**Comments and Suggestion on CoDel Algorithm**

**Anil Agarwal**

**ViaSat Inc.**

**Jun 16, 2015**

**CoDel Algorithm Weaknesses and Suggestions**

1. **In the dropping state, when a packet is dropped, drop\_next should be updated to drop\_next + new\_interval, not now + new\_interval**.

This will result in more accurate time-keeping and more stable behavior when drop rates are high.

It’s best to modify the pseudo-code and the implementations to modify the control\_law() routine to return the next drop interval value and the caller to add it to drop\_next or now, to compute the drop\_next time.

1. When the dropping state is exited, it is not re-entered for Interval ms, even if the count value is high. No drops are performed for Interval ms. This is because the dodeque() routine sets first\_time\_above to now + Interval when the first packet with delay >= target is detected.

This results in unnecessary build-up of the queue for Interval ms during high congestion episodes.

**Suggestion: When the first packet with delay >= target is detected, set first\_time\_above to now + the last drop interval value (= Interval / sqrt(count)) rather than now + the default Interval value.**

In addition, the count value and the last drop interval value must be decreased over time in the non-dropping state. See next item on how to handle this.

1. In the non-dropping state, the count value and the drop interval remain unchanged irrespective of how long the algorithm stays in the non-dropping state. This creates an issue with how to reset it when the dropping state is next entered.

In draft-ietf-aqm-codel-01, when entering the dropping state -

 count\_ = (count\_ > 2 && now - drop\_next\_ < 8\*interval\_)?

 count\_ - 2 : 1;

This can prematurely set count to 1 under congestion conditions.

The Linux 4.0 source code has the following -

delta = vars->count - vars->lastcount;

if (delta > 1 &&

codel\_time\_before(now - vars->drop\_next,16 \* params->interval)) {

vars->count = delta;

}

else {

vars->count = 1;

}

This does not look right. At steady state, during congestion, count would typically be at some appropriate high value and the state machine would switch often between the two states and delta would be small and we will end up resetting count to a small value.

The CoDel wiki says something altogether different -

If the lowest queuing delay for the interval is less than 5 milliseconds, the packet is forwarded and interval is reset to 100 milliseconds

**Suggestion - the count value should be decreased in the non-dropping state, in a manner similar to how it is increased in the dropping state.**

In the non-dropping state, the count value should be decreased by 1 after every drop interval time, count should remain >= 1.

This can be done by maintaining an additional time variable next\_count\_dec\_time, which is set to now + current\_drop\_interval whenever the non-dropping state is entered and whenever count is decreased. count should be decreased in the non-dropping state whenever a dequeued packet has delay < target and now >= next\_count\_dec\_time.

**Suggestion - replace the logic when dropping state is entered by -**

 count\_ = (count\_ > 2 && now - drop\_next\_ < 16 \* interval\_) ?

 count\_ - 2 : 1;

This is similar to the logic in the draft-ietf-aqm-codel-01.

With the suggested changes, the count value adapts naturally to the correct value, depending on how long the logic stays in the non-dropping state. During sustained congestion, the algorithm goes back and forth between the dropping and non-dropping states, and the count value remains in a narrow range.

1. In the dropping state, whenever a packet is dropped, count is increased. **Under high drop rates, it is better not to increase count and thereby decrease the drop interval, when consecutive packets are dropped**. A state variable can used to track whether the last packet was dropped or not in the dropping state.
2. When the dropping state is entered, one packet is dropped and the next one is not. This seems incorrect in general. Depending on the value of count and the drop interval, the next packet may also need to be dropped

A loop similar to the one in the dropping state would be algorithmically correct, even if the effect is negligible in common scenarios.

Alternatively, codel\_dequeue() can be re-structured as two routines – a lower layer codel\_dequeue1() routine and higher layer codel\_dequeue() routine.

* codel\_dequeue1() should return an skb or drop one and return an indication that a packet was dropped. codel\_dequeue1() should be based on the current codel\_dequeue() routine but should not contain any loops.
* codel\_dequeue() should loop and call codel\_dequeue1() repeatedly if it indicates that a packet was dropped.

This will simplify and shrink the logic size.

1. Suggest we add a note that the count variable should not be allowed to increase beyond a ceiling value or to rollover to zero.

count should be <= Interval^2

For example, for Interval = 100,000 microseconds, count must be <= 1E10.

For a 32-bit representation of count, this is higher than the maximum 32-bit value.

Hence, the limit should be 2^31-1 and the count variable should not be allowed to roll-over.

For Interval = 50,000 microseconds, count must be <= 2.5E9.

This limit is not reached under normal circumstances but may be reached with high rate unresponsive UDP traffic.

The following diagrams summarize the suggested changes to the CoDel state machine.



**CoDel State Machine from draft-ietf-aqm-codel-0**



**Modified CoDel State Machine**

1. The target delay value is currently a fixed configured value. As we know, the value depends on the number of active TCP connections and should be a function of RTT / sqrt(numConnections) for large number of connections, but equal to RTT for a single connection scenario.

In the long run, it would be useful to dynamically estimate numConnections and adapt the target delay value accordingly. I do not have a specific suggestion for this at this time.

These suggestions are in the true spirit of the CoDel algorithm and will make CoDel more robust in a wide variety of network conditions including those with heavy congestion and/or unresponsive flows.

One of the ways to see the effects of these suggestions is to run a simulation of a single queue with CoDel, with some link rate, say 10 Mbps, and run UDP unresponsive traffic at twice the rate. Ideally, CoDel, should drop half the packets, maintain queue size close to the target delay and maintain count and current interval in a narrow range.

**Editorial Comments on draft-ietf-aqm-codel-01**

1. In the pseudo-code, it’s not clear why many of the variables are appended by the “\_” character.
2. In the routine dodeque(), clarify the meaning of bytes() and maxpacket\_.
3. In the routine dodeque(),

dodequeResult r = { NULL, queue\_t::deque() }

should be

dodequeResult r = { queue\_t::deque(), 0 };

1. In the routine dodeque(), the following comment is inaccurate -

 // One is is sojourn-time-based

 // and takes effect when the time to send an MTU-sized

 // packet is less than target

1. The pseudo-code should be updated to follow the linux implementation, which has a better structure for the routine deque(). The packet deque operation is performed in the main dequeuer routine and routine deque() is replaced by the routine codel\_should\_drop(), which returns a boolean. A better name for codel\_should\_drop() would be ok\_to\_drop().

**CoDel Linux Source Code:**

1. Rename rec\_inv\_sqrt to inv\_sqrt\_count (reciprocal inverse seems redundant).
2. Use 2^16 scale for rec\_inv\_sqrt. No need to convert and treat it as scale 2^32. Saves a few CPU cycles.
3. codel\_time\_before() is not really the right function to call when comparing time intervals, rather than time values.
4. Use “ok\_to\_drop” metaphor for all related variable and routine names. For example, rename codel\_should\_drop to codel\_ok\_to\_drop().
5. Following comment is incorrect – since the code below it has been changed from the CoDel spec.

 /\* if min went above target close to when we last went below it

 \* assume that the drop rate that controlled the queue on the

 \* last cycle is a good starting point to control it now.

 \*/

**draft-ietf-aqm-fq-codel-00**

1. Document reads like a report and a description of an implementation rather than a design specification. Perhaps, we can make it more specification-like. And separate out the implementation details and background stuff.
2. The draft rfc states - “For example, there is about an 86% probability that no more than two of the 100 VoIP sessions will be involved in any given collision”

Based on analytical equations for hash collision probabilities (I derived them independently some time ago), the probability of no collision = 90.78%, probability that no more than two of the 100 VoIP sessions will be involved in any given collision = 99.57%, probability that no more than three of the 100 VoIP sessions will be involved in any given collision = 99.99%.

1. Add a note recommending random perturbation of the hash function at startup.