is using the actual delay experienced by each packet, which is independent of link rate, gives superior performance to use of buffer size, and is directly related to the user-visible performance.

In addition to lending itself to an efficient single state variable implementation, use of the minimum value has important advantages in implementation. The minimum packet sojourn can only be decreased when a packet is dequeued which means that all the work of CoDel can take place when packets are dequeued for transmission and that no locks are needed in the implementation. The minimum is the only statistic with this property. The only addition to code at packet arrival is creation of a timestamp of packet arrival time. If the buffer is full when a packet arrives, the packet is dropped as usual.

3.1. Overview of CoDel's Algorithm

   To ensure that link utilization is not adversely affected, CoDel's
   design assumes that a small "target" standing queue delay (discussed
   in more detail below) is acceptable and that it is unacceptable to
   drop packets when the drop would leave the queue empty or there are
   fewer than a maximum transmission unit (MTU) worth of bytes in the
   buffer. A persistent delay above the target indicates a standing
   queue. The standing queue can be detected by tracking the (local)
   minimum queue delay packets experience. To ensure that this minimum
   value does not become stale, it has to have been experienced
   recently, which is can be determined by using an appropriate
   interval of time (discussed further below). When the queue delay
   has exceeded the target for at least an interval, a packet is
   dropped and a control law used to set the next drop time. The next
   drop time is decreased in inverse proportion to the square root of
   the number of drops since the dropping state was entered, using the
   well-known relationship of drop rate to throughput to get a linear
   change in throughput. [REDL1998, MACTCP1997] When the queue delay
   goes below target, the controller stops dropping. No drops are
   carried out if the buffer contains fewer than an MTU worth of bytes.
   Additional logic prevents re-entering the dropping state too soon
   after exiting it and resumes the dropping state at a recent control
   level, if one exists. Target and interval are constants with
   straightforward interpretations described below.

   CoDel only enters its dropping state when the local minimum sojourn
   delay has exceeded an acceptable value for standing queue for a time
   interval long enough to for normal bursts to dissipate. This ensures
   that a burst of packets will not be dropped as long as the burst can
   be cleared from the queue within a reasonable interval.

   CoDel's efficient implementation and lack of configuration are
   unique features and make it suitable to manage modern packet
   buffers. The three innovations: minimum statistic, simplified single
   state variable tracking of minimum, and use of queue sojourn time
   lead directly to these unique features. For more background and
   results on CoDel, see [CODEL2012], available on-line at
   queue.acm.org.

3.2. About the target value

   The target value constant is the maximum acceptable standing queue
   delay under a prolonged overload. Our initial focus with CoDel was
   on devices for the ordinary open internet where bottleneck standing
   queues of a few milliseconds are acceptable for ordinary internet
   traffic. We experimented with values between 1 and 20 milliseconds
   to determine a target that gives a consistently high utilization
   while controlling delay across a range of bandwidths, RTTs, and
   traffic loads. Below a target of 5ms, utilization suffers for some

conditions and traffic loads, above 5ms we saw very little or no
improvement in utilization. Thus target SHOULD be set to 5ms.

CoDel has to see sojourn times that remain above target for an
entire interval in order to enter the drop state. Any packet with a
sojourn time less than target will reset the time that the queue was
last below the target. Since internet traffic has very dynamic
characteristics, the actual sojourn delays experienced by packets
varies greatly and is often less than the target unless the overload
is excessive. When a link is not overloaded, it is not a bottleneck
and packet sojourn times will be small or nonexistent. In the usual
case, there are only one or two places along a path where packets
will encounter a bottleneck (usually at the edge), so the amount of
queuing delay experienced by a packet should be less than 10 ms even
under extremely congested conditions. Contrast this to the queuing
delays that grow to orders of seconds that have led to the
"bufferbloat" term [NETAL2010, CHARRB2007].

3.3. About the interval

The interval constant is loosely related to RTT since it is chosen
to give endpoints time to react without being so long that response
times suffer. A setting of 100ms works well across a range of RTTs
from 10ms to 1 second (excellent performance is achieved in the
range from 10 ms to 300ms). For devices intended for the normal,
terrestrial internet interval SHOULD have the value of 100ms.
Smaller values are likely to cause CoDel to over drop packets since
insufficient time is given to senders to react.

3.4. Use with multiple queues

Unlike other AQMs, CoDel is easily adapted to multiple queue
systems. With other approaches there is always a question of how to
account for the fact that each queue receives less than the full
link rate over time and usually sees a varying rate over time. This
is exactly what CoDel excels at: using a packet's sojourn time in
the buffer completely bypasses this problem. A separate CoDel
algorithm can run on each queue, but each CoDel uses the packet
sojourn time the same way a single queue CoDel does. Just as a
single queue CoDel adapts to changing link bandwidths[CODEL2012], so
do the multiple queue CoDels.

3.5. Use of stochastic bins or sub-queues to improve performance

Shortly after the release of the CoDel pseudocode, Eric Dumazet
created fq_codel, applying CoDel to each bin, or queue, used in an
SFQ (stochastic fair queuing) approach. (To understand further, see
[SFQ1990] or the linux sfq at http://linux.die.net/man/8/tc-sfq.)
Fq_codel hashes on the packet header fields to determine a specific
bin, or sub-queue, for each five-tuple flow, and runs CoDel on each

bin or sub-queue thus creating a well-mixed output flow and obviating issues of reverse path flows (including "ack compression"). Dumazet's code is part of the CeroWrt project code at the bufferbloat.net's web site.

We've experimented with an ns-2 simulator version inspired by Dumazet's approach with excellent results thus far: median queues remain small across a range of traffic patterns than includes bidirectional file transfers (that is, the same traffic sent in both directions on a link), constant bit-rate VoIP-like flows, and emulated web traffic and utilizations are consistently better than single queue CoDel, generally very close to 100%. Our version differs slightly from Dumazet's by using a packet-based round robin of the bins rather than byte-based DRR and by doing a simple drop tail when bins are full. There may be other minor differences in implementation. Andrew McGregor has an ns-3 version of fq_codel and we have heard good reports of his results. Implementers should consider using this approach if possible. As more experiments are completed, future versions of this draft should be able to include particular pseudocode for a recommended approach.

4. Annotated Pseudo-code for CoDel

What follows is the CoDel algorithm in C++-like pseudo-code. Since CoDel adds relatively little new code to a basic tail-drop fifo-queue, we've tried to highlight just these additions by presenting CoDel as a sub-class of a basic fifo-queue base class.

Implementors are strongly encouraged to also look at Eric Dumazet's Linux kernel version of CoDel - a well-written, well tested, real-world, C-based implementation. As of this writing, it is at: http://git.kernel.org/?p=linux/kernel/git/torvalds/ linux.git;a=blob_plain;f=net/sched/sch_codel.c;hb=HEAD

This code is open-source with a dual BSD/GPL license:

Codel - The Controlled-Delay Active Queue Management algorithm

Copyright (C) 2011-2012 Kathleen Nichols <nichols@pollere.com>

 Redistribution and use in source and binary forms, with or without modification, are

permitted provided that the following conditions are met:

   * Redistributions of source code must retain the above copyright notice, this list of conditions, and the following disclaimer, without modification.

4.1. Data Types

time_t is an integer time value in units convenient for the system. Resolution to at least a millisecond is required and better resolution is useful up to the minimum possible packet time on the output link; 64- or 32-bit widths are acceptable but with 32 bits the resolution should be no finer than 2^{-16} to leave enough dynamic range to represent a wide range of queue waiting times. Narrower widths also have implementation issues due to overflow (wrapping) and underflow (limit cycles because of truncation to zero) that are not addressed in this pseudocode. The code presented here uses 0 as a flag value to indicate "no time set."

packet_t* is a pointer to a packet descriptor. We assume it has a tstamp field capable of holding a time_t and that field is available for use by CoDel (it will be set by the enque routine and used by the deque routine).

queue_t is a base class for queue objects (the parent class for codel_queue_t objects). We assume it has enque() and deque() methods that can be implemented in child classes. We assume

        it has a bytes() method that returns the current queue size
        in bytes. This can be an approximate value. The method is
        invoked in the deque() method but shouldn't require a lock
        with the enque() method.

    flag_t is a Boolean.

4.2. Per-queue state (codel_queue_t instance variables)

    time_t first_above_time; // Time when we'll declare we're above
                             // target (0 if below)
    time_t drop_next;   // Time to drop next packet
    uint32_t count;   // Packets dropped since entering drop state
    flag_t dropping;   // Equal to 1 if in drop state

4.3. Constants

    time_t target = MS2TIME(5); // Target queue delay (5 ms)
    time_t interval = MS2TIME(100); // Sliding-minimum window (100ms)
    u_int maxpacket = 512; // Maximum packet size in bytes
                          // (should use interface MTU)

4.4. Enque routine

    All the work of CoDel is done in the deque routine. The only CoDel
    addition to enque is putting the current time in the packet's tstamp
    field so that the deque routine can compute the packet's sojourn
    time.

    void codel_queue_t::enque(packet_t* pkt)
    {
        pkt->timestamp() = clock();
        queue_t::enque(pkt);
    }

4.5. Deque routine

    This is the heart of CoDel. There are two branches: In packet-
    dropping state (meaning that the queue-sojourn time has gone above
    target and hasn't come down yet), then we need to check if it's time
    to leave or if it's time for the next drop(s); if we're not in
    dropping state, then we need to decide if it's time to enter and do
    the initial drop.

    Packet* CoDelQueue::deque()
    {
        double now = clock();;
        dodequeResult r = dodeque(now);

        if (dropping_) {

```
            if (! r.ok_to_drop) {
                // sojourn time below target - leave dropping state
                dropping_ = 0;
            }
            // Time for the next drop. Drop current packet and dequeue
            // next.  If the dequeue doesn't take us out of dropping
            // state, schedule the next drop. A large backlog might
            // result in drop rates so high that the next drop should
            // happen now, hence the 'while' loop.
            while (now >= drop_next_ && dropping_) {
                drop(r.p);
                ++count_;
                r = dodeque(now);
                if (! r.ok_to_drop) {
                    // leave dropping state
                    dropping_ = 0;
                } else {
                    // schedule the next drop.
                    drop_next_ = control_law(drop_next_);
                }
            }

        // If we get here we're not in dropping state. The 'ok_to_drop'
        // return from dodeque means that the sojourn time has been
        // above 'target' for 'interval' so enter dropping state.
        } else if (r.ok_to_drop) {
            drop(r.p);
            r = dodeque(now);
            dropping_ = 1;

            // If min went above target close to when it last went
            // below, assume that the drop rate that controlled the
            // queue on the last cycle is a good starting point to
            // control it now. ('drop_next' will be at most 'interval'
            // later than the time of the last drop so 'now - drop_next'
            // is a good approximation of the time from the last drop
            // until now.)
            count_ = (count_ > 2 && now - drop_next_ < 16*interval_)?
                        count_ - 2 : 1;
            drop_next_ = control_law(now);
        }
        return (r.p);
    }
```

4.6. Helper routines

Since the degree of multiplexing and nature of the traffic sources
is unknown, CoDel acts as a closed-loop servo system that gradually
increases the frequency of dropping until the queue is controlled
(sojourn time goes below target). This is the control law that

governs the servo. It has this form because of the sqrt(p)
dependence of TCP throughput on drop probability. Note that for
embedded systems or kernel implementation, the inverse sqrt can be
computed efficiently using only integer multiplication. See Eric
Dumazet's excellent Linux CoDel implementation for example code (in
file net/sched/sch_codel.c of the kernel source for 3.5 or newer
kernels).

```
time_t codel_queue_t::control_law(time_t t)
{
    return t + interval / sqrt(count);
}
```

Next is a helper routine the does the actual packet dequeue and
tracks whether the sojourn time is above or below target and, if
above, if it has remained above continuously for at least interval.
It returns two values, a Boolean indicating if it is OK to drop
(sojourn time above target for at least interval) and the packet
dequeued.

```
typedef struct {
    packet_t* p;
    flag_t ok_to_drop;
} dodeque_result;

dodeque_result codel_queue_t::dodeque(time_t now)
{
    dodequeResult r = { NULL, queue_t::deque() };
    if (r.p == NULL) {
        // queue is empty - we can't be above target
        first_above_time_ = 0;
        return r;
    }

    // To span a large range of bandwidths, CoDel runs two
    // different AQMs in parallel. One is sojourn-time-based
    // and takes effect when the time to send an MTU-sized
    // packet is less than target.  The 1st term of the "if"
    // below does this.  The other is backlog-based and takes
    // effect when the time to send an MTU-sized packet is >=
    // target. The goal here is to keep the output link
    // utilization high by never allowing the queue to get
    // smaller than the amount that arrives in a typical
    // interarrival time (MTU-sized packets arriving spaced
    // by the amount of time it takes to send such a packet on
    // the bottleneck). The 2nd term of the "if" does this.
    time_t sojourn_time = now - r.p->tstamp;
    if (sojourn_time_ < target_ || bytes() <= maxpacket_) {
        // went below - stay below for at least interval
        first_above_time_ = 0;
```

```
        } else {
           if (first_above_time_ == 0) {
               // just went above from below. if still above at
               // first_above_time, will say it's ok to drop.
               first_above_time_ = now + interval_;
           } else if (now >= first_above_time_) {
               r.ok_to_drop = 1;
           }
       }
       return r;
   }
```

4.7. Implementation considerations

Since CoDel requires relatively little per-queue state and no direct
communication or state sharing between the enqueue and dequeue
routines, it's relatively simple to add it to almost any packet
processing pipeline, including ASIC- or NPU-based forwarding
engines. One issue to think about is dodeque's use of a 'bytes()'
function to find out about how many bytes are currently in the
queue. This value does not need to be exact. If the enqueue part of
the pipeline keeps a running count of the total number of bytes it
has put into the queue and the dequeue routine keeps a running count
of the total bytes it has removed from the queue, 'bytes()' is just
the difference between these two counters. 32 bit counters are more
than adequate. Enqueue has to update its counter once per packet
queued but it doesn't matter when (before, during or after the
packet has been added to the queue). The worst that can happen is a
slight, transient, underestimate of the queue size which might cause
a drop to be briefly deferred.

5. CoDel for specialized networks

CoDel's constants are set for use in devices in the open Internet.
They have been chosen so that a device, such as a small WiFi router,
can be sold without the need for those values to be made adjustable,
a "parameterless" implementation. CoDel is useful in environments
with significantly different characteristics from the normal
internet, for example, in switches used as a cluster interconnect
within a data center. Since cluster traffic is entirely internal to
the data center, round trip latencies are low (typically <100us) but
bandwidths are high (1-40Gbps) so it's relatively easy for the
aggregation phase of a distributed computation (e.g., the Reduce
part of a Map/Reduce) to persistently fill then overflow the modest
per-port buffering available in most high speed switches. A CoDel
configured for this environment (target and interval in the
microsecond rather than millisecond range) can minimize drops (or
ECN marks) while keeping throughput high and latency low.

Devices destined for these environments MAY have different
constants, ones that are suitable for those environments. But these
settings will cause problems such as over dropping and low
throughput if used on the open Internet so devices that allow the
CoDel constants to be configured MUST default to Internet
appropriate values given in this document.

6. Resources and Additional Information

CoDel is being implemented and tested in a range of environments.
Dave Taht has been instrumental in the integration and distribution
of bufferbloat solutions, including CoDel, and has set up a website
for CeroWRT implementers. This is an active area of work and an
excellent place to track developments. Eric Dumazet has put CoDel
into the Linux distribution. Andrew McGregor has an ns-3
implementation of both CoDel and FQ_CoDel and we have made our ns-2
implementation public. Dave Taht set up a web site and mailing list
for implementers and Eric Dumazet put CoDel into the Linux
distribution. An experiment by Stanford graduate students
successfully duplicated our published work using the linux code
which can be found at: http://
reproducingnetworkresearch.wordpress.com/2012/06/06/solving-
bufferbloat-the-codel-way/.

Our ns-2 simulations are available at http://pollere.net/CoDel.html.
We continue to do some small experiments and are periodically
updating the code. The basic algorithm of CoDel remains unchanged,
but we continue to experiment with drop interval setting when
resuming the drop state, whether to "clear out" extremely aged
packets from the queue, and other minor details. Our approach to
changes is to only make them if we are both convinced they do more
good than harm, both operationally and in the implementation. With
this in mind, some of these issues won't be settled until we can get
more experimental deployment. Our ns-2 version of the stochastic
flow binning adapted from Dumazet's fq_codel will be released before
the Vancouver IETF meeting.

7. Security Considerations

This document describes an active queue management algorithm for
implementation in networked devices. There are no specific security
exposures associated with CoDel.

8. IANA Considerations

This document does not require actions by IANA.

9. Conclusions

   CoDel is a very general, efficient, parameterless active queue
   management approach that can be applied to single or multiple
   queues. It is a critical tool in solving bufferbloat. CoDel's
   settings MAY be modified for other special-purpose networking
   applications.

   On-going projects are creating a deployable CoDel in Linux routers
   and experimenting with applying CoDel to stochastic queuing with
   very promising results.

10. References

10.1. Normative References

   [RFC896] Nagle, J., "Congestion Control in IP/TCP Internetworks",
            RFC 896, January 1984.

   [RFC2309] Braden, R. et al., "Recommendations on Queue Management
            and Congestion Avoidance in the Internet", RFC 2309, April
            1998.

10.2. Informative References

   [TSV2011] Gettys, J., "Bufferbloat: Dark Buffers in the Internet",
            presentation to Transport Area Open Meeting, http://
            www.ietf.org/proceedings/80/tsvarea.html, IETF 80, March,
            2011.

   [BB2011] Gettys, J. and K. Nichols, "Bufferbloat: Dark Buffers in
            the Internet", Communications of the ACM 9(11) pp. 57-65.

   [CODEL2012] Nichols, K. and V. Jacobson, "Controlling Queue Delay",
            Communications of the ACM Vol. 55 No. 11, July, 2012, pp.
            42-50.

   [VANQ2006] Jacobson, V., "A Rant on Queues", talk at MIT Lincoln
            Labs, Lexington, MA, July, 2006, http://www.pollere.net/
            Pdfdocs/QrantJul06.pdf

   [REDL1998] Jacobson, V., K. Nichols and K. Poduri, "RED in a
            Different Light",September, 1999, http://
            www.cnaf.infn.it/~ferrari/papers/ispn/red_light_9_30.pdf

   [NETAL2010] Kreibich, C., et. al., "Netalyzr: Illuminating the Edge
            Network", Proceedings of the Internet Measurement
            Conference, Melbourne, Australia, 2010.

   [CHARB2007] Dischinger, M., et. al, "Characterizing Residential
             Broadband Networks", Proceedings of the Internet
             Measurement Conference, San Diego, CA, 2007.

   [MACTCP1997] Mathis, M., J. Semke, J. Mahdavi, "The Macroscopic
             Behavior of the TCP Congestion Avoidance Algorithm", ACM
             SIGCOMM Computer Communications Review, Vol. 27 no. 1,
             Jan. 2007.

   [SFQ1990] McKenney, P., "Stochastic Fairness Queuing", Proceedings
             of IEEE INFOCOMM'90, San Francisco, 1990.

11. Acknowledgments

Authors' Addresses

   Kathleen Nichols
   Pollere, Inc.
   325M Sharon Park Drive #214
   Menlo Park, CA 94025

   Email: nichols@pollere.com

   Van Jacobson
   PARC, Inc.
   3333 Coyote Hill Road
   Palo Alto, CA 94304

   Email: van@parc.com