CKCD: A Fair and Low Latency Queue Control Algorithm for Heterogeneous TCP Flows

Qiong Liu*, Peng Yang†, Ming Yang*, and Li Yu*
*School of Electronic Information and Communications, Huazhong University of Science and Technology, Wuhan, China
†Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada
Email: *{liuqiong1027, m_yang, hustlyu}@hust.edu.cn, †p38yang@uwaterloo.ca

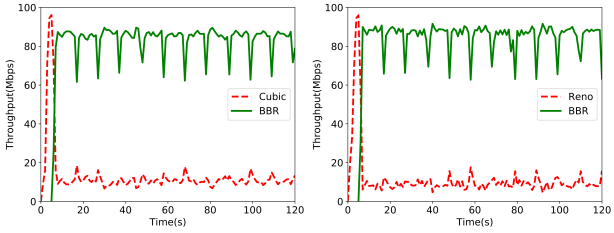
***Abstract*—The network congestion control algorithm BBR is proved to be effective in improving throughput and reducing latency. However, when BBR coexists with other loss-based congestion control algorithms (e.g., CUBIC, Reno), its flow unfairly consumes excessive bandwidth if the bottleneck node buffer size is small. In this paper, we find that such inter-flow is caused by BBR's unresponsiveness to packet loss. To address this issue, we develop an in-network queue control algorithm called Choose-Keep and Controlled-Delay (CKCD), which actively penalizes flows that are either unresponsive to packet loss or have large queuing delay. CKCD requires limited packet information and hence is stateless and with low complexity. Extensive Mininet based experimental results show that, CKCD outperforms other recent benchmarks, including DropTail, RED, CHOKe and CoDel, with improved fairness and lower latency. In particular, compared to the widely adopted DropTail, CKCD improves inter-flow fairness by up to 62%, without deteriorating intra-flow fairness. It is also able to reduce the average latency by up to 92% while guaranteeing 96% link utilization.**

***Index Terms*—Congestion control, BBR, in-network queue management, fairness, low-latency**

I. Introduction

Transmission control protocol (TCP) is the most widely used transport layer protocol since its invention. Along the development of TCP, many high-performance variants of congestion control algorithms have been proposed to address the high latency and low bandwidth utilization problem. Among others, CUBIC has been configured as the default congestion control in Linux kernels [3]. It treats packet loss as a signal of congestion, resulting in high throughput but large queuing delay [1]. Recently, a new congestion control algorithm, called BBR (Bottleneck Bandwidth and Round trip time), was proposed [2]. Instead of focusing on packet loss, BBR detects the bottleneck bandwidth (BtlBw) and round trip time (RTT) of the data pipeline, and determines the sending rate based on the measured BtlBw. In this way, it is able to accommodate the traffic dynamic without excessive backlogs on the network nodes. Consequently, maximum throughput and minimum latency are achieved simultaneously.

However, the bottleneck bandwidth share between a BBR flow and other loss-based flows (such as flows controlled by CUBIC and Reno) depends on the bottleneck node buffer size. It is shown that BBR consumes excessive bandwidth when the bottleneck node buffer size is small [4], which is also verified by our simulation results. Particularly, Fig. 1 shows the Mininet simulated bandwidth share dynamics on a bottleneck link with 100 Mbps capacity. Both CUBIC and Reno flow enjoy full bandwidth capacity before the appearance of BBR flow. Thereafter, the share of CUBIC (Reno) flow falls rapidly to less than 7 Mbps for most of the time.



(a) BBR and CUBIC (b) BBR and Reno

Fig. 1: Throughput of heterogeneous TCP flows on a bottleneck link.

To address the inter-flow fairness issue which is of importance for quality of user experience (QoE), Zhang *et al.* proposed Modest BBR, which promotes other TCP flows by adjusting the way BBR responds to retransmissions [5]. Consequently, loss-based TCP flows obtain fair share of the bottleneck bandwidth. Unfortunately, other performance metrics such as end-to-end (E2E) transmission latency and throughput were not considered. To achieve better overall performance, Winstein *et al.* developed Remy which is an optimization tool that generates congestion control algorithms for a variety of scenarios, with the objective of low latency and high throughput [6]. Other learning-based congestion control algorithms include PCC Vivace with improved throughput, delay and convergence speed [7], deep reinforcement learning based TCP-RL with dynamic initial window size and congestion control scheme [8], and DRL-CC for multi-path TCP [9]. Moreover, application layer information has also been used to reduce E2E latency, Yang *et al.* proposed a location-aware mobile edge caching scheme in order to maximize the total content hit rate and thus reduce transmission latency [10]. Since in-network queue management and sender congestion control are equally important in improving the E2E performance [3], Sasaki *et al.* investigated the combination of BBR and CoDel (Controlling Queue Delay) [12] or RED (Random Early Detection), where proactive packet dropping policy is designed [11]. However, the combination was shown to be ineffective in addressing the inter-flow fairness issue.

In this paper, we develop a new in-network queue manage-

ment called *Choose-Keep and Controlled-Delay* (CKCD) to address the inter-flow fairness problem when heterogeneous TCP flows coexist, while providing low latency and high throughput. Specifically, CKCD is designed with the following two considerations. Firstly, instead of passively dropping packets like DropTail [3], CKCD actively drops packets to penalize flows that are considered to be unresponsive to packet loss. Meanwhile, CKCD also actively drops the packets whose queuing delay exceeds a certain threshold, which is carefully designed and tuned according to the bandwidth share of different TCP flows. Extensive simulation results show that CKCD is a simple yet effective solution that not only guarantees inter-flow fairness without deteriorating the intra-flow fairness, but also achieves low transmission latency and high throughput. Main contributions of this paper are summarized as follows:

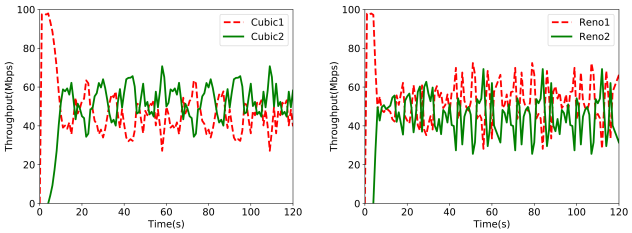
- We develop an active queue management algorithm called CKCD to address the inter-flow fairness problem when BBR flow coexists with other TCP flows. In addition to improved fairness, CKCD also achieves low E2E transmission latency and high throughput.
- CKCD requires limited packet information to make control decisions, i.e., the flowID and the packet enqueue timestamp. Therefore, CKCD is stateless and with low complexity, and can be readily implemented.
- Experimental results show that CKCD outperforms other state-of-the-art queue management benchmarks. In particular, compared to DropTail, CKCD improves inter-flow fairness by up to 62%, reduces latency by up to 92%.

The remainder of the paper is organized as follows. Section II introduces related works. In Section III, we describe the CKCD algorithm with fluid model based analysis. Section IV presents the experimental results. Finally, concluding remarks and future works are given in Section V.

II. Related Works

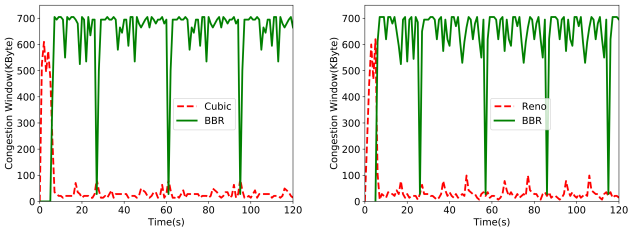
With the development of computer networks, a large number of congestion control algorithms have been proposed. Among them, congestion-based congestion control (e.g., BBR [2]) and learning-based congestion control algorithms (e.g. Remy [6], PCC Vivace [7]) are gaining increased attention. Very recently, Nie *et al.* proposed TCP-RL, which utilizes reinforcement learning techniques to dynamically configure the initial sending window size and congestion control to improve the E2E transmission performance [8]. Based on deep reinforcement learning technique, Xu *et al.* targeted on the multi-path TCP and implemented the DRL-CC control framework, which uses a single agent on an end host to dynamically and collectively perform congestion control for all multi-path subflows, so as to maximize the total utility [9]. In practice, however, traffic flows controlled by multiple TCP variants will share the network. With the prevalence of BBR, the fairness issue of bandwidth allocation among heterogeneous TCP flows becomes crucial.

To address the inter-flow fairness problem, Hock *et al.* showed that flows with larger RTT get more bandwidth if the bottleneck node buffer size is large [4]. Along this direction, Ma *et al.* proposed BBQ to continuously detect the existence



(a) Two CUBIC flows (b) Two Reno flows

Fig. 2: Throughput share of same flows on a bottleneck link.



(a) One CUBIC and one BBR (b) One Reno and one BBR

Fig. 3: The congestion window variation of the competing flows.

of the persistent queue through RTT measurements [13]. Once the measured RTT exceeds a predefined threshold, all flows perform a carefully designed probing period so that they have an equal share of the queue backlog. Unfortunately, BBQ guarantees fair share of bandwidth among different RTT flows, but the transmission latency was not addressed. Yang *et al.* proposed adaptive-BBR to strike the balance between fairness and latency [14]. In particular, adaptive-BBR determines the *pacing gain* according to the queue length, rather than blind bandwidth detection during the probeBW phase.

On the other hand, it is found that CUBIC and other loss-based traffic flows are suppressed by BBR flow if the bottleneck node buffer size is small [15]. Zhang *et al.* proposed Modest BBR which adjusts the pacing rate according to network conditions and effectively reduces the packet retransmission [5]. Modest BBR effectively improves inter-flow fairness but it does not consider the queuing delay. Further, Sasaki *et al.* propose to actively drop BBR flow packets based on CoDel or RED [11], which may not be effective in addressing the inter-flow fairness of heterogeneous TCP flows.

In contrast, the proposed CKCD focuses on a more practical scenario where heterogeneous TCP flows coexists. By carefully penalizing both unresponsive flows and large RTT flows, the inter-flow fairness is improved and, meanwhile, lower transmission latency is achieved without sacrificing throughput performance.

III. The CKCD Algorithm

In this section, we present the proposed CKCD algorithm and the fluid model based analysis.

A. Motivation and the CKCD Design

To reveal the root cause of the inter-flow fairness issue, we conduct extensive Mininet-based experiments. First, we show

the simulation results of bandwidth share when BBR, CUBIC, and Reno flows coexist on a 100 Mbps bottleneck link in Fig. 1 and Fig. 2. Fig. 2 shows that, without BBR flow, two CUBIC (or Reno) flows can obtain approximately fair share of throughput. Collectively, Fig. 1 and Fig. 2 indicate that the joining of BBR flow will unfairly affect the performance of both CUBIC and Reno.

In addition, we also simulate the congestion window (cwnd) dynamics after BBR flow joins the network. As shown in Fig. 3, at first, CUBIC (or Reno) continuously increases its cwnd to uptake more bandwidth resources. Then, BBR joins the bottleneck link, its aggressiveness rapidly fills the bottleneck node buffer, which triggers in-network queue management and results in packet loss. Consequently, CUBIC and Reno flows have to continuously half their cwnd. In contrast, BBR will not decrease its cwnd until it enters ProbeRTT phase. Eventually, both CUBIC and Reno flows are unfairly suppressed. Based on those observations, the main reason of the inter-flow fairness are summarized as follows.

- BBR is unresponsive to packet loss and it is able to keep an excessive amount of inflight packets, which consumes unfairly high link bandwidth. In contrast, loss-based congestion control algorithms, such as CUBIC and Reno, are responsive to packet loss, and they rapidly halve their cwnd, which quickly eliminates their in-network backlog.
- DropTail is the default queue management mechanism on most network devices. When the bottleneck buffer size is small, packets of the BBR flow quickly fill the buffer space. In contrast, packets from other loss-based flows will be dropped.

In practice, it is difficult to coordinate senders that are based on different congestion control algorithms. Hence, advanced in-network queue management algorithm that is able to continuously balance the backlog share of different flows is of great interest. To this end, we present our active queue management algorithm CKCD. The core idea of CKCD is to actively drop two types of packets: 1) packets from the flow that has more backlog in the queue, and 2) packets whose queuing delay exceeds a dynamically adjusted threshold. Specifically, for a set of heterogeneous TCP flows $\mathcal{I} = \{1, 2, \cdots, I\}$, denote by i the i-th TCP flow, the algorithm consists of the following three phases.

1) *Initialization*: In this phase, CKCD sets two thresholds on the buffer queue length and a time constant for average queue size calculations, a minimum threshold q_{min} that equals the bandwidth delay product (BDP), a maximum threshold q_{max} that equals three times of BDP. The underlying rationale is that the bottleneck backlog should be kept comparable to BDP. In addition, similar to CoDel, we set a queuing delay threshold D that equals 5 ms for packets from all the flows [3]. Based on the D, $d^i_{\text{th}}(t)$ of the i-th flow at time instant t will be dynamically adjusted (elaborated later). For packets from a certain flow, if their queuing delay constantly exceeds the corresponding threshold $d^i_{\text{th}}(t)$ for a monitoring duration Δ, the coming packets will be dropped. Empirically, the value of Δ is set as 100 ms [3].
2) *Enqueue Processing*: Upon the packet arrival, similar to RED, the average queue length $q_{\text{avg}}(t)$ is firstly updated based on the instantaneous queue length $q(t)$. Then, CKCD makes packet drop decisions based on $q_{\text{avg}}(t)$. If $q_{\text{avg}}(t)$ is greater than q_{max}, the packet will be dropped. If $q_{\text{avg}}(t) \in [q_{\text{min}}, q_{\text{max}}]$, CKCD will randomly choose a packet from the queue and drop packets based on the following *comparison*. If the selected packet and the coming packet are from the same flow, both of them will be dropped. Otherwise, the selected packet remain in the queue and the coming packet is dropped with probability $p_c(t) = (q_{\text{avg}}(t) - q_{\text{min}})/(q_{\text{max}} - q_{\text{min}})$. If $q_{\text{avg}}(t)$ is less than q_{min}, the coming packet enters the queue with a timestamp.
3) *Dequeue Processing*: Before forwarding a packet, CKCD calculates its queuing delay $d_i(t)$ and update the corresponding per-flow delay threshold $d^i_{\text{th}}(t)$. If the queuing delay $d_i(t)$ continuously exceeds the corresponding $d^i_{\text{th}}(t)$ for a time interval larger than Δ, the packet will be dropped. Otherwise, the packet will be forwarded.

Algorithm 1 The CKCD Queue Management Algorithm

Input: q_{min}, q_{max}, D, Δ, $\omega = 0.002$
Initialization: $q_{\text{avg}} = 0$, $d_i = 0$, $d^i_{\text{th}} = D$
for each packet arrival **do**
  $q_{\text{avg}}(t) \leftarrow (1-\omega) q_{\text{avg}}(t) + \omega q(t)$
  if $q_{\text{avg}}(t) > q_{\text{max}}$ **then**
    Drop the packet
  else if $q_{\text{min}} \le q_{\text{avg}}(t) \le q_{\text{max}}$ **then**
    Randomly select a packet from the backlog
    Drop packets based on comparison (see phase 2)
  else
    Enqueue packet with a timestamp
  end if
end for
for each packet departure **do**
  Obtain $d_i(t)$ and update $d^i_{\text{th}}(t)$
  if $d_i(t) \ge d^i_{\text{th}}(t)$ lasts longer than Δ **then**
    Drop the packet
  else
    Forward the packet
  end if
end for

The CKCD algorithm is sketched in Algorithm 1. In what follows, we describe how to obtain the per-flow delay thresholds $d^i_{\text{th}}(t)$ based on the fluid model analysis.

B. Fluid Model based Analysis

CKCD strikes the balance between inter-flow fairness and low transmission delay via dynamically configuring the delay

threshold $d^i_{\text{th}}(t)$ for each flow. Specifically, in CKCD's dequeue process, $d^i_{\text{th}}(t)$ is dynamically updated based on D, flow's virtual arrival rate $\mu_i(t)$ and ideal forwarding rate μ_f. Consider a set of I heterogeneous TCP flows that are traversing a bottleneck node with forwarding capacity C. Let $h_i(t) = b_i(t)/q(t)$ be the instantaneous buffer share ratio of the i-th flow, where $b_i(t)$ is amount of queuing packets from flow i and $q(t)$ is the total backlog (instantaneous queue length) on the network node. Then, $h_i(t)$ is equivalent to the probability that packet from the i-th flow is selected during CKCD enqueue processing.

Now we focus on the packet loss probability of each flow. Since current internet is highly reliable, packet loss due to link error is neglected. Recall that in the proposed CKCD algorithm, packets may be dropped during both enqueue and dequeue phase. Before a packet is enqueued, it may be dropped either due to CKCD enqueue processing or congestion [16]. Denote by $p^e_i(t)$ the loss probability before dequeuing the packet from the i-th flow, then we have

$$p^e_i(t) = 2h_i(t) + (1 - h_i(t))p_c(t), \tag{1}$$

where $p_c(t)$ is congestion-based dropping ratio determined by $p_c(t) = (q_{\text{avg}}(t) - q_{\text{min}})/(q_{\text{max}} - q_{\text{min}})$.

Alternatively, the survive probability can also be obtain in the following way. Let $x_i(t)$ and $d_i(t)$ be the instantaneous sending rate and queuing delay of the i-th flow, respectively. Then, we have $d_i(t) = q(t)/C$. When this packet stays in the queue, there are on average $x_i(t)d_i(t)$ packets arrive at the queue for processing. According to the second phase comparison of CKCD, the probability that a queued packet is not chosen for comparison is

$$p^b_i(t) = (1 - \frac{1}{q(t)})^{d_i(t)x_i(t)}. \tag{2}$$

Then, we have

$$1 - p^e_i(t) = (1 - h_i(t))(1 - p_c(t))p^b_i(t). \tag{3}$$

Consequently, the resulting virtual arrival rate $\mu_i(t)$ of the i-th flow is $\mu_i(t) = (1-p^e_i(t))x_i(t)$. In order to achieve inter-flow fairness among heterogeneous TCP flows, the ideal forwarding rate μ_f of each flow should be equal to C/I.

Recall that during dequeue processing, if the queuing delay $d_i(t)$ of packets from the i-th flow keeps exceeding $d^i_{\text{th}}(t)$, the packet will be dropped. Now, we determine the per-flow delay threshold $d^i_{\text{th}}(t)$ based on fluid model analysis. Specifically, the dynamics of the congestion window $w_i(t)$ can be expressed as the following non-linear differential equation [17]

$$\dot{w}_i(t) = \frac{w_i(t - \tau_i(t))}{\tau_i(t - \tau_i(t))}(\frac{1 - p_i(t)}{w_i(t)} - \frac{p_i(t)w_i(t)}{2}), \tag{4}$$

where $\tau_i(t)$ is RTT of the i-th flow, $p_i(t)$ is the total packet drop probability that equals $p^e_i(t) + p^d_i(t)$, where $p^d_i(t)$ is the drop ratio of packet of flow i at departure, and it is proportional to the queuing delay $d_i(t)$ [18]. Therefore, $p_i(t) - p^e_i(t) = kd_i(t)$, where k is a positive constant and $d_i(t)$ can be calculated by (assuming that propagation delay is negligible compared with queuing delay)

$$d_i(t) = \frac{Iw_i(t)}{C}. \tag{5}$$

Thus, the dynamics of virtual arrival rate $\dot{s}_i(t)$ is given by

$$\dot{s}_i(t) = \frac{\dot{w}_i(t)}{\tau_i(t)} = \frac{1 - p_i(t)}{\tau_i(t)^2} - \frac{1}{2}p_i(t)s_i(t)^2. \tag{6}$$

Let s^*_i be the equilibrium of arrival rate when $\dot{s}_i(t) = 0$, then we have $s^*_i = \frac{1-p^*_i}{\tau^2_i}\frac{\sqrt{2}}{\sqrt{p^*_i}} \approx \frac{1}{\tau^2_i}\frac{\sqrt{2}}{\sqrt{p^*_i}}$ [19]. With $p_i = p^e_i + kd_i$, we have $s^*_i \approx \frac{1}{\tau^2_i}\frac{\sqrt{2}}{\sqrt{\frac{kIw^*_i}{C}+p^{e*}_i}}$. Note that, at the equilibrium point, p^e_i converges to $1/I$. Meanwhile, as the target sending window size w^*_i is determined by $d^i_{\text{th}}C/I$ in (5), the target arrival rate s^*_i of flow i is obtained by

$$s^*_i = \frac{\sqrt{2}}{\tau^2_i}(kd^i_{\text{th}} + \frac{1}{I})^{-\frac{1}{2}}. \tag{7}$$

As the threshold of d^i_{th} is set as D, the sending window size $w_i(t)$ is limited by DC/I. Hence, the arrival rate $s_i(t)$ of flow i is obtained by

$$s_i(t) = \frac{w_i(t)}{\tau_i(t)} = \frac{DC}{I\tau_i(t)}. \tag{8}$$

With the virtual arrival rate $s_i(t) = \mu_i(t)$ and s^*_i equals the ideal per-flow forwarding rate μ_f, $d^i_{\text{th}}(t)$ can be expressed as

$$d^i_{\text{th}}(t) = K(\frac{\mu_i(t)}{\mu_f})^{-2} + Q, \tag{9}$$

where $K = (\frac{\sqrt{2}I}{\sqrt{k}D^2C})^{-2}$, $Q = -\frac{1}{kI}$. Thus, if the virtual transmission rate $\mu_i(t)$ of flow i is greater than the fair transmission rate μ_f, its target delay $d^i_{\text{th}}(t)$ will decrease, and more packets are dropped to penalized the i-th flow. Otherwise, its $d^i_{\text{th}}(t)$ will be increased, more packets are delivered and flow i is compensated. In this way, CKCD can achieve a balance between fairness and low latency among heterogeneous flows. Note that CKCD is a stateless algorithm that does not require any packet information, except the flowID of each incoming packet and the timestamp of each enqueued packet. The parameters used in the algorithm are shown in Table I.

TABLE I: Parameter Description

Parameter	Description
q_{avg}	average queue length
q_{min} (q_{max})	lower (higher) threshold of queue length
p_c	packet congestion-based dropping probability
d_i	actual queuing delay of packets from flow i
D	queuing delay threshold of packets from all the flows
d^i_{th}	target queuing delay of packets from flow i
Δ	delay monitoring interval

IV. Performance Evaluation

In this section, extensive Mininet-based experiments are performed to verify the effectiveness of the proposed CKCD algorithm in improving the inter-flow fairness and reducing E2E transmission latency. The simulated network topology, parameter setting, and experimental results are presented.

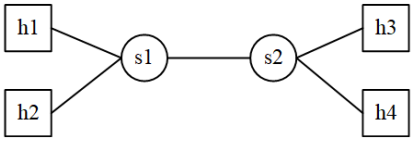


Fig. 4: The dumbbell network topology.

TABLE II: Parameter Settings

	Parameters
DropTail	buffer size: 80 packets;
CoDel	buffer size: 80 packets; Δ: 100 ms; D: 5 ms
RED	buffer size: 80 packets; bandwidth: 100 Mbps
CHOKe	buffer size: 80 packets; bandwidth: 100 Mbps
CKCD	buffer size: 80 packets; Δ: 100 ms; D: 5 ms bandwidth: 100 Mbps

A. Simulation Setup

We set up a dumbbell network topology consisting of four hosts and two switches on Mininet, as shown in Fig. 4. All the network nodes are configured to run Linux kernel 4.15. Traffic flows are generated using *iperf*[1] at h1 and h2, and delivered to h3 and h4, respectively. The package *TrafficControl*[2] is employed to configure router's queueing discipline, i.e., the queue management algorithm. Three sets of experiments with different combination of TCP flows are performed, namely, 1) one BBR flow and one CUBIC flow; 2) three flows consisting of one BBR flow, one CUBIC flow, and one Reno flow; and 3) fifteen flows with five BBR flows, five CUBIC flows, and five Reno flows. Each set of experiment lasts for 120 seconds. The bottleneck link bandwidth and propagation delay are configured as 100 Mbps and 10 ms, respectively. Then, we evaluate the inter-flow bandwidth fairness, average latency, throughput and link utilization under the same flow RTT of different queue management algorithms. The comparison benchmarks include Droptail, CoDel, RED, and CHOKe, whose parameter settings are shown in Table II. In addition, we also conduct another experiment with five BBR flows and five CUBIC flows to demonstrate intra-flow fairness of CKCD.

B. Simulation Results

1) One BBR and One CUBIC: In the first experiment, we evaluate the performance of CKCD when one BBR flow competes with one CUBIC flow. The experimental results on time-varying throughput and delay are depicted in Fig. 5(a)-(e). It is shown that both the proposed CKCD algorithm and CHOKe provide fair bandwidth share, as compared with RED, CoDel and DropTail. Yet, the E2E delay of CKCD is lower and more stable than that of CHOKe. Furthermore, we use Jain's index to explicitly characterize the fairness of different algorithms, the closer the index is to 1, the better fairness is achieved. As shown by the histogram in Fig. 5(f)(g), given the 100 Mbps link capacity, CKCD achieves 96% bandwidth utilization (measured as the summarized throughput of all the flows over the link capacity), which is similar to other benchmarks. However, CKCD achieves the lowest delay and the highest inter-flow fairness. In summary, CKCD can simultaneously satisfy bandwidth allocation fairness and low latency without sacrificing throughput (link utilization) when one BBR flow coexists with one CUBIC flow.

2) One BBR, One CUBIC and One Reno: For the second experiment with one BBR flow, one CUBIC flow and one Reno flow, the histogram of fairness, throughput and delay is shown in Fig. 6. It is shown by Fig. 6(a) that, compared with the first experiment, the advantage of CKCD over other benchmarks in terms of fairness increases. Meanwhile, Fig. 6(b) indicates that CKCD provides the lowest average delay performance, while guaranteeing comparable throughput and link utilization with other benchmarks.

3) Five BBR, Five CUBIC and Five Reno: In the third set of experiments, we introduce five BBR flows, five CUBIC flows and five Reno flows. The histogram of fairness, throughput and delay is shown in Fig. 7. It can be seen that with the increase of number of flows, CKCD still provides the best fairness performance and maintains high throughput. As the number of flows grows, the average latency increases under all the queue management algorithms, but the average latency of CKCD is still the lowest. Hence, as the number and type of heterogeneous TCP flows in the bottleneck link increase, the effectiveness and advantage of the proposed CKCD over other benchmarks remain unchanged.

4) Intra-flow Fairness: we show that CKCD achieves inter-flow fairness without deteriorating the intra-flow fairness. As shown in Fig. 8(a), when five CUBIC flows sequentially join and share a bottleneck node running the proposed CKCD, their throughput eventually merges. Similar phenomenon is observed for five BBR flows in Fig. 8(b).

V. Conclusions

In this paper, we present the design, implementation, and performance evaluation of a novel active queue management mechanism called CKCD, which addresses the problem of unfair bandwidth allocation when heterogeneous flows share the bottleneck link. It achieves fairness by actively punishing unresponsive, aggressive flows, and guarantees low latency by actively dropping packets with large queuing delay. CKCD is with low complexity and can be easily implemented. Simulation results confirmed that CKCD can effectively improve inter-flow fairness and reduce average latency, while ensuring 96% link utilization. In the future, we will investigate the performance of CKCD for multi-path TCP flows.

Acknowdgement

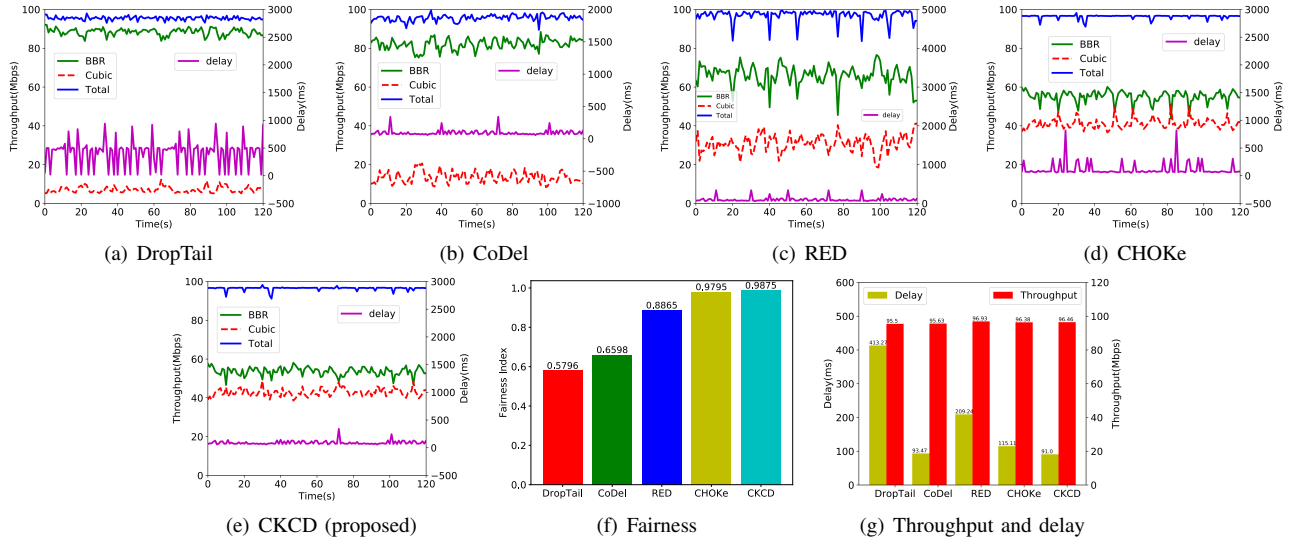
This work is supported by the Natural Science Foundation of China under Grant No. 61871437 and in part by the Natural Science Foundation of Hubei Province of China under Grant 2019CFA022.

References

[1] S. Ha, I. Rhee, and L. Xu, "CUBIC: A new TCP-friendly high-speed TCP variant," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 64-74, 2008.

[1]https://iperf.fr/

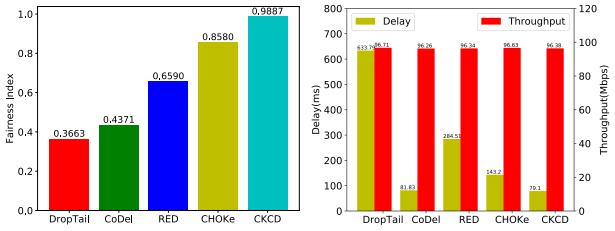
[2]https://wiki.debian.org/TrafficControl



(a) DropTail (b) CoDel (c) RED (d) CHOKe

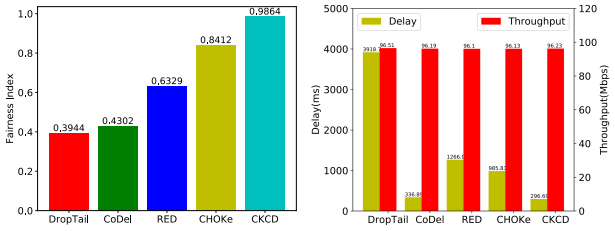
(e) CKCD (proposed) (f) Fairness (g) Throughput and delay

Fig. 5: Performance comparison of different schemes when supporting one BBR flow and one CUBIC flow.



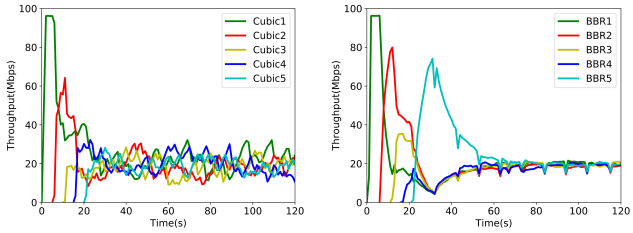
(a) Fairness (b) Throughput and delay

Fig. 6: Performance of one BBR, one CUBIC and one Reno.



(a) Fairness (b) Throughput and delay

Fig. 7: Performance of five BBR, five CUBIC and five Reno.



(a) Intra-flow fairness of five CUBIC (b) Intra-flow fairness of five BBR

Fig. 8: Intra-flow fairness with CKCD.

[2] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-based congestion control," *ACM Queue*, vol. 14, no. 5, pp. 20-53, 2016.

[3] G. Carlucci, L. De Cicco, and S. Mascolo, "Controlling queuing delays for real-time communication: the interplay of E2E and AQM algorithms," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 46, no. 3, pp. 1-7, 2018.

[4] M. Hock, R. Bless, and M. Zitterbart, "Experimental evaluation of BBR congestion control," in *Proc. of IEEE ICNP*, Toronto, Canada, 2017, pp. 1-10.

[5] Y. Zhang, L. Cui, and F. P. Tso, "Modest BBR: Enabling Better Fairness for BBR Congestion Control," in *Proc. of IEEE ISCC*, Natal, Brazil, 2018, pp. 646-651.

[6] K. Winstein and H. Balakrishnan, "TCP ex machina: Computer-generated congestion control," in *Proc. of ACM SIGCOMM*, Hong Kong, China, 2013, vol. 43, no. 4, pp. 123-134.

[7] M. Dong, D. Zarchy, E. Arslan, Y. Gilad, B. Godfrey, and A. Valadarsky, "PCC Vivace: Online-learning congestion control," in *Proc. of USENIX NSDI*, Renton, USA, 2018, pp. 343-356.

[8] X. Nie, Y. Zhao *et al.*, "Dynamic TCP initial windows and congestion control schemes through reinforcement learning," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 6, pp. 1231-1247, 2019.

[9] Z. Xu, J. Tang *et al.*, "Experience-driven congestion control: When multi-path TCP meets deep reinforcement learning," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 6, pp. 1325-1336, 2019.

[10] P. Yang, N. Zhang, S. Zhang, L. Yu, J. Zhang, and X. Shen, "Content popularity prediction towards location-aware mobile edge caching," *IEEE Trans. Multimedia*, vol. 20, no. 4, pp. 915-929, Apr. 2019.

[11] K. Sasaki, M. Hanai *et al.*, "TCP fairness among modern TCP congestion Control algorithms including TCP BBR," in *Proc. of IEEE CloudNet*, Tokyo, Japan, 2018, pp. 1-4.

[12] K. Nichols and V. Jacobson, "Controlling queue delay," *ACM Queue*, vol. 10, no. 5, pp. 20-34, 2012.

[13] S. Ma, J. Jiang, W. Wang, and B. Li, "Fairness of congestion-based congestion control: Experimental evaluation and analysis," *arXiv preprint*, arXiv: 1706.09115, 2017.

[14] M. Yang, P. Yang, C. Wen, Q. Liu, J. Luo, and L. Yu, "Adaptive-BBR: Fine-grained congestion control with improved fairness and low latency," in *Proc. of IEEE WCNC*, Marrakech, Morocco, 2019.

[15] T. Dai, X. Zhang, and Z. Guo, "Analysis and experimental investigation of BBR," in *Proc. of IEEE INFOCOM*, Honolulu, USA, 2018, pp. 1-2.

[16] A. Tang, J. Wang, and S. H. Low, "Understanding CHOKe," in *Proc. of IEEE INFOCOM*, San Francisco, CA, USA, 2003, pp. 114-124.

[17] S. H. Low, F. Paganini, J. Wang, S. Adlakha, and J. C. Doyle, "Dynamics of TCP/RED and a scalable control," in *Proc. of IEEE INFCOM*, New York, USA, 2002, pp. 239-248.

[18] L. Xue, S. Kumar *et al.*, "Towards fair and low latency next generation high speed networks: AFCD queuing," *Journal of Network and Computer Applications*, vol. 70, pp. 183-193, 2016.

[19] Q. Peng, A. Wakid *et al.*, "Multipath TCP: Analysis, design, and implementation," *IEEE/ACM Trans. Netw.*, vol. 24, no. 1, pp. 596-609, 2016.