Evaluation of ARED, CoDel and PIE

Jens Schwardmann, David Wagner, and Mirja Kühlewind

Institute of Communication Networks and Computer Engineering (IKR)
University of Stuttgart, Germany

Abstract. In this paper we compare the three Active Queue Managements (AQMs) Adaptive Random Early Detection (ARED), Controlled Delay (CoDel) and Proportional Integral controller Enhanced (PIE) in static as well as dynamic scenarios. We find significant issues when these algorithms are used for big Round Trip Times (RTTs) as well as a significant utilization decrease when used for high bandwidth links. When used for low and medium sized links, CoDel, PIE and ARED are suitable alike, but for corner scenarios clear recommendations can be given.

1 Introduction

In the recent years high delays and jitter were observed [7] in the Internet, originating from oversized network buffers; often referred to as Bufferbloat. Long lasting Transmission Control Protocol (TCP) transmissions fill up these buffers, delaying all traversing packets. Since these are a problem for delay-sensitive applications like VoIP, since 2012 new AQM schemes have been proposed aiming to keep queuing delays below a target delay, e.g. CoDel and PIE. This paper extends the research on performance of these algorithms by simulations using own AQM code of CoDel, PIE and ARED, a well known algorithm, fed by most realistic TCP traffic generated by Linux TCP stacks embedded in the simulation [14].

2 Related work

Although CoDel and PIE date from 2012 and 2013 respectively, there already exists some research evaluating their performance. In [8] the authors present statistical simulation results targeting the same AQM algorithms only for rather low bandwidths between 400 kbps and 5 Mbps. Moreover, they just use the standard ns-2 TCP model to generate traffic which significantly differs from TCP congestion control used in today's operating systems (e.g. regarding initial window size or proportional rate reduction [11]). In contrast, the authors of [9] use testbed measurements using the Linux kernel implementation of the aforementioned three AQMs. The measurement scenarios also just cover static scenarios while dynamics, such as a newly starting flow, are the real challenge when aiming for small queues. There is another publication to appear [10] which evaluates the overall performance of a system using not only loss-based TCP congestion

control (cubic in this case) but also delay-based TCP congestion control (vegas). Nevertheless, this research only considers a bottleneck bandwidth of 1 Mbps and again uses the ns-2 TCP traffic generator.

Evaluation of AQM algorithms is currently discussed in research and also in the AQM working group of the IETF [3]. According to current discussions, the aforementioned evaluations are not exhaustive and we aim to fill some of the remaining gaps. In contrast to existing work, we use own implementations of the AQM algorithms in our simulations combined with real Linux TCP stacks embedded in the simulation [14] for most realistic traffic generation. Moreover, we evaluate not only static scenarios with long lasting TCP flows but also dynamic ones.

3 Active Queue Management Schemes

3.1 Adaptive Random Early Detection (ARED)

The original Random Early Detection (RED) [5] algorithm calculates a drop probability p from the average queue length q_{avg}, calculated as an Exponentially Weighted Moving Average (EWMA). p is zero below a minimum threshold min_th, increases linearly to max_p at a maximum threshold max_th and equals one for $q_{avg} > max\_th$. When using RED, the mentioned parameters have to be set by the operator according to the topology properties, such as the bandwidth of the outgoing link and the expected RTT.

The extension ARED [6] initially sets all parameters automatically based on the bandwidth of the outgoing link and a reference delay value. During operation, ARED periodically adopts max_p depending on the traffic load in order to keep q_{avg} between min_th and max_th. For min_th it uses a minimum value of 5 packets to guarantee a high throughput for small bandwidths. As recommended in [6] our ARED implementation used RED in gentle-mode, where not all packets are dropped when q_{avg} is greater than max_th, but instead p is increased linearly between max_th and $2 * max\_th$. When q_{avg} is greater, all incoming packets are dropped.

3.2 Controlled Delay (CoDel)

In contrast, CoDel [12] monitors the real delay for each packet and, if all packets in a configured *observationInterval* are delayed more than the target delay, drops single selected packets. When dropping a packet, CoDel calculates the next packet drop taking into account the number of dropped packets since the first drop ($noOfDrops$) according to Equation 1.

$$nextDropTime \leftarrow lastDropTime + \frac{observationInterval}{\sqrt{noOfDrops}} \tag{1}$$

CoDel stops dropping when a packet's delay is below the target delay. Since CoDel actually measures the packets' queuing delays, in contrast to ARED and PIE packets are dropped at the head of the queue.

Our implementation is aligned to the Linux Kernel implementation [1].

3.3 Proportional Integral controller Enhanced (PIE)

PIE [13] in contrast to CoDel does not measure the queue delay but estimates it using the smoothed average of the measured draining rate. PIE uses a drop probability p like ARED but in contrast updates it just every 30 ms. As shown in Equation 2 this calculation takes into account both the current relation to the target delay as well as the trend since the last update.

$$p \leftarrow p + \alpha * (queueDelay - refDelay) + \beta * (queueDelay - lastQueueDelay) \quad (2)$$

The factors α and β increase with increasing p. Additionally, PIE avoids drops during short bursts during generally low congestion. When the estimated delay has been below half the target for two update intervals, for 100 ms PIE ignores p and enqueues all incoming packets. We decided to align our implementation to the Linux Kernel implementation [2], adopting three minor extensions not mentioned in [13]:

- When there are less than 3000 bytes in the queue, no packets are dropped.
- The drop probability p is not raised by more than 2 percentage points, except for queueing delays greater than 250 ms, for which it is increased by an additional 2 percentage points.
- If *queueDelay* and *lastQueueDelay* equal zero, p will be decreased nonlinearly by multiplying it by 0.98.

4 Simulative Evaluation

4.1 Simulation Setup and Scenarios

We use IKR SimLib [4] and its Linux Virtual Machine (VM) integration [14] for simulations to generate realistic TCP traffic. We use the Cubic congestion control of Linux kernel version 3.10.9. The modeled topology is depicted in Figure 1: A greedy TCP cubic sender has always data to send and transmits packets through a bottleneck to a TCP receiver. Bandwidth and delay of this link are fixed during each simulation, but we varied each parameter in a series of simulations. The bottleneck queue is managed by one of the candidate AQMs and the buffer size is twice the Bandwidth-Delay-Product (BDP) but minimum ten packets. Since a first bottleneck shapes the overall bandwidth of all traversing flows to the bottleneck bandwidth, there is no need to examine scenarios with several queues in a row. If not otherwise stated, we use one TCP flow and 25 Mbps links with a delay of 25 ms (50 ms RTT). All simulations ran 1010 seconds, consisting of ten seconds start up phase ignored in statistics and 1000 seconds measurements. The target delay was configured to 5 ms in all experiments.

4.2 Static scenario

We evaluated the three AQMs in the static scenario for several values of bottleneck bandwidth, RTT and number of flows regarding average delay but also bottleneck utilization.

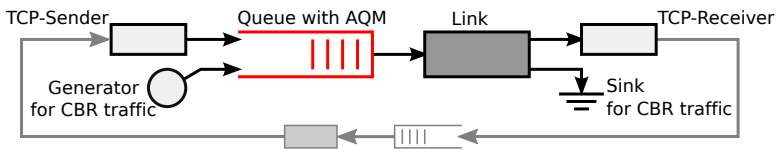

Fig. 1: Simulation Scenario

Table 1: Delay d in ms and link utilization u in % in static scenarios

(a) for different bandwidths (in Mbps)

bw	CoDel d	CoDel u	PIE d	PIE u	ARED d	ARED u
1	22.24	99.8	16.15	99.7	94.69	100
2	12.07	99.8	7.03	97.8	45.05	100
5	2.87	96.0	5.57	96.7	16.13	99.5
10	2.59	95.9	3.34	95.7	7.01	98.2
35	1.01	93.9	1.56	94.1	1.18	92.6
100	0.90	93.8	1.31	93.7	0.71	86.7
mean	4.95	96.0	4.62	95.7	18.06	94.9

(b) for different RTTs (in ms)

RTT	CoDel d	CoDel u	PIE d	PIE u	ARED d	ARED u
5	4.66	100	5.00	100	4.66	100
10	3.48	100	4.70	99.8	4.51	99.8
20	2.46	99.0	3.51	98.7	3.44	99.1
50	2.23	95.9	2.21	95.1	1.91	94.1
100	0.85	90.6	1.47	89.9	1.26	81.7
200	0.75	69.6	1.26	85.1	1.30	72.6
mean	2.19	92.0	2.83	94.7	2.51	91.4

(c) for different numbers of TCP flows

No of flows	CoDel d	CoDel u	PIE d	PIE u	ARED d	ARED u
1	2.23	95.8	2.21	95.1	1.91	94.1
2	2.73	99.2	3.40	98.7	3.11	98.0
5	3.28	98.9	4.97	99.8	5.05	99.5
10	5.16	99.8	5.18	99.9	7.62	99.9
20	7.09	99.9	4.99	100	10.45	99.9
mean	4.45	98.9	4.30	98.9	6.18	98.6

For varied bandwidth, the results are given in Table 1a. For low bandwidths PIE shows the smallest mean delays, while ARED results in very high delays. For bandwidths greater than 10 Mbps the delay is acceptable for all candidates. Utilization is close to the optimum for all AQMs for low bandwidths, but for high bandwidths was significantly lower with ARED than with CoDel and PIE. For our simulations with different RTTs we found decreasing link utilization and mean delay with increasing RTT for all three AQMs, see Table 1b. There is no significant difference between the algorithms for RTTs up to 100 ms, while for 200 ms PIE achieves significantly higher utilization.

When simulating with different numbers of TCP flows, ARED and CoDel could only observe the target delay for few flows, whereas PIE always satisfied the target delay, see Table 1c.

4.3 Adaptation to Newly Starting CBR traffic

In order to evaluate the candidates' ability to adapt to changing situations, we performed simulations with five TCP flows and, starting from a random point in time, 10 Mbps of Constant Bit Rate (CBR) traffic. In such situations, two metrics can be of interest:

- the extend of the impact, i.e. the time τ the AQM needs to recover
- the severity of the impact, i.e. the maximum queuing delay occuring in consequence of such event

We measure the period τ until packet delay decreases below the reference delay for the first time after starting the CBR traffic and the maximum delay *delay_max* within that period. As the exemplary traces shown in Figure 2 indicate, we found significant differences as shown in Table 2.

Table 2: Measured mean and standard deviation for τ and *delay_max* in milliseconds

	CoDel	PIE	ARED
τ mean	412	869	293
τ standard deviation	64	148	38
delay_max mean	30.3	41.0	39.6
delay_max standard deviation	6.0	3.8	4.0

On one hand, the mean reaction time τ of ARED, 0.29 s, is lower than CoDel's, 0.41 s, and by far lower than PIE's, 0.87 s. Although we executed just ten runs of this simulation, the derived standard deviation suggests a statistically significant advantage of ARED and CoDel with this respect. On the other hand, the emerged discrepancy from the configured target delay, i.e. *delay_max* in that phase, is much higher for PIE and ARED than for CoDel. The average for ten runs is 41.0 ms and 39.6 ms for PIE and ARED, but 30.3 ms for CoDel. Again, the derived standard deviation indicates a statistically significant advantage of CoDel with this respect.

5 Conclusion and Outlook

We evaluated the robustness of the three AQM algorithms CoDel, PIE and ARED for various static and dynamic scenarios. For low bandwidth links, PIE achieves significantly lower delays than CoDel and ARED. For high RTTs, utilization decreases for all candidates but PIE performs clearly best. When there are many flows, only PIE still keeps the target delay. In dynamic scenarios, CoDel achieves lower maximum delay than the other candidates. Moreover, CoDel and ARED recover significantly faster from changes in the traffic offer than PIE does. Overall, CoDel, PIE and ARED are suitable alike for most scenarios, but

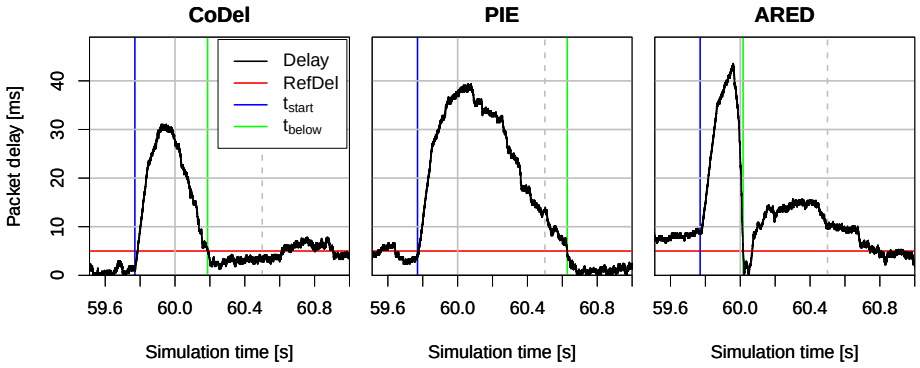

Fig. 2: Transient period with starting CBR traffic at 59.77 s simulation time.

for corner scenarios clear recommendations can be given. To show robustness and estimate performance for deployment in real Internet, it is still necessary to evaluate a broader set of scenarios, in particular including other TCP congestion control algorithms and including more traffic patterns such as web traffic.

References

[1] Codel linux kernel implementation, `https://git.kernel.org/cgit/linux/kernel/git/stable/linux-stable.git/tree/include/net/codel.h`

[2] Pie linux kernel implementation, `ftp://ftpeng.cisco.com/pie/linux_code/pie_code/linux_code/pie.c`

[3] Charter of ietf working group active queue management and packet scheduling. `http://tools.ietf.org/wg/aqm/charters` (April 2014), retrieved at 2014-07-121

[4] Ikr simulation and emulation library. `http://www.ikr.uni-stuttgart.de/Content/IKRSimLib/` (April 2014), retrieved at 2014-05-14

[5] Floyd, S., Jacobson, V.: Random Early Detection gateways for Congestion Avoidance. IEEE/ACM Transactions on Networking pp. 397–413 (Aug 1993)

[6] Floyd, S., Gummadi, R., Shenker, S.: Adaptive red: An algorithm for increasing the robustness of red s active queue management. Tech. rep. (2001)

[7] Gettys, J., Nichols, K.: Bufferbloat: Dark buffers in the internet. Queue 9(11), 40:40–40:54 (Nov 2011)

[8] Grigorescu, E., Kulatunga, C., Fairhurst, G.: Evaluation of the impact of packet drops due to aqm over capacity limited paths. In: ICNP. pp. 1–6 (2013)

[9] Khademi, N., Ros, D., Welzl, M.: The new aqm kids on the block: An experimental evaluation of codel and pie. In: Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on. pp. 85–90 (April 2014)

[10] Kuhn, N., Lochin, E., Mehani, O.: Revisiting old friends: Is codel really achieving what red cannot? In: Arvind Krishnamurthy, S.R. (ed.) CSWS 2014, ACM SIGCOMM Capacity Sharing Workshop. ACM, Chicago, IL, USA (August 2014)

[11] Mathis, M., Dukkipati, N., Cheng, Y.: Proportional Rate Reduction for TCP. RFC 6937 (Experimental) (May 2013), `http://www.ietf.org/rfc/rfc6937.txt`

[12] Nichols, K., Jacobson, V.: Controlling queue delay. Queue 10(5), 20:20–20:34 (May 2012)

[13] Pan, R., Natarajan, P., Piglione, C., Prabhu, M.S., Subramanian, V., Baker, F., VerSteeg, B.: Pie: A lightweight control scheme to address the bufferbloat problem. In: HPSR. pp. 148–155. IEEE (2013)

[14] Werthmann, T., Kaschub, M., Kühlewind, M., Scholz, S., Wagner, D.: VMSimInt: a network simulation tool supporting integration of arbitrary kernels and application. In: SIMUTools '14: Proceedings of the 7th International ICST Conference on Simulation Tools and Techniques. ACM (March 2014)