On the Coexistence of Transport Protocols in Data Centers

Syed Mohammad Irteza, Adnan Ahmed, Sana Farrukh, Babar Naveed Memon, Ihsan Ayyub Qazi
Computer Science Department, LUMS
Email: {syed.irteza,13100204,15100122,15100142,ihsan.qazi}@lums.edu.pk

***Abstract*—The emergence of cloud data centers has led to the design of customized transport protocols such as Data Center TCP (DCTCP). These protocols improve the performance of cloud applications by explicitly accounting for the unique network and traffic characteristics in data centers. However, such protocols have only been evaluated under greenfield deployment scenarios, where the entire data center is assumed to use the same protocol, which may not always be desirable or feasible. This leads to scenarios where these protocols coexist with TCP and thus share the same network resources. This paper considers such scenarios and presents a comprehensive study of the coexistence of DCTCP and TCP. In particular, we evaluate their bandwidth sharing properties under different active queue management schemes (AQM) including RED, DCTCP AQM, and CHOKe. Our results show that under the DCTCP AQM, DCTCP can starve TCP flows. This problem is mitigated through the use of RED, however, significant unfairness remains. Interestingly, we find that CHOKe exacerbates this unfairness. We show that a modified version of CHOKe considerably improves fairness by more accurately penalizing dominating flows.**

I. Introduction

The emergence of large-scale data centers has transformed the computing landscape with massive online services such as web search, social networking, and advertising systems, and the rise of cloud computing service providers like Amazon, Google, and Microsoft [1]. These data centers have unique network and traffic characteristics and host diverse applications. The diverse requirements of such applications has led to the design of customized transport protocols for data centers [2], [3], [4], [5], [6]. For example, Data Center TCP (DCTCP) was designed to meet the requirements of soft real time applications like web search, advertising, and retail [2].

Generally, these protocols have been evaluated under greenfield deployment scenarios where the entire data center is assumed to use the *same* protocol. However, such scenarios may not always be desirable or feasible. First, typical data centers host multiple applications at the same time to enable flexible use and high utilization of resources [1], [3]. While some applications (e.g., soft real time applications such as web search and social networking) may benefit from the use of deadline-aware ([3]) or latency-minimizing ([5], [6]) transport protocols, other applications (e.g., cloud services in multi-tenant environments having service level agreements, or SLAs, associated with them) may require the use of fair-sharing transport protocols [2]. Second, the use of new transport protocols requires changes in software and/or hardware and may mandate rewriting of applications ([4], [6]), which may not always be feasible (e.g., due to the use of proprietary systems) or desirable (e.g., due to the need for a significant data center downtime for greenfield deployment, thereby degrading data center availability).

This leads to cases where customized data center transport protocols will coexist and share network resources with existing transport protocols like TCP. This sharing, however, can lead to unfairly low throughput or starvation for one protocol or the other. This paper presents a comprehensive study of such scenarios and studies the coexistence properties of DCTCP and TCP under several active queue management (AQM) schemes.

Our results show that the DCTCP AQM can lead to the starvation of TCP flows. This happens due to DCTCP AQM's aggressive marking based on instantaneous queue length coupled with DCTCP's use of smaller back-off factors compared to TCP. We show that Random Early Detection (RED) [7] improves fairness between protocols but still leads to significant differences in throughput. We then consider CHOKe [8] and interestingly, show that it degrades fairness due to its marking policy which negatively impacts non-dominating TCP flows. We suggest a simple change to CHOKe which improves the detection accuracy of dominating flows and considerably improves fairness over RED.

Altogether, this paper makes the following contributions.

- We present a rigorous analysis of the coexistence behavior of DCTCP and TCP when they compete at a bottleneck under different AQM schemes. Using simulations and a model, we study the impact of increase and decrease parameters as well as of AQM on the performance of protocols.
- We show that while RED improves fairness, CHOKe actually degrades it. We propose a modification to CHOKe, which improves performance compared to RED by more accurately penalizing dominating flows.
- We evaluate DCTCP and TCP under typical data center specific traffic patterns in terms of average flow completion times (AFCT) and throughput.

The rest of the paper is organized as follows. We describe DCTCP and discuss AQM schemes in section II. We discuss the coexistence of protocols in section III and present evaluation in sections IV and V. The related work is discussed in section VI and we offer concluding remarks in section VII.

978-1-4799-2003-7/14/$31.00 ©2014 IEEE

II. Background: DCTCP and AQM Schemes

We now describe DCTCP and the AQM schemes over which we evaluate the coexistence of DCTCP and TCP.

A. Data Center TCP

DCTCP [2] aims to achieve low delay and high throughput in data center environments. It reacts to the *extent* of congestion by using the fraction of marked packets. As a result, when congestion is low, it applies a small back-off factor. However, when congestion increases, it increases the back-off factor. The maximum back-off factor used by DCTCP is 0.5; the same as that of TCP. DCTCP maintains a moving average, α, of the fraction of marked packets, which it uses to determine the back-off factor, as:

$$\alpha = \alpha \times (1 - g) + F \times g \tag{1}$$

where F is the fraction of marked packets in the last RTT. DCTCP uses $\alpha/2$ as the back-off factor.

B. Active Queue Management Schemes

DCTCP AQM: The DCTCP AQM marks all packets when the queue length exceeds a certain threshold K by setting the Congestion Experienced (CE) bits using ECN. Unlike RED, the DCTCP AQM uses the *instantaneous* queue length, which leads to more aggressive marking compared to RED.

RED: The RED AQM probabilistically marks packets based on the average queue length, q_{avg}. When q_{avg} exceeds the lower threshold min_{th}, it starts marking packets. The marking probability p increases linearly till the q_{avg} exceeds max_{th} as:

$$p = p_{max} \times \left(\frac{q_{avg} - min_{th}}{max_{th} - min_{th}} \right) \tag{2}$$

where p_{max} is the maximum marking probability. When q_{avg} becomes greater than max_{th}, RED marks each arriving packet with probability one.

CHOKe: The CHOKe AQM builds on RED by incorporating preferential marking/dropping of packets based on flow classification, which helps in appropriately penalizing dominating flows. In particular, when q_{avg} exceeds min_{th}, CHOKe picks a random packet from the queue and checks if the incoming packet is of the same flow as the random packet. If true, it marks[1] both packets. Otherwise, it marks the incoming packet with a probability calculated using Equation (2). If q_{avg} exceeds max_{th}, it marks the packet.

III. Coexistence of Protocols

Many data center transport protocols adapt TCP parameters based on network conditions to achieve high performance [2], [3], [5]. For example, DCTCP adapts the back-off factor in the range $[0.5, 1]$ based on the degree of congestion in the network. D^{2}TCP [3] adapts the back-off factor based on flow deadlines and L^{2}DCT [5] adapts both the increase factor as well as the back-off factor to minimize completion times by approximating the Shortest Remaining Processing Time (SRPT) scheduling discipline. Thus, when competing with TCP, these protocols achieve different throughput performance based on the differences in parameter settings.

Several models exist which characterize the performance of protocols using different Additive Increase Multiplicative Decrease (AIMD) parameters, when they coexist [9], [10]. However, they assume the same packet dropping/marking probability for both the protocols, which may not hold in reality because it depends on the choice of the AQM scheme employed at the routers. Thus, besides the differences in parameters, the AQM scheme can significantly impact the performance seen by protocols. In fact, in some cases using an AQM can exacerbate unfairness and even to lead to starvation of one of the protocols.

Our goal in this work is to analyze the impact of several commonly used AQMs on the coexistence of protocols. In particular, we assess how closely these AQMs enable fair dropping/marking so that the differences in performance only occur due to differences in their AIMD parameters rather than the queueing dynamics at the switches.

A. Quantifying Throughput Differences

We now analyze the differences in throughput we expect based *solely* on the use of different AIMD parameters used by DCTCP and TCP.

Consider an AIMD system of N flows where each flow uses a different AI and MD parameter i.e., α_i and β_i, respectively, similar to the models in [9], [10]. All flows share a single bottleneck link and are indexed by $i \in \{1, 2, ..., n\}$. We assume that flows have homogeneous RTTs equal to T and that each flow is informed of congestion one RTT after a queue becomes full. Flows are assumed to be synchronized; a typical scenario in data center environments [1]. Given the above assumptions, the network of AIMD sources described in [9] converges to a unique stationary point $W_{ss} = \theta x_p$, where θ is a positive constant, W_{ss} is the window size, and x_p is given by:

$$x_p^T = \left[\frac{\alpha_1}{1 - \beta_1}, ..., \frac{\alpha_n}{1 - \beta_n} \right]. \tag{3}$$

Since flows have the same RTT, the ratio of throughput T_r of two flows (one DCTCP and the other TCP) using different AIMD parameters is given by:

$$T_r = \frac{\alpha_{dctcp}(1 - \beta_{tcp})}{\alpha_{tcp}(1 - \beta_{dctcp})} \tag{4}$$

where $\alpha_{dctcp} = 1$, $\beta_{dctcp} \in [0.5, 1]$, $\alpha_{tcp} = 1$, and $\beta_{tcp} = 0.5$ correspond to the AIMD parameters for DCTCP and TCP, respectively.

Figure 1 shows T_r as the back-off factor β_{dctcp} of DCTCP is varied in the range $[0.5, 1]$. Observe that as β_{dctcp} increases, T_r also increases. This happens because by backing off less, DCTCP is able to maintain a larger window size compared to TCP and thus achieves higher throughput. Note that when $\beta_{dctcp} = 0.5$, T_r is 1 and flows achieve the same throughput.

[1]We assume that end-hosts and routers are ECN capable, however, without this capability packets would be dropped rather than marked.

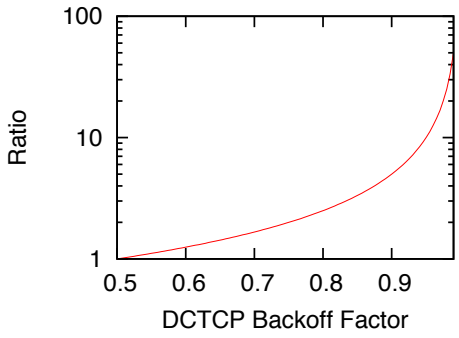


Fig. 1. Throughput ratio T_r (DCTCP/TCP) with various β_{dctcp} values.

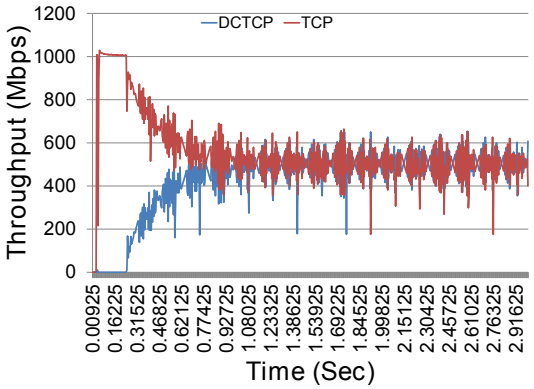


Fig. 2. Throughput of one long-lived DCTCP and TCP flow under a Drop-Tail queue. The bottleneck capacity was 1 Gbps.

IV. TCP AND DCTCP WITH DIFFERENT AQMS

In this section, we evaluate the coexistence properties of DCTCP and TCP when they share a bottleneck link. We consider several AQMs including RED, DCTCP AQM and CHOKe, and compare them with Drop-Tail queues.

Simulation Setup: We conduct our evaluation using ns-2 simulations. We use a single-rooted tree topology for our evaluation as it is commonly used in data centers [2], [11], [12]. The bottleneck link capacity is set to 1 Gbps whereas all other links have a capacity of 10 Gbps. One source generates TCP traffic whereas the other generates DCTCP flows. We use TCP NewReno with ECN capability and the packet size is set to 1500 bytes. The round-trip propagation delay (RTT) is set to 250 μs as reported in prior works [2], resulting in a bandwidth-delay product (BDP) of ∼22 packets. The buffer size is set to 250 packets [2]. To remove the impact of RTT heterogeneity, both DCTCP and TCP flows use the same RTT. The flow starting times are randomized to isolate the impact of any phase effects. Unless stated otherwise, each experiment is repeated ten times, and we report the average of these results.

Metrics: We compare the performance of DCTCP and TCP using the following two metrics:

- To capture the fairness properties, we use the throughput ratio $T_r = T_{dctcp}/T_{tcp}$ where T_{dctcp} and T_{tcp} are the average throughputs achieved by DCTCP and TCP flows, respectively. Note that $T_r = 1$ implies that both protocols achieve the same throughput.
- To track system efficiency, we also measure the aggregate throughput T_a, which is the sum of the throughputs achieved by all flows.

A. TCP and DCTCP with Drop-Tail Queues

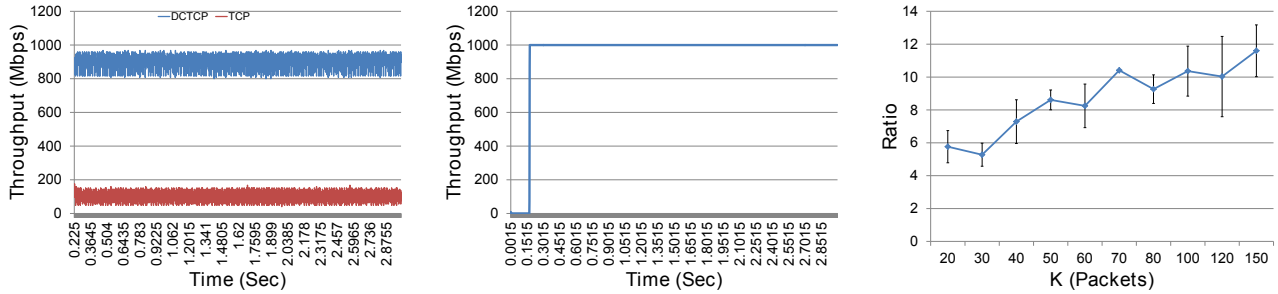
We first consider the coexistence properties of TCP and DCTCP when the bottleneck router uses a Drop-Tail queue. Under this scenario, packets (either TCP and/or DCTCP) are dropped from the tail of the queue when an arriving packet finds the queue to be full. Thus, the back-off mechanism for both protocols is driven by packet drops, which results in DCTCP backing off by the same amount as TCP i.e., 0.5. This can be seen in Figure 2 where both flows converge to the same throughput values. However, there are two challenges with using Drop-Tail queues: (a) it leads to large queueing delays (the average queue occupancy was 71%) which is undesirable as it increases the completion times of latency-sensitive traffic and (b) achieving the same throughput reduces the incentives for the deployment of new protocols like DCTCP.

B. TCP and DCTCP with DCTCP AQM

We now consider the DCTCP AQM. Figure 3(a) shows the throughput (measured over the RTT timescale) as a function of time. Observe that DCTCP virtually starves TCP flows and achieves ∼8× more throughput than TCP flows. This happens due to (a) the aggressive marking of the DCTCP AQM based on instantaneous queue length and (b) milder back-off factors used by DCTCP. DCTCP's use of smaller back-off factors than TCP causes it to maintain an average queue length close to the marking threshold K. Due to the aggressive marking of DCTCP, this allows very few packets from TCP flows to be accommodated in the buffers and leads to frequent back-offs for the TCP flow, thus degrading its throughput. Note that T_a remains at 1 Gbps (See Figure 3(b)).

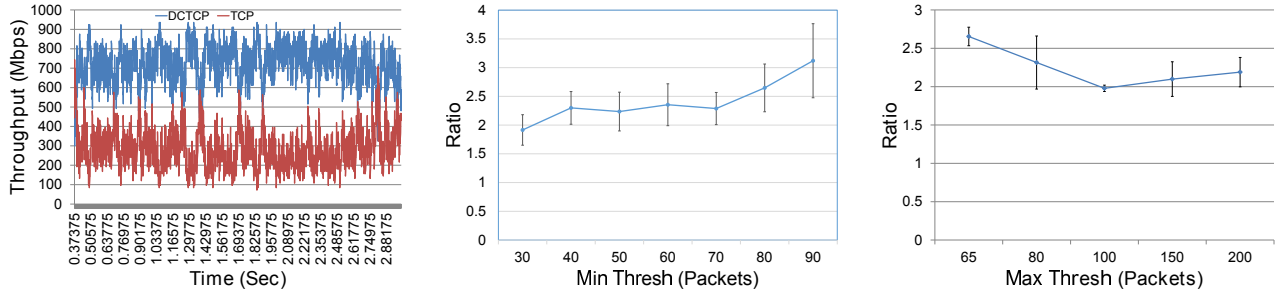
Impact of the Marking Threshold K: The marking threshold K determines the delay and throughput achieved by DCTCP flows. [2] suggests that K should be at least $BDP/7$ to avoid any loss of link throughput. Unlike TCP, which requires BDP buffers to maintain full link throughput [13], DCTCP requires less due to its use of smaller back-off factors.

Figure 3(c) shows T_r as a function of K. Observe that as K increases, T_r increases from ∼6 at K=20 to ∼12 for K=150, indicating that DCTCP is obtaining a larger share of the bottleneck link capacity. This happens because the average queue occupancy increases as we increase K and flows now compete over a larger buffer space. In this larger buffer space, DCTCP dominates TCP, thus increasing T_r. In particular, the value of K determines the average buffer space available for both the flows. When K increases, DCTCP achieves a higher congestion window size and thus maintains a larger number of packets in the queue. Since DCTCP applies a back-off factor smaller than TCP, it leaves little headroom for TCP packets in the buffers. On the hand other, TCP's aggressive back-off allows the DCTCP flow to dominate the buffer space. This translates into an increase in the throughput for the DCTCP flow and a decrease in the throughput of the TCP flow.



(a) Throughput as a function of time for K=60 (b) Aggregate throughput with K=60 (c) Throughput ratio as a function of K

Fig. 3. The figure shows the performance of long-lived DCTCP and TCP flows when the bottleneck link uses the DCTCP AQM. (a) Shows the throughput of flows over time, (b) shows the aggregate throughput of flows, and (c) shows the throughput ratio T_r as a function of K.



(a) Throughput over time: min_{th}=50, max_{th}=100 (b) T_r as a function of min_{th} (c) T_r as a function of max_{th}

Fig. 4. The figure shows the performance of long-lived DCTCP and TCP flows when the bottleneck link uses RED. (a) Shows the throughput of flows over time for $min_{th} = 50$ and $max_{th} = 100$, (b) shows T_r as a function of min_{th} while keeping max_{th} at 100, and (c) shows T_r as a function of min_{th} while keeping min_{th} at 45.

In summary, the DCTCP AQM leads to very low throughput for TCP flows when they coexist with DCTCP due to the latter's aggressive AQM and smaller back-off-factors. Moreover, T_r depends on the marking threshold K. As K increases, T_r and queueing delays also increase.

C. TCP and DCTCP with RED

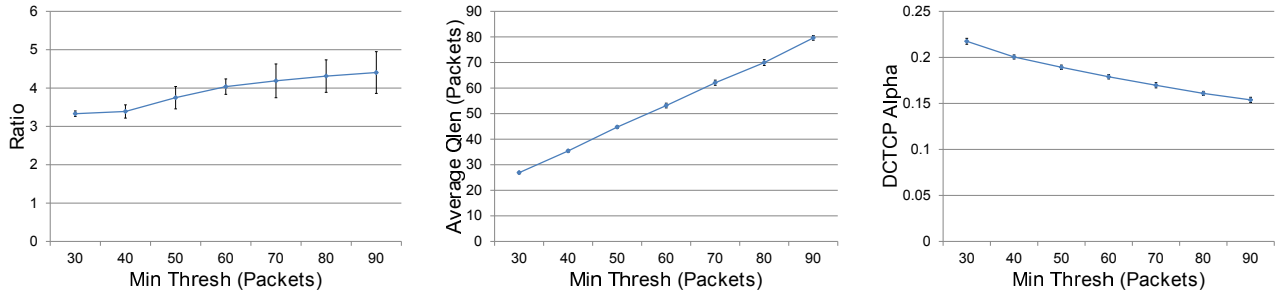
The primary reasons for DCTCP having comparatively much higher throughput than TCP are its smaller back-off factor and aggressive marking at the switches, which leads to the monopolization of buffers by DCTCP. This in turn causes TCP to back-off frequently. Thus an AQM which is fairer than the DCTCP AQM in its marking of packets can improve fairness among the protocols' flows. RED provides such an option. Hence, we now evaluate the coexistence properties under RED. We use RED with ECN marking and study the impact of minimum and maximum queue thresholds in the performance of DCTCP and TCP flows.

Figure 4(a) shows the throughput of DCTCP and TCP flows as a function of time with min_{th}=50 and max_{th}=100. Observe that TCP achieves better throughput than under the DCTCP AQM. However, both flows show large fluctuations in throughput due to the probabilistic marking of packets and the consequent queueing behavior.

Impact of min_{th}: As we increase min_{th} while keeping max_{th} fixed at 100, observe that T_r also increases. This happens because when a marked packet is received, TCP backs off more aggressively than DCTCP. Thus, on average DCTCP will get a higher share of buffers with a higher min_{th}. Note that when min_{th} is set to 30, the throughput ratio is $\sim$2. This increases to 3.5 as we increase min_{th} to 90. Thus DCTCP achieves a much higher congestion window size than TCP just like the case with DCTCP AQM. Another factor to be considered is the change in the marking probability which depends on the difference between the two thresholds. As we increase min_{th}, the difference $(max_{th} - min_{th})$ also decreases, thus making RED more more aggressive in marking, resembling the DCTCP AQM.

Impact of max_{th}: We now vary the max_{th}. Note that when max_{th} increases, the slope of the marking probability decreases (see Equation (2)). This results in increasing q_{avg} and flows now compete over a larger buffer space. However, the feedback effect is still there. The dominating flow will get its packets marked more frequently on average and this will result in a fairer distribution of throughput among flows. This is evident from Figure 4(c) where the ratio among the throughput of the flows goes down from 2.7 to around 2.25 as we vary max_{th} from 65 to 200.

In summary, RED significantly improves fairness between DCTCP and TCP flows. Increasing min_{th} degrades fairness, however, increasing max_{th} can improve fairness but this comes at the cost of increasing q_{avg}, which is undesirable for latency sensitive traffic.



(a) T_r as a function of min_{th} (b) Queue length as a function of min_{th} (c) α as a function of min_{th}

Fig. 5. The figure shows the performance a single long-lived DCTCP and TCP flow under CHOKe with ECN. (a), (b), and (c) show T_r, average queue length, and α as a function of min_{th}, respectively.

D. TCP and DCTCP with CHOKe

We now evaluate the performance of DCTCP and TCP under the CHOKe AQM. CHOKe has a fairer marking policy compared to RED. It achieves this by comparing a random packet from the queue with the arrived packet. If they both belong to the same flow, both the packets are marked. Figure 5(a) shows the throughput ratio under CHOKe with ECN. Observe that T_r is higher under CHOKe compared to RED. This happens because of the higher penalty CHOKe assigns to non-dominating flows. While marking of two packets does not affect DCTCP much due to its larger window size, it causes the throughput of the TCP flow to degrade considerably, thereby, increasing T_r. Note that both T_r and average queue length increase whereas α decreases with min_{th} as shown in Figures 5(a), 5(b), and 5(c), respectively[2].

E. TCP and DCTCP with modified CHOKe

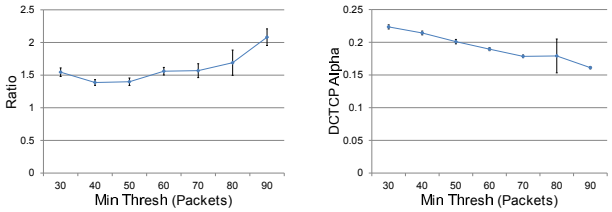
We now consider a modified version of CHOKe. With this version, when the q_{avg} exceeds min_{th}, we pick 'm' packets at random from the queue and see if they belong to the same flow as the incoming packet. The motivation behind this is that even with CHOKe, there are chances that the packets of non-dominating protocol's flows get frequently marked. When 'm' packets are checked for this criteria, the probability of the non-dominating flow getting marked is reduced.

Figure 6(a) shows T_r with modified CHOKe. Observe that it considerably improves fairness. When min_{th} is 30, T_r is $\sim$1.5 and increases to $\sim$2 when min_{th} increases to 90 due to increase in the DCTCP AQM like behavior. This happens because at small values of min_{th}, the likelihood of the DCTCP flow's packets getting more heavily penalized increases relative to TCP, as shown in Figure 6(b), which improves fairness relative to other AQM schemes.

Also, since DCTCP and TCP reduce windows only once in a RTT, the probability that TCP will have its packets marked in two different congestion window instances is even lower. So TCP is mostly affected by RED marking. On the other hand, DCTCP observes a greater back-off frequency and in addition, backs off by a greater amount than before.

[2]Interestingly, we found that the average value of DCTCP's α parameter was larger under CHOKe than under RED. However, penalization of non-dominating TCP flows had a larger effect on the resulting T_r.

According to Equation 4, when the back-off factor is 0.215 (corresponding to $min_{th} = 40$), T_r should be equal to 4.6. However, according to the evaluation results shown in Figure 6(a), T_r is lower than predicted by the model. The key reason for this behavior is that the model assumes that protocols' flows receive synchronized feedback (i.e., have the same back-off frequency). However, a flow which backs off less but uses the same increase factor, observes a higher back-off frequency. For example, under these settings and over a single simulation run, we found that DCTCP backed off $\sim$1600 times with an average back-off factor of $\sim$0.215. On the other hand, TCP backed off only $\sim$400 times even though it applies a more aggressive back-off factor of 0.5. This leads to a more fair bandwidth allocation among flows.



(a) T_r as a function of min_{th} (b) α as a function of min_{th}

Fig. 6. The figure shows the performance of a single long-lived DCTCP and TCP flow under CHOKe with $m = 7$ and ECN enabled. (a) and (b) show T_r and DCTCP α as a function of min_{th}, respectively.

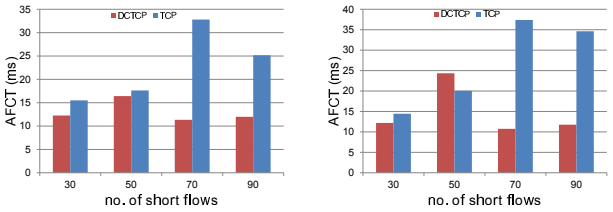
In summary, modified CHOKe improves upon RED. This is due to fairer marking of packets which penalize dominating flows more and in addition, reduces the likelihood of penalizing flows with lower throughput.

V. Data Center Specific Impairments

We now evaluate the performance of DCTCP and TCP in data center specific scenarios with RED and CHOKe. We first consider the TCP incast scenario. We then consider the benchmark settings in which latency-sensitive short flows compete with long-lived flows [2].

A. TCP Incast

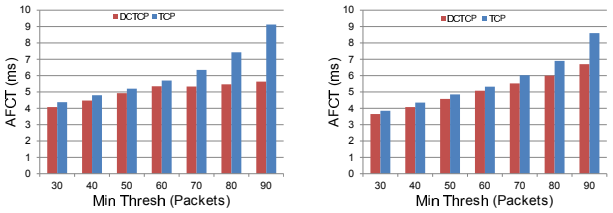
For this scenario, our setup comprises of several machines connected to a switch with 1 Gbps links. One machine acts as a client, whereas others act as servers. The client requests



(a) RED (b) Modified CHOKe

Fig. 7. The figure shows the AFCT of DCTCP and TCP flows under the incast scenario with RED and modified CHOKe. min_{th} was set to 70.

1 MB/n amount of data from each server, where n is the total number of servers the client requests, and the servers respond with the requested data. This results in synchronized responses and leads to the well-known incast impairment [1]. We always maintain one long-lived DCTCP flow and one long-lived TCP flow in the background, which is a common case in data center networks [2]. Figure 7 shows the average flow completion time (AFCT) as a function of the number of senders with RED and CHOKe under the incast scenario (Note that min_{th} was set to 70). Observe that DCTCP flows generally have shorter AFCTs compared to TCP flows under RED and CHOKe. The difference in AFCT is greater when the number of senders is large. For example, when there are 90 senders, the AFCT of TCP flows is more than 2× larger than that of DCTCP.



(a) RED (b) Modified CHOKe

Fig. 8. The figure shows the AFCT of DCTCP and TCP flows under the non-incast scenario with RED and modified CHOKe. The offered load of short flows is set to 20% of bottleneck capacity.

B. Non-Incast Scenario

For this scenario, we generate two long-lived background flows (one TCP and one DCTCP flow). In addition, short flows arrive into the network with an offered load of 20% of the bottleneck capacity; a realistic load for data center networks [5]. This load is equally distributed between TCP and DCTCP flows (i.e., 10% each). The inter-arrival times of short flows are exponentially distributed. The short flow sizes are uniformly distributed in the interval [10KB, 50KB] with an average size of 30 KB. Figures 8(a) and 8(b) show that the AFCT decreases for both RED and CHOKe when min_{th} decreases. This happens because the average queue length decreases when min_{th} decreases, which in turn improves flow completion times. Observe that modified CHOKe leads to fairer AFCTs compared to RED at higher min_{th} values.

VI. Related Work

Several protocols and mechanisms have been proposed in the past to allow coexistence of heterogeneous transport protocols. [14], [15] propose to map each protocol to a separate queue and use weighted processor sharing to schedule packets from the queues. However, that raises complex management issues and unnecessarily increases cost. We, however, assume a single queue. [16] uses a single queue but proposes to use different AQMs for different protocols. [17] proposes to use protocols in isolated islands that only use one of the protocols.

VII. Conclusion

In this paper, we studied the coexistence properties of DCTCP and TCP. We found that the DCTCP AQM can starve TCP flows. Our results showed that under RED, DCTCP significantly outperforms TCP (by at least 2×) and adapting the parameters does not improve fairness. Interestingly, we find that CHOKe degrades fairness. We proposed a modified version of CHOKe, which considerably improves fairness by accurately detecting dominating flows. In the future, we plan to investigate protocol inter-operability on a real testbed using heterogeneous applications.

References

[1] D. Abts and B. Felderman, "A guided tour of data-center networking," *Communications of the ACM*, vol. 55, no. 6, pp. 44–51, Jun. 2012.
[2] M. Alizadeh, A. Greenberg, D. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," in *ACM SIGCOMM'10*.
[3] B. Vamanan, J. Hasan, and T. N. Vijaykumar, "Deadline-aware datacenter tcp (d2tcp)," in *ACM SIGCOMM'12*.
[4] C.-Y. Hong, M. Caesar, and P. B. Godfrey, "Finishing flows quickly with preemptive scheduling," in *ACM SIGCOMM'12*.
[5] 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 *IEEE INFOCOM'13*.
[6] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pfabric: Minimal near-optimal datacenter transport," in *ACM SIGCOMM'13*.
[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] R. Pan, B. Prabhakar, and K. Psounis, "Choke, a stateless active queue management scheme for approximating fair bandwidth allocation," in *IEEE INFOCOM'00*.
[9] R. Shorten, F. Wirth, and D. Leith, "A positive systems model of TCP-like congestion control: Asymptotic results," *IEEE/ACM Transactions on Networking*, vol. 14, pp. 616–629, 2006.
[10] M. Corless and R. Shorten, "Deterministic and stochastic convergence properties of aimd algorithms with nonlinear back-off functions," in *Automatica 2011*.
[11] 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 *ACM SIGCOMM'09*.
[12] H. Wu, Z. Feng, C. Guo, and Y. Zhang, "Ictcp: Incast congestion control for tcp in data center networks," in *ACM CoNEXT'10*.
[13] G. Appenzeller, I. Keslassy, and N. McKeown, "Sizing router buffers," in *ACM SIGCOMM'04*.
[14] C. H. Tai, J. Zhu, and N. Dukkipati, "Making large scale deployment of RCP practical for real networks," in *IEEE INFOCOM Mini-Conference*, 2008.
[15] N. Dukkipati, M. Kobayashi, R. Zhang-Shen, and N. McKeown, "Processor sharing flows in the internet," in *IWQoS*, 2005.
[16] I. A. Qazi, L. L. H. Andrew, and T. Znati, "Incremental deployment of new ECN-compatible congestion control," in *PFLDNeT*, Tokyo, Japan, May 2009.
[17] D. Katabi, M. Handley, and C. Rohrs, "Internet congestion control for high bandwidth-delay product networks," in *ACM SIGCOMM'01*.