Data Center TCP (DCTCP)

Ajinkya Kadam
adk427@nyu.edu

***Abstract*—Goal of my project is to validate the following micro-benchmarks as given in [1]**

- **Queue length CDF for DCTCP and TCP : In this result i am trying to verify whether Data Center TCP maintains low buffer occupancy for long throughput sensitive flows while giving the same throughput as given by TCP. Our results verify that DCTCP in presence of long throughput sensitive flows, achieves low queue size concurrently giving high throughput.**
- **Queue Build Up : Mixing of long throughput sensitive flows and short delay sensitive flows is common in data center networks. Long flows build up the queue which delays the completion time of short flows. We find that DCTCP unfolds this problem by maintaining low queue size and allowing short flows to complete faster. The result obtained from our experiments is consistent with the result as given in the paper.**
- **Buffer Pressure : Table 2, 95^{th} percentile of query completion time. Our realization of experiment is a little different than what the authors have done. The trend is quite similar however our results deviate due to the limited resource availability.**

It is worth noting that the major goal of the paper was to demonstrate how DCTCP provides a solution to the issues faced in data center networks i.e incast, queue build up and buffer pressure. Our observations corroborate that DCTCP algorithm substanially outperforms traditional TCP and solves all three challenges.

I. Introduction

Cloud services are the prime services used today. With the increased demand of cloud computing it requires that applications be faster. Internet archietecture and protocols were not designed keeping in my mind these services. So the legacy protocols fail to meet the diverse demands that cloud services necessitate. To meet these requirements we need to design better protocols that suite these applications, providing faster access. These new applications generate traffics flows which vary from as short as a few 100KB to traffic flows of size 100MB or greater. Failing to meet the demands of this mixture results in reduced revenues for the companies that provide cloud services. DCTCP algorithm is one of the new protocols designed to mainly solve challenges inside data center networks. It is observed that 99% traffic in data centers is TCP traffic [1].

Data center networks consist of commodity switches connecting servers at gigabit speeds. The traditional commodity switches have less buffer size as compared to the routers present in wide area network (WAN). Building higher memory switches is costly. Inside data center there is a lot of distributed computation occuring simultaneously which directly affect the end user. Three important challenges are seen within the data center network namely incast, queue build up, and buffer pressure. Incast occurs when many short flows pass through a single common output port. This requires allocating more memory to that port of the switch. If we have enough memory then incast can be handled, however in the lack of enough memory there can be losses. Traditional TCP determines loss and retransmits after the retransmision timer goes off. However the default value of retranmission timer is 300ms which when compared to the data center environment seems too high. One possible solution is to reduce the timer value to say 10ms. This can help alleviate the incast issue, but not the other problems. Next, is queue build up which occurs when long flows and short flows mix at a common port on a switch. The long flows fill the queue leading to increased waiting time for short flows. This increases the completion time of the short flows. Inside data center networks there is a tight bound on the completion time of queries, if the server fails to return the queries in allocated time then the result from that server is ignored, which results in less quality of experience for user and decreased revenue for service providers. Third, buffer pressure arises due to the shared memory architecture of the switches. When many long lived flows pass through switch they consume all the buffer memory leaving no room for incast burst.

Let's see how traditional TCP works. On sensing congestion TCP reduces its window size to half of its current window size. This can lead to unnecessary drop of throughput if congestion is not severe. Authors in [1] propose Data Center TCP a congestion control algorithm precisely for data center networks. This algorithm is the first that solved all three problems and is easily deployable with little modification in TCP source code. To use DCTCP switch must be ECN capable and all commodity switches available in data centers are ECN capabled. ECN is Explicit Congestion Notification, which is used to indicate the sender that a congestion has been occured. DCTCP at senders side uses ECN notifications to extract multibit feedback, thereby measuring the amount of congestion and then modifiying its window size. Switch on experiencing congestion sets the Congestion Encountered bit in the differential services field of the IP packet. Upon receiving packets with CE bit set the receiver keeps sending ECN Notifications via setting ECN-Echo bit in ACK packets until it receives a acknowledgement from sender in the form of Congestion Window Reduced (CWR) bit set. Both ECN-Echo and CWR bit are set in the reserved field of TCP header at 9^{th} and 8^{th} bits respectively [2]. DCTCP modifies its congestion

window as follows

$$cwnd = cwnd * (1 - \alpha/2) \quad (1)$$

$$\alpha = (1 - g) * \alpha + g * F \quad (2)$$

cwnd - congestion window

α - is the estimate of the fraction of packets that are marked, updated once every window of data

F - is the fraction of packets that were marked in the last window of data

g - is the weight given to new samples against the past in the estimation of α, $0 < g < 1$

II. Experiment Design

A. Validating Queue Size CDF

Authors in [1] use a triumph switch with ECN functionality connected to hosts via 1Gbps link. We emulate a switch using a machine which has ECN capability. Topology is formed as shown in fig 1. Five servers are connected by a switch to a client node via 1Gbps link. All machines have dctcp added as congestion control algorithm and are capable of ECN mechanism. For DCTCP the marking threshold (K) which is the limit when switch starts marking packets has a lower bound as follows :

$$K > (C * RTT)/7 \quad (3)$$

For, C = 1Gbps, measured end to end RTT from servers to client is approximately 600μs. We get K = 7.1428 packets. However as the author suggests, to accomodate small micro bursts we set the marking threshold value a little higher as 20 packets. Iperf3 is used to generate long lived TCP flows from servers to client. Marking threshold mechanism is obtained by deploying Random Early Detection (RED) [3] queueing discipline available in linux traffic control [4] utility on switch port connected to client. We allocate a maximum buffer size of 700KB and marking threshold of 20 packets as done by authors. Note we set minimum threshold equal to 30000B and maximum threshold as 31500B, which equals 20 packets and 21 packets respectively of packet size 1500B. Current RED implementation doesn't allow to set both minimum threshold and maximum threshold to the same value. As the probability parameter is set to 1 switch begins marking when queue size reaches 20 packets which is what we need. After this setup we start 2 long lived flows from server-1 and server-2 towards client. On switch port we sample instantaneous queue length every 125ms. In case of TCP, switch operates in drop tail mode. We use pfifo queueing [5] discipline available in linux traffic control with "limit" parameter set to 467 packets corresponding to 700KB buffer. Note both DCTCP and TCP give the same amount of throughput which is 944Mbps in our case. This is also maximum achievable throughput of the link.

Due to the challenges in getting resources we scaled down our topology. Challenges are described in detailed in section III. Instead of starting 20 flows from 20 different hosts we start 20 flows as mentioned below.

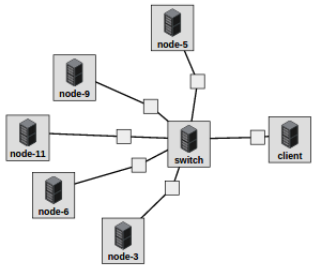

Fig. 1. Network Topology

- First, we use 2 servers, 10 long lived flows per server and measure queue length at switch port connected to client node for both DCTCP and TCP.
- Next, with 5 servers, 4 flows per server we measure queue length for both DCTCP and TCP.

Results for both cases are plotted in figure 2. We can clearly observe the difference, the more servers we use and spread the flows evenly among servers the results start to closely match figure 13 in [1] for 20 flows. However due to challenges in getting the resources we continue using topology in figure 1 as this is the max amount of resources I was able to get. In addition note that when we use two servers the variation in queue length is higher and we see many outliers. Maximum queue length as predicted from theoretical analysis in [1] is $Q_{max} = K + N$, which is 40 in our case as we have 20 long lived flows and marking threshold is 20 packets. We observe deviations from this value and outliers are present near 70 packets when we use two servers, however the variations reduce to a great extent and observed value of Q_{max} is closer to the theoretical prediction when we use 5 servers to generate long lived flows. In addition TCP's Q_{max} is dominated by the maximum buffer size available at the switch port which is 467 packets in our case.

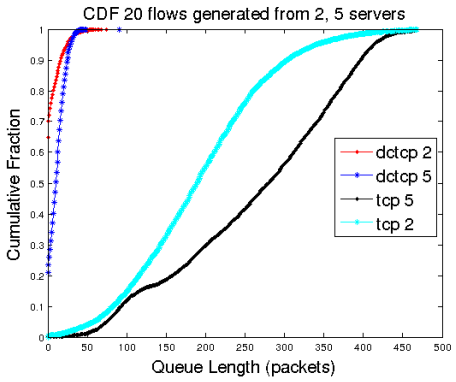

Fig. 2. CDF Queue Length 20 flows generated from 2, 5 servers

Fig 4 represents CDF of queue length reproduced from our experimental setup. DCTCP results are significantly similar and also verifies that buffer occupancy is always lower. We

observe that TCP results deviate from what is given in [1]. TCP behavior highly depend on the RTT values and as RTT varies a lot with the time so does the queue size. Also we are generating multiple flows from the same server so the flows are less evenly distributed. We should also note that kernel computation speed matters in this case, as packets need to be generated for each flows. However the general trend is similar and our result are consitent.

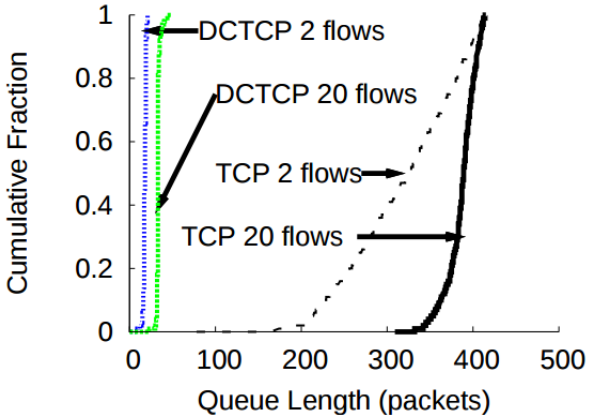

Fig. 3. Original result as given in [1]

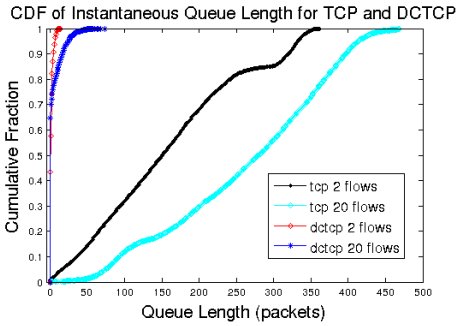

Fig. 4. Reproduced result for queue length CDF

on switch port connected to receiver with limit parameter set to 100. Client requests 20KB of data from server2 and this is repeated 1000 times. The request completion times of the flows measured are ploted in figure 6. From comparison of figure 5 and figure 6 the result closely matches. We see that with dctcp the flow completion time of short flows is less than TCP. In case of DCTCP the buffer occupancy is always smaller which reduces the latency of short flows. On the contrary, short flows see large buffer occupanices with TCP and thus the total completion time of short flows is much higher. Median delay observed for DCTCP is 2.2ms and TCP is 10.85ms.

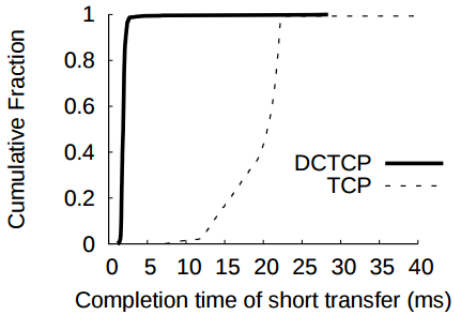

Fig. 5. Original result as given in Fig 21 in [1]

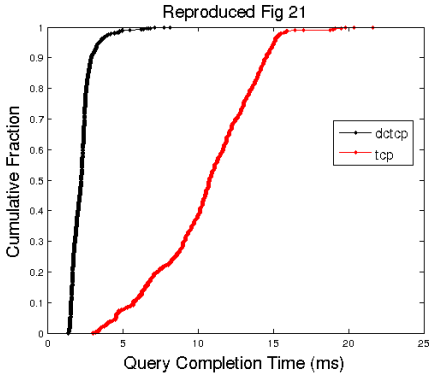

Fig. 6. Reproduced result for fig 21

B. Validating Fig 21

In this section we verify whether DCTCP maintains low latency for short flows. In data center networks it is common to have long flows and small flows incoming at a common port. Long flows consume the buffer, which leads to increased delay for short flows. Data size of short flows is around few 100KB. With Gigabit speeds data transfer of such small data flows is very small. However the data transfer speed is majorly determined by the queue occupancy. Higher queue occupancy leads to unnecessary increase in latency for these short flows. We create a similar topology as said in section 4.2.2 of [1]. We use 4 machines, 2 servers, 1 client and 1 switch. All are connected by 1Gbps links. We create short flows of 20KB using traffic generator application used in [6], [7]. 2 long lived flows each from one server to the receiver are started. In case of TCP we deploy pfifo queue discipline

C. Validating Table 2

Validating this experiment requires 48 machines all with 1Gbps link capacities in order to get results. As described in seciton III, it is tough to get machines that provide us with all three capabilities. Even scaling the network topology would not help us, which can be seen by intuition. I tried two different settings which both gave results which are not very much reliable and have large deviation.This was the best that could be achieved with the resources I had. The configuration I tried are as follows :

- With only 4 machines available I used two as servers, one as client and the other as switch. I started two long lived flows from the two servers to client. Client requested from the same servers 200KB chunks of data in total i.e 100KB data from each server, which is repeated 10000 times. The 95^{th} percentile of query completion time is

shown in table II. The results demonstrate similarity with the original result, noting that our results are scaled down since we are using lesser long lived flows and requesting less data. Both DCTCP and TCP with no background traffic take approximately same time to fetch data from the two servers. However we observe significant difference when background traffic is started. DCTCP due to low buffer occupancy provides faster query completion when compared to TCP. With two flows TCP fills the buffer untill packets drop and time out occur resulting in sawtooth behavior. Thus when TCP is used as congestion control algorithm we observe increased query completion times attributed to the fact that buffer is always full.

- Next, we start 20 flows from two servers and the same setup as above. The 95^{th} percentile of query completion time is given in table III. As we increase the degree of statistical multiplexing the query completion times increases for both DCTCP and TCP. TCP continues to perform worse while DCTCP performance degrades a little.

Please note that even though the trend that we observe in table II and III is similar as to what we expected, the validation of the given micro-benchmark would have been more realistic if we could reproduce the exact same setup as done in [1].

TABLE I
ORIGINAL RESULT

Protocol	With BG	Without BG
TCP	46.94ms	9.87ms
DCTCP	9.09ms	9.17ms

TABLE II
WITH 2 LONG LIVED FLOWS

Protocol	With BG	Without BG
TCP	54.99ms	2.33ms
DCTCP	13.2965ms	2.3020ms

TABLE III
WITH 10 LONG LIVED FLOWS

Protocol	With BG	Without BG
TCP	123.2625ms	2.33ms
DCTCP	48.9050ms	2.3020ms

III. CHALLENGES

The requirements of our experiment was majorly as follows

- Machines which support linux kernel version 3.19, since this kernel has dctcp congestion control algorithm embedded in it.
- Maximum number of machines required were 48, all machines connected via a 1Gbps link to commodity switch.

On InstaGeni I was able to get higher number of VMs but getting high capacity links was difficult. Note that data center TCP is mainly designed for data center networks where RTTs are the order of 250μs. Even getting the VM booted with the new installed kernel version was unrealiable. Using raw pc's was another good option as they are easily configurable, but obtaining large number of raw pc nodes on InstaGeni was not possible. In case of Exo Geni I was not able to reboot the nodes after installing the new kernel. So this option was ruled out. So we tried cloud lab. On cloud lab it is easy to get high capacity links and the new kernel loaded. However getting large number of nodes on cloud lab is not possible. I was able to get atmost 5 machines for 16 hours period once. As cloud lab provides with almost an equivalent environment like a data center in terms of link capacities and as it is easy to load new kernel I made a choice of using cloud lab for my experiments with a scaled down topology.

IV. CONCLUSION

Our findings validated the claims about Data Center TCP algorithm and developed a detailed understanding as how this algorithm solves the major challenges in data center networks. In our experiments we scaled down topolgies to make them fit in the frame of our GENI resources, however i would like to experiment with a data center like topology and see how much deviation occurs from our results. Additionally, the recent work in [8] is exciting and would like to contribute along the same lines.

ACKNOWLEDGMENT

I thank the following user groups which helped me in completion of this project and Prof.Fraida Fund for her advise.

- Geni user group [9]
- Cloud lab [10]
- NMA class forum
- Networking stack exchange forum [11]
- Reproducing network research [12]

REFERENCES

[1] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, pages 63–74, New York, NY, USA, 2010. ACM.

[2] RFC 3168. https://tools.ietf.org/html/rfc3168.

[3] TC-RED. http://linux.die.net/man/8/tc-red.

[4] TC. http://linux.die.net/man/8/tc.

[5] TC-PFIFO. http://linux.die.net/man/8/tc-pfifo.

[6] Amar Phanishayee, Elie Krevat, and Vijay Vasudevan. https://github.com/amarp/incast.

[7] Amar Phanishayee, Elie Krevat, Vijay Vasudevan, David G. Andersen, Gregory R. Ganger, Garth A. Gibson, and Srinivasan Seshan. Measurement and analysis of tcp throughput collapse in cluster-based storage systems. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies*, FAST'08, pages 12:1–12:14, Berkeley, CA, USA, 2008. USENIX Association.

[8] Wei Bai, Li Chen, Kai Chen, and Haitao Wu. Enabling ecn in multi-service multi-queue data centers. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 537–549, Santa Clara, CA, March 2016. USENIX Association.

[9] GENI. https://groups.google.com/forum/#!forum/geni-users.

[10] Cloud LAB. https://groups.google.com/forum/#!forum/cloudlab-users.

[11] Networking Stack Exchange. http://networkengineering.stackexchange.com/.

[12] Nikhil Handigol, Brandon Heller, Vimal Jeyakumar, and Bob Lantz. https://reproducingnetworkresearch.wordpress.com/2012/06/09/dctcp-2/.