On the Effectiveness of CoDel in Data Centers

Saad Naveed Ismail, Hasnain Ali Pirzada, Ihsan Ayyub Qazi
Computer Science Department, LUMS
Email: {14100055,15100061,ihsan.qazi}@lums.edu.pk

***Abstract*—Large-scale data center applications like web search, social networking, and recommendation systems simultaneously require high throughput for long flows, small completion times for short flows, and high burst tolerance. However, meeting these requirements is a challenging task. First, long TCP flows maintain high buffer occupancy, which increases the completion time of short flows. Second, many data center applications are architected to follow the Partition-Aggregate workflow pattern, which gives rise to synchronized flows. In the presence of long flows, this significantly degrades the performance of short flows. CoDel is a promising active queue management scheme proposed for wide area networks to address *bufferbloat*; a condition in which excessive buffering leads to high latency. In this paper, we rigorously evaluate the effectiveness of CoDel in data center environments. Our results show that CoDel significantly outperforms RED and DropTail under both incast and non-incast scenarios due to its ability to better handle bursty traffic. In particular, it reduces average flow completion times by up to 35% and 65% compared to RED and DropTail, respectively. We also suggest ways to configure CoDel for data center environments.**

I. Introduction

Large-scale online services such as web search, social networking, advertising systems, and recommendation systems generate a mix of short and long flows and require three things from the underlying data center network: high throughput for long flows, low latency for short flows, and high burst tolerance [1].

However, meeting these requirements is a challenging task. First, long-lived TCP flows deliberately fill buffers [2]. This causes the average queue length to increase which in turn increases the completion time of short flows [3]. Second, many data center applications (e.g., web search, MapReduce) have workflow patterns that lead to many-to-one communication [1], [4]. For example, under the *Partition/Aggregate* workflow pattern, a user request is partitioned amongst several worker nodes within the data center and the responses from these nodes are then combined by an aggregator node before a response is sent to the user. Such workflows result in a large number of short flows that arrive synchronously at a switch leading to incast, which significantly degrades system throughput and increases the completion time of short flows [5], [6].

Active queue management (AQM) schemes aim to achieve high throughput and low delay by pro-actively marking/dropping packets at the routers [7], [8]. However, existing AQM schemes, such as Random Early Detection (RED) [7] and Random Exponential Marking (REM) [8], do not work well when the traffic is bursty and there is low statistical multiplexing, both of which hold in data center environments (see [3] and the references therein).

Controlled Delay Management (CoDel) [9] is an AQM scheme, proposed for wide area networks, that addresses the *bufferbloat* problem; a condition in which excessive buffering in routers and switches leads to high latency. CoDel aims to eliminate standing queues that introduce large delays but do not contribute to throughput improvement. A key benefit of CoDel over AQMs like RED is the lack of need to adapt parameters based on changing traffic and network conditions. This paper studies the effectiveness of CoDel in meeting the requirements of data center applications.

Towards this end, we conduct a rigorous evaluation of CoDel under typical data center settings such as incast and non-incast scenarios [6], [10] and compare results with RED and DropTail queues. Our results show that CoDel outperforms RED and DropTail under both incast and non-incast scenarios in terms of average flow completion times (AFCT) while maintaining high link utilization. In particular, it reduces AFCTs by up to 35% and 65% compared to RED and DropTail, respectively. This happens for two key reasons: (a) CoDel accommodates packet bursts better than RED and (b) CoDel adaptively adjusts the packet dropping rate by reducing the time between drops based on the persistence of congestion. The first characteristic helps CoDel in accommodating large packet bursts that arrive under incast without reacting too aggressively to instantaneous queue buildups. The second characteristic, ensures that drops occur only when congestion is persistent, which helps in ensuring high throughput as well as in eliminating any extra delays. In addition to our insights, we also suggest ways to configure CoDel for data center environments. To our knowledge, this is the first work which systematically analyzes the performance of CoDel in data centers.

The rest of the paper is organized as follows. We discuss characteristics of data center traffic and describe AQM schemes in section II. We analyze the performance of CoDel in section III. The related work is discussed in section IV. We offer concluding remarks in section V.

II. Background: Data Center Traffic and AQMs

In this section, we first discuss the nature of data center traffic and the requirements of cloud applications from the underlying data center network. We then describe CoDel and the RED AQM.

A. Data Center Traffic

User facing and large-scale online applications like web search, recommendation systems, and social networks as well as data processing frameworks like MapReduce [4] achieve horizontal scalability by partitioning a task of responding to users amongst several machines (possibly involving multiple layers) [1].

Such a *Parition/Aggregate* application structure of data center applications often results in bursts of concurrent flows that can severely degrade network throughput, leading to the *incast* impairment [3]. Moreover, the presence of long-lived TCP flows, which keep high buffer occupancy increases the response times of latency sensitive flows. These issues lead to three key application requirements from the underlying data center network fabric. (1) High throughput for long flows, (2) small completion times for short flows, and (3) high burst tolerance to handle bursts under incast.

B. Active Queue Management Schemes

CoDel: CoDel [9] is a delay based AQM scheme that aims to keep low delays while maintaining high network throughput. Many prior AQMs, such as RED and REM, that aim for a similar goal require adaptive tuning of parameters based on network characteristics and traffic conditions, which can change over time [11]. CoDel, on the other hand, does not require tuning of parameters and can adapt to dynamically changing link rates and round-trip delays. CoDel maintains two constants namely, `target` and `interval`. `target` refers to the acceptable queueing delay and `interval` is the time over which congestion is measured and is in the order of a worst-case RTT of connections.

The basic mode of operation of CoDel is as follows: Every packet that enters the queue is given a time stamp and on leaving the queue, using the time stamp, the time taken for the packet to travel through the queue (packet sojourn time) is found [9]. If a packet takes more than a `target` amount of time, the time is noted. If more than an `interval` time passes during which all packets that arrive take more than `target` time then CoDel enters into the dropping state and a packet is subsequently dropped. Additionally, after every packet drop, a control law is used to set the time for the next packet drop according to (see Figure 1):

$$\mathrm{T}_{next} = \mathrm{T}_{now} + \frac{interval}{\sqrt{count}}, \tag{1}$$

where T_{next} is the scheduled time for the next drop, and T_{now} is the current time. Count is the number of packets dropped since entering the current dropping state. Thus, higher drops indicate that there are a lot of packets experiencing delays greater than the `target` queueing delay; signifying persistence of congestion. CoDel exits the dropping state when either there are less than MTU bytes in the queue or a packet exits the queue with a delay of less than `target`. The proposed values of `target` and `interval` are 5 ms and 100 ms, respectively, for wide area networks.

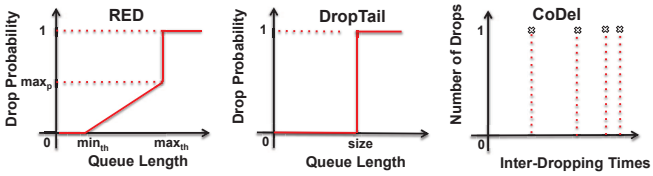
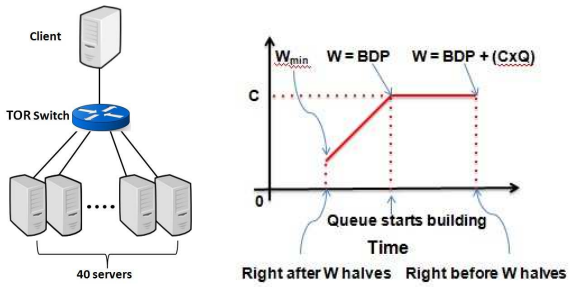


Fig. 1. The figures shows (a) drop probability of RED as a function of the queue length, (b) drop probability of DropTail as a function of queue length, and (c) inter-dropping times with CoDel under persistent congestion.

RED: The RED AQM probabilistically drops/marks packets when the average queue length exceeds a minimum threshold min_{th}. The dropping probability increases linearly from zero at min_{th} to max_p at the threshold max_{th}. When the average queue length exceeds this second threshold (max_{th}), all packets are dropped with probability 1. The dropping pattern of the three queueing schemes, DropTail, RED and CoDel, is visualized in Figure 1.

III. CoDel in Data Centers

We now analyze the performance of CoDel in data center environments. First, we study impact of `target` on network utilization and average queue length with the help of a model and ns2 simulations. We then carry out a performance evaluation of CoDel with RED and DropTail queues under incast and non-incast scenarios using ns-2 simulations.



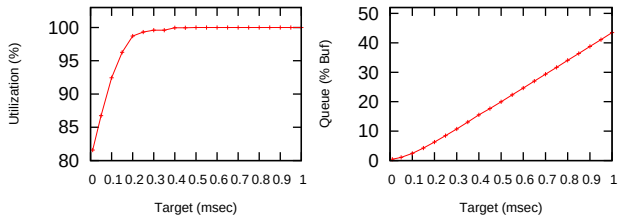
(a) Topology (b) Throughput of Link

Fig. 2. Topology of the network and throughput through bottleneck link.

Evaluation Setup: We use a single-rooted tree topology for our evaluation as used in [3], [6], [12], [13]. We use 1 Gbps interfaces, round-trip propogation delay of 300 μs (resulting in a bandwidth-delay product, or BDP, of ≈38 kB) and a static buffer size of 200 packets unless stated otherwise. We use TCP with SACK in all our simulations. We set the RTO_{min} of TCP to be 10 ms as suggested by previous studies [3], [13]. We use the Adaptive RED (ARED) algorithm [11] for our evaluation, whose implementation is available in ns-2[1]. min_{th} is 25 packets and the packet size is set to 1 kB. Since ARED tries to maintain an average queue length of $(min_{th} + max_{th})/2$, we set the max_{th} to 51 packets so that the average queue length equals the path BDP, which is the minimum buffer size needed to ensure full link utilization [2].

[1]Any mention of RED later in this paper refers to Adaptive RED (ARED).

For CoDel, we set `target` to a value equal to one RTT in order to have one BDP worth of buffering at the bottleneck. We set `interval` to 2.5×RTT as suggested in [9]. We use these values to compare the performance of CoDel against RED and DropTail under incast and non-incast scenarios.



(a) Link Utilization (b) Average Queue Length

Fig. 3. Bottleneck utilization and buffer occupancy as a function of $target \in [0.01, 1]$ ms.

A. CoDel: Choosing values for target and interval

To maximize network throughput while keeping low delays, a data center operator needs to determine a suitable value for `target` and `interval`. We now analyze the impact of these parameters and discuss the tradeoffs in the choice of their values.

Impact of target:

Variable	Meaning
Q	value of target
N	number of long-lived TCP flows
C	capacity of bottleneck link
T	time of 1 RTT
W	window size of N synchronized flows
W_{max}	window size before packet dropping state starts
W_{min}	window size after packet dropping state ends
K	bandwidth-delay product $= C \times T$
X	num. RTTs for W to go from W_{min} to K, full utilization
Y	num. RTTs where link is fully utilized, before packet drop
U	network utilization

TABLE I
MEANINGS OF VARIABLES

To understand the impact of Q (the value of `target`) on network utilization, we now present the analysis of CoDel in a simplified setting. Consider a network of N long-lived TCP flows with identical RTTs sharing a single bottleneck link of capacity C. We assume that the N flows are synchronized i.e., their window dynamics are in phase. Of course, this assumption does not always hold true, however, this is the case we care about most in data centers, where synchronized flows are common [1]. Due to this, the N synchronized flows behave like a single flow with a window size of W.

When the queueing delay exceeds `target` for all passing packets, CoDel starts dropping packets. Just before dropping a packet the aggregate window size W equals W_{max}. After the sources have responded to the packets loss, the window size reduces to W_{min}. The window becomes equal to the bandwidth-delay product $K = C \times T$ in X RTTs, at which point, the link becomes fully utilized. Then for Y RTTs, the link would remain fully utilized till the queue of size $C \times Q$ builds up. After this, a packet would be dropped and window size would half. To find the number of RTTs this makes up, we can calculate the number of packets that the window increases by. Therefore,

$$W_{max} = K + C \times Q = C \times T + C \times Q = C(T + Q) \quad (2)$$

Thus Y is:

$$RTT_B = W_{max} - K = C(T + Q) - C \times T = C \times Q \quad (3)$$

The window size would be halved to give:

$$W_{min} = W_{max}/2 = C(T + Q)/2 \quad (4)$$

Thus the number of RTTs to increase throughput to capacity is:

$$X = K - W_{min} = (C \times T) - \frac{C(T + Q)}{2} = \frac{C(T - Q)}{2} \quad (5)$$

The total number of RTTs is then given by:

$$= W_{max} - W_{min} = \frac{C(T + Q)}{2}$$

To obtain link utilization, we need to take into consideration the fraction of time for which the throughput is at capacity and the fraction of time where the throughput is not at full capacity (see Figure 2(b)). Thus, the network utilization is given by:

$$U = \left(\frac{K + W_{min}}{2 \times K}\right)\left(\frac{X}{X + Y}\right) + (1)\left(\frac{Y}{X + Y}\right) \quad (6)$$

which simplifies to

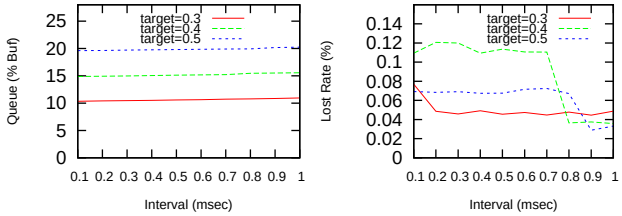
$$U = \left(\frac{3T + Q}{4T}\right)\left(\frac{T - Q}{T + Q}\right) + \left(\frac{2Q}{T + Q}\right) \quad (7)$$

Therefore, for $\beta = 1/2$ as in TCP, if $Q = T$, then $U = 1$ and when $Q = T/2$ then $U = 23/24 (\approx 95.83\%)$.

Figure 3(a) shows link utilization as a function of `target`. Observe that utilization increases with `target` until the latter becomes greater than $T = 300\,\mu$s. This happens because a single TCP flow (or N synchronized TCP flows) require at least one RTT worth of buffer at the bottleneck to avoid any throughput loss [2]. Note that increasing `target` beyond $300\,\mu$s only increases delay, which impacts the delay experienced by passing flows but does not contribute to improvement in throughput. When $Q = T/2$, the average utilization is ∼96%, which is more than 95.83% as given by Equation 7.

Impact of interval: We now consider the impact of varying the `interval`. We can think of `interval` as determining how quickly CoDel reacts to packets having queueing delay larger than the `target`. If `interval` is small, then after only a few packets taking more than `target` amount of time, it would start dropping (see Figure 4(a)). If `interval` is large, then only after a considerable amount of packets have exited the buffer with delays greater than `target` would it initiate the dropping state (see Figure 4(b)). Thus, to allow the

system time to react to congestion, we keep an `interval` value of 750 μs i.e., 2.5 times RTT. So from here on, the value of `target` and `interval` are fixed at 300 μs and 750 μs, respectively, unless stated otherwise.

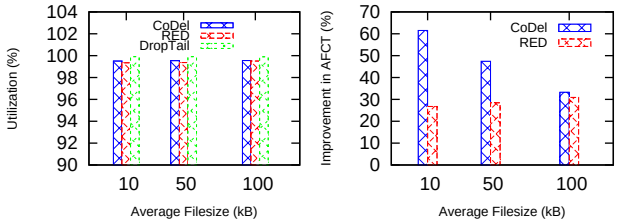


(a) Average Queue Length (b) Loss Rate

Fig. 4. Impact of $interval$ on queue length and loss rate.

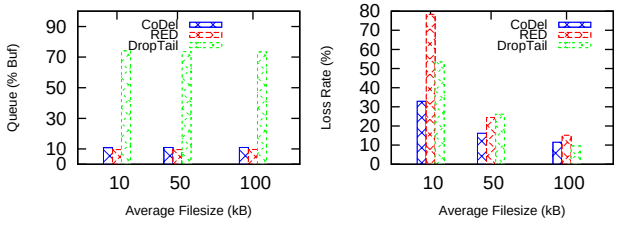
B. Incast Scenario - Varying Request Sizes

We now evaluate the performance of CoDel, RED, and DropTail under the incast scenrio [13]. Our experimental setup comprises of forty one machines that are connected to a switch with 1 Gbps links. One machine acts as a client, whereas others act as servers. The client requests S kB from each server, and the server responds with the requested data. The client waits until all responses are received before issuing another query. We always have one long-lived TCP flow active in the background; a common case in data centers [3]. We repeat this pattern several times and report the average results. Moreover, we also report results for average request sizes S of 10 kB, 50 kB and 100 kB.



(a) Link Utilization (b) Improvement in AFCT

Fig. 5. Bottleneck utilization and the improvement in AFCT (over DropTail) for CoDel, RED, and DropTail under the incast scenario with varying request sizes. One long-lived TCP flow is active at all times.



(a) Queue occupancy (b) Packet loss rate

Fig. 6. Queue occupancy and packet loss rate for CoDel, RED, and DropTail under incast with 40 senders (each generating a flow of size 100 kB). We maintain one long-lived TCP flow in the background.

Figure 5 shows the bottleneck utilization and the improvement in AFCT with CoDel, RED, and DropTail for flow sizes of 10 kB, 50 kB, and 100 kB. Observe that TCP achieves >98% link utilization under all AQMs. However, CoDel improves AFCTs by up to 62% and 35% over DropTail and RED, respectively. This happens because RED induces a higher loss rate under incast scenarios (see Figure 8). In particular, the loss rate with RED is ∼16% whereas it is ∼11% under CoDel. This suggests that CoDel is able to accomodate packet bursts better than RED without being overly aggressive in dropping packets.

Since CoDel estimates congestion over an interval which is 2.5×RTT, it allows temporary packet bursts into the queue. It is only under persistent congestion that CoDel becomes aggressive in dropping packets by reducing the inter-drop time. RED, on the other hand, starts marking packets when the average queue length exceeds min_{thresh} and increasing the dropping probability linearly as a function of the queue length. When the queue length exceeds max_{thresh}, all packets are dropped with probability one. This aggressive marking effectively reduces RED to a DropTail buffer when the queue length exceeds max_{thresh}, which increases the packet loss rate as well the timeouts. Interestingly, note that both RED and CoDel are able to maintain an average queue occupancy of less than 10%.

1) Understanding Packet Drop Behavior Under Incast: In case of incast, a large number of packet drops can occur due to buffer overflows when the number of synchronized packet arrivals exceed the buffer size. Thus, it is important to isolate the impact of packet drops due to the AQM and those due to buffer overflows. Figure 8(a) shows packet drops as a function of time for CoDel and RED under the incast scenario. In this scenario a long flow is started at time $t = 0$ s and 40 synchronized flows are initiated at time $t = 100$ ms. Observe that RED introduces significantly more packet drops than CoDel. When incast occurs, it suddenly increases the queue length causing large number of packet drops. RED's queue length quicly reaches max_{thresh} and thus drops every packet until the queue length decreases to below max_{thresh}. On the other hand, CoDel would only start dropping once the queueing delay has been greater than the `target` for an `interval` amount of time. CoDel reduces the inter-drop time as the more congeestion builds up, which helps in controlling the queue length.

2) Understanding Timeouts Under Incast: Figure 7 shows the number of flows in timeouts under the incast scenario. Observe that a few milliseconds after 100 ms flows start going into timeouts. Due to the more aggressive dropping behaviour of RED, there are more flows in timeouts for RED as compared to CoDel. Later at 150 ms as both RED and CoDel have similar dropping behavior due to having stabilized (see Figure 8(a)), it can be noted that they have a similar number of flows in timeouts. Due to more flows being in timeouts for RED than for CoDel soon after 100 ms, leads to larger AFCT for RED compared to CoDel.

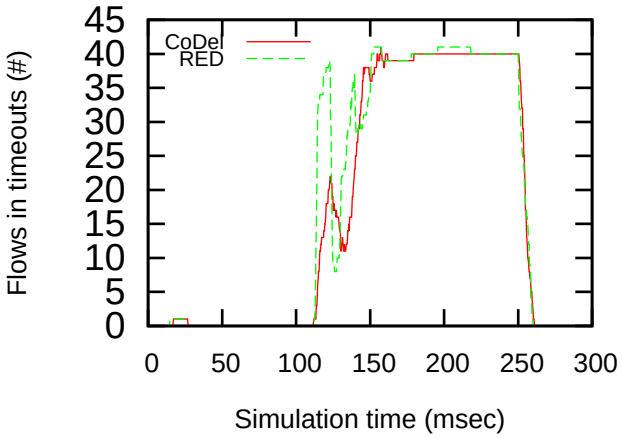
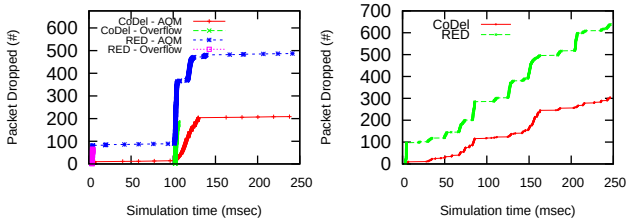


Fig. 7. The number of TCP flows in timeouts as a function of time under the incast scenario.



(a) Incast (b) Non-incast

Fig. 8. Packet drops under the incast and non-incast scenarios as a function of time. Note that the figure shows packet drops due to the AQMs as well as due to buffer overflows under incast. Since buffer overflows are negligible under the non-incast scenario, we omit them from the figure.

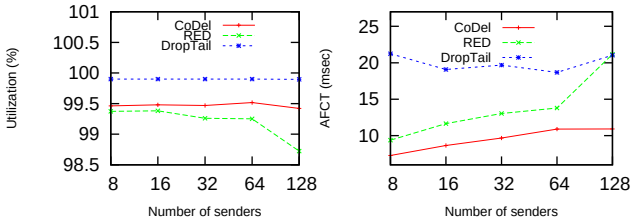
C. Incast Scenario - Varying Number of Senders

To achieve benefits of horizontal scalability, data center operators often partiation tasks into larger number of worker machines. To study such settings, we now vary the number of simultaneous senders from 8 to 128. The total request size is fixed at 1 MB, thus each sender sends 1 MB$/n$ amount of data, where n is the total number of senders. Observe that RED does not handle bursty traffic as well as CoDel or DropTail. CoDel significantly improves AFCT by upto 65% and 85% compared to RED and DropTail, respectively, as shown in Figure 9). We can see that CoDel and DropTail have larger queue lengths compared to RED, thus allowing more packets to come and handle bursts better.

D. Non-Incast Scenario

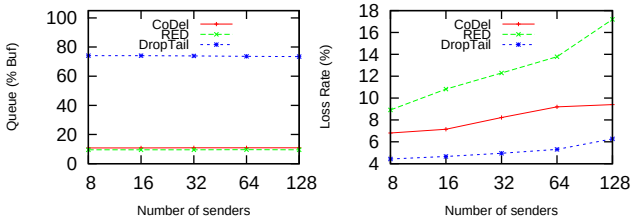
We now consider scenarios in which short flows (corresponding to query and other delay-sensitive traffic in data centers [3]) arrive at random times (with exponentially distributed inter-arrival times) while a single long-lived TCP flow is active[2]. We consider a range of offered load due to short flows and study the resulting performance. Flow sizes are drawn from the interval [2 kB, 98 kB] using a uniform

[2]Note that one long-lived TCP flow represents the 50th percentile traffic multiplexing in data center networks [3].



(a) Link Utilization (b) Improvement in AFCT

Fig. 9. Bottleneck utlization and AFCT for TCP with CoDel, RED, and DropTail under the incast scenario. 1 long lived TCP flows was active in the background.



(a) Queue occupancy (b) Packet loss rate

Fig. 10. Queue occupancy and packet loss rate with CoDel, RED, and DropTail under incast with 8, 16, 32, 64, 128 senders (each generating a flow of size 1000 kB). We maintain 1 long-lived TCP flow in the background.

distribution as done in [6]. Each simulation is run for 3 s (i.e., 10000 RTTs).

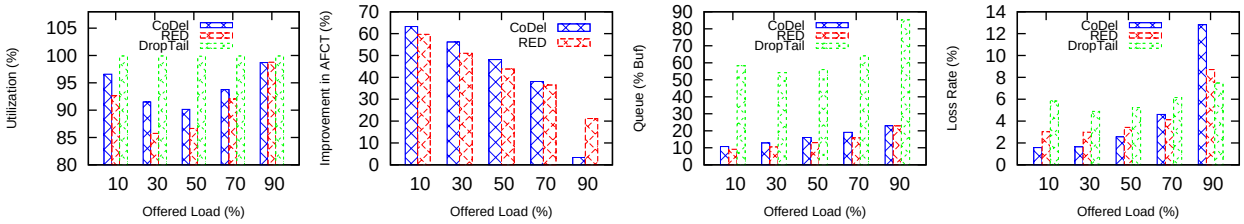
Figure 11 shows the link utilization and improvement in AFCT under the non-incast scenario. Observe that across a range of offered loads, CoDel has better link utilization (up to 6%) compared to RED. On the other hand, DropTail maintains $\sim$100% link utilization. CoDel improves the AFCT by up to 63% over DropTail and up to 7% over RED (except when the load is 90%[3]) due to lower packet loss rates.

1) Understanding Packet Drop Behavior Under Non-Incast Scenarios: Figure 8(b) shows packets with CoDel and RED under the non-incast scenario as a function of time. Observe that RED consistently drops more packets compared to CoDel. In particular, RED starts dropping packets when the queue length exceeds 25. Thus every time a short flow arrives, it starts immediately dropping probabilistically at 25 queue length and then as queue length reaches 51 it drops all packets indiscriminately. While CoDel will start dropping with inter-drop times at queue lengths of 38 and greater, i.e. it drops steadily. Therefore, we can see in Figure 8(b) as a short flow arrives RED would behave more aggressively than CoDel in maintaining queue lengths, as a result we can see that the difference in drops also increases as time passes and short flows keep arriving.

IV. Related Work

Several AQMs have been proposed in the past to address the limitations of DropTail and RED queues [9]. REM [8] replaces RED's linear marking with an exponential one. It decouples the congestion measure from the performance measure by

[3]In this case, CoDel becomes more aggressive then RED.



(a) Link Utilization (b) Improvement in AFCT (c) Queue occupancy (d) Packet loss rate

Fig. 11. Link utilization, improvement in AFCT, average queue occupancy and packet loss rate under CoDel, RED, and DropTail with one long-lived TCP flow in the background under the non-incast scenario.

capturing congestion in a dynamic variable called *price*, which depends on the number of active users and bandwidth usage. The Proportional Integral (PI) makes use of control theory to control the queues. However, PI can result in unnecessarily high loss rates in case of bursty traffic, which raises concern of its suitability under incast scenarios. CHOKe [14] improves the fairness of RED through preferential dropping of packets based on flow classification, which helps in fairly penalizing dominating flows. AVQ [15] tries to keep the queue lengths small by maintaining virtual queues at the switches.

BLUE [16] aims at reducing loss rate and queue length oscillations by using a heuristic approach. However, setting the parameters of BLUE appropriately under varying traffic patterns can be a challenge. SRED [17] is similar to RED but it estimates congestion based on current queue length and diversity of recently active flows also called the *zombie* list. FRED is a variant of RED that tracks congestion by measuring bandwidth utilization of each flow. It then drops each flow in proportion to its bandwidth utilization. All these AQMs, however, have been analyzed under wide-area networks, which are significantly different than data center networks. Hence, it is unclear how they perform under data center traffic scenarios. AQMs that build on RED will likely need significant parameter tuning under different scenarios. The *no-knobs* nature of CoDel coupled with its self adapting ability under different traffic patterns makes it easier for it to pave its way into the data center switches.

V. Conclusion

In this paper, we studied the effectiveness of CoDel in meeting the requirements of large-scale data center applications. We find that CoDel accomodates packet bursts better than existing AQMs like RED, thereby considerably improving completion times in common data center traffic scenarios such as incast. Moreover, it also improves completion times under non-incast scenarios where the traffic comprises of a mix of short and long flows. These results suggest that CoDel can be effective in data centers. In the future, we plan to test the performance of CoDel over a real testbed and under heterogeneous data center applications.

References

[1] D. Abts and B. Felderman, "A guided tour of data-center networking," *Commun. ACM*, vol. 55, no. 6, pp. 44–51, Jun. 2012.

[2] G. Appenzeller, I. Keslassy, and N. McKeown, "Sizing router buffers," in *SIGCOMM'04*.

[3] M. Alizadeh, A. Greenberg, D. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," in *SIGCOMM'10*.

[4] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[5] B. Vamanan, J. Hasan, and T. N. Vijaykumar, "Deadline-aware datacenter tcp (d2tcp)," in *SIGCOMM'12*.

[6] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowstron, "Better never than late: Meeting deadlines in datacenter networks," in *SIGCOMM'11*.

[7] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," in *IEEE/ACM Transactions on Networking, 1(4):397-413*, Aug 1993.

[8] S. Athuraliya, V. Li, S. Low, and Q. Yin, "REM: Active queue management," in *IEEE Network, 15(3):48-53*, May 2001.

[9] K. Nichols and V. Jacobson, "Controlling queue delay," *Queue*, vol. 10, no. 5, pp. 20:20–20:34, May 2012.

[10] A. Munir, I. A. Qazi, Z. A. Uzmi, A. Mushtaq, S. N. Ismail, M. S. Iqbal, and B. Khan, "Minimizing Flow Completion Times in Data Centers," in *INFOCOM'13*.

[11] S. Floyd, R. Gummadi, and S. Shenker, "Adaptive red: An algorithm for increasing the robustness of red's active queue management," Tech. Rep., 2001.

[12] H. Wu, Z. Feng, C. Guo, and Y. Zhang, "Ictcp: Incast congestion control for tcp in data center networks," in *Co-Next'10*.

[13] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. Andersen, G. Ganger, G. Gibson, and B. Mueller, "Safe and effective fine-grained tcp retransmissions for datacenter communication," in *SIGCOMM'09*.

[14] R. Pan, B. Prabhakar, and K. Psounis, "Choke - a stateless active queue management scheme for approximating fair bandwidth allocation," in *INFOCOM'00*.

[15] S. Kunniyur and R. Srikant, "Analysis and design of an adaptive virtual queue (AVQ) algorithm for active queue management," in *ACM SIGCOMM*, Aug 2001.

[16] W.-c. Feng, K. G. Shin, D. D. Kandlur, and D. Saha, "The blue active queue management algorithms," *IEEE/ACM Trans. Netw.*, vol. 10, no. 4, pp. 513–528, Aug. 2002.

[17] L. W. Teunis Ott Lakshman, T. V. Lakshman, "Sred: Stabilized red," pp. 1346–1355, 1999.