xDCTCP : Extended Data Center TCP

Piyush Anil Nahar*, Ajeet Kumar† and Harsha Aduri‡
Department of Computer Science, University of California San Diego
Email: *pnahar@cs.ucsd.edu, †ajk054@cs.ucsd.edu, ‡vaduri@cs.ucsd.edu

***Abstract*—Datacenter environment is a special subset of network where diverse set of applications with varied requirements run together. These applications generate diverse mix of long and short flows, and require 3 things from the Datacenter networks: low latency for short flows, high throughput for long flows and high burst tolerance. DCTCP is a popular congestion control protocol used in the data center environment which tries to address these requirements. However, it treats long flows and short flows alike in the event of congestion and doesn't take into account the diverse nature of the traffic.**

In our project, we propose xDCTCP, an approach which improves upon DCTCP by differentiating between long flows and short flows in the event of congestion. The objective is to reduce the latency of short flows without compromising on the throughput of the longer flows. We try to compare our results with the original DCTCP and show how the latency can be reduced by tailoring our congestion control protocol based on flow length without adversely affecting the throughput and switch buffer occupancy.

I. Introduction

With the rise of various online services, it has become increasingly important that applications get maximum advantage of datacenter networks. Most of the datacenter traffic comes from soft real-time applications that support web search, retail, advertising, and recommendation systems. This typical Datacenter traffic is mainly composed of latency sensitive short flows and throughput sensitive long flows. Due to the real-time nature of the applications, low latency requirements directly impact the quality of the result returned and thus revenue. At the same time, continuous update of internal data structures of these applications is necessary to maintain the quality of the results returned at any time. These updates constitute of long flows and require high throughput.

DCTCP [1] proposed by Microsoft Research and Stanford University in 2010 attempts to address above mentioned requirements of datacenter traffic. DCTCP uses Explicit Congestion Notification (ECN) to mark packets after a certain threshold of queue length at a switch is reached. Thereafter, the fraction of packets marked is used to estimate the extent of congestion and finally DCTCP senders reduce their congestion window in accordance with the extent of congestion instead of reducing the window alway by half as in TCP.

Although DCTCP succeeds to an extent in maintaining low queue occupancy at the switches and other requirements mentioned above, it treats both short and long flows alike. But, the requirements of short and long flows are different, so here we propose an extension to DCTCP which finds different parameters for reducing the congestion window of short and long flows. We show that with this approach the latency of the short flows is improved by approximately 6.5% without adversely affect-

ing the throughput as compared to DCTCP.

II. Literature Survey

Datacenters have become extremely important in modern network. Most real-time applications rely on datacenters to have high performance. Due to this there have been many recent studies to understand nature of datacenter traffic and improving datacenter performance with new traffic/congestion protocols engineered specifically for datacenters. We list a few of these below:

Inside The Social Networks (Datacenter) Network [7] presents a study of network traffic observed in some of Facebooks datacenters. The paper shows that traffic patterns found in the studied Facebooks datacenter deviate substantially from the services considered in the literature. This effectively implies that datacenter problems are constantly evolving and further in-depth analysis of more datacenter networks is required to solve these problems.

Deadline-Aware Datacenter TCP (D2TCP) [6] uses both deadline information and the extent of congestion to modulate the congestion window. One essential requirement of this protocol is that applications need to pass the deadline information to the transport layer in the request to send data, so that D2TCP can modulate the congestion window in a deadline-aware manner. In case of congestion, far-deadline flows back off aggressively and near-deadline flows back off only a little or not at all. With this approach, the authors showed that D2TCP reduces the fraction of missed deadlines compared to DCTCP by 75%.

Our approach in xDCTCP is simpler and doesnt require any changes at the aggregator and comparatively few changes at the workers. We use cues from [7] to design the traffic pattern for our performance analysis.

III. The xDCTCP Algorithm

We now describe the design of xDCTCP, an improvement over DCTCP for Datacenter networks. The design of xDCTCP is based on the shortfalls discussed in §I. The goal of this approach is to reduce the latency of short flows while not hampering the throughput of long flows.The basic idea behind xDCTCP is to have different reduction factors for congestion windows of short and long flows. In this particular case of Dataventer related congestion, we notice that short flows are highly latency sensitive, while long flows aren't. Hence, we decrease the congestion window of short flows by a smaller factor as compared to that of long flows.

A. Parameters

In DCTCP, the marking of the packets is done based upon the hard threshold set at the switch i.e. K. An arriving packet is marked with the CE codepoint if the queue occupancy is greater than K upon its arrival. And then the receiver sets an ECN-Echo flag corresponding to each of the marked bits in the ACK back to the sender.

Now the sender calculates the fraction of packets marked in the last window as F. Based on this value it computes the estimate of fraction of packets marked α as follows.

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

where g is the weight given to the new samples against the old ones. Finally, the congestion window size $cwnd$ is readjusted based on the following

$$cwnd \leftarrow cwnd \times (1 - \frac{\alpha}{2}) \tag{2}$$

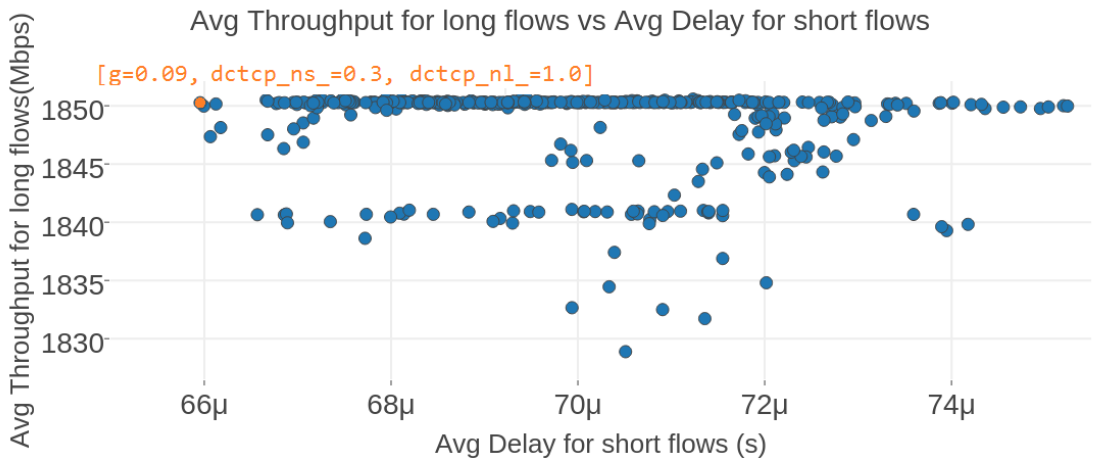

Fig. 1. The plot of average throughput vs average delay for each triplet of values $[g, dctcp\_ns\_, dctcp\_nl\_]$

B. Algorithm

In xDCTCP we treat both long and short flows differently and hence the congestion windows are assigned with different reduction factors i.e. $dctcp\_nl\_$ and $dctcp\_ns\_$ respectively. The recalculation of congestion window is based on the above factors as follows:

$$cmnd \leftarrow cwnd \times (1 - dctcp\_ns\_ \times \alpha) \quad (3)$$

$$cmnd \leftarrow cwnd \times (1 - dctcp\_nl\_ \times \alpha) \quad (4)$$

In this equation, $dctcp\_ns\_ \leq dctcp\_nl\_$. The reasoning behind this approach is that for reduction of delay for short flows, it makes sense to reduce their windows by a smaller value while reducing the windows for long flows by a greater value in case of congestion. But we need to make sure that this doesnt affect the throughput of long flows significantly.

To meet the above requirements, we find the optimal values for the triplet $[g, dctcp\_ns\_, dctcp\_nl\_]$. According to [1] the upper limit on g is calculated as

$$g < \frac{1.386}{\sqrt{2(C \times RTT + K)}} \quad (5)$$

Based on this equation we vary the value of g from the theoretical maximum and downward. For each value of g we vary the values of the window reduction factors, $dctcp\_ns\_$ and $dctcp\_nl\_$ that from 0.1 - 0.5 and 0.5 - 1.0 respectively.

For each of the triplet, we compute the average delay for short flows and average throughput for long flows in the network. The data obtained in this step is then processed to find the optimum value that satisfies our design goal. We choose the triplet that corresponds to the least average delay which doesn't compromise on the average throughput.

In fig. 1 we plot the graph of average delay against its corresponding throughput for every triplet $[g, dctcp\_ns\_, dctcp\_nl\_]$. The marked point corresponds to the optimum triplet based on our test setup, which we will see in §V. At this value of $g, dctcp\_ns\_$ and $dctcp\_nl\_$

we obtain the minimum latency for short flows while maintaining the throughput for long flows. We also try to analyze the performance and consistency of our algorithm for various kinds of traffic that occurs in Data Centers and try to prove that using different reduction parameters for long and and short flows indeed perform better than the original DCTCP.

IV. Implementation

We used NS2 [2] for simulation of our analysis. Building on the code for DCTCP NS2 simulation obtained from the authors [3], we made modifications to the congestion control code in the core TCP files `tcp.h, tcp.cc, tcp-full.h and tcp-full.cc`. We then interface our newly introduced parameters with Tcl using Tcl bindings of NS2.

To test our modifications in xDCTCP, we wrote a Tcl script consisting of code to:

- Setup the topology
- Set values for link parameters
- Set values for TCP parameters
- Establish traffic flows
- Measure flow-wise throughput and delay
- Measure congestion window size and switch queue length

For every simulation, the Tcl script produces files corresponding to instantaneous throughput and instantaneous delay at chosen sampling rate. Additionally, it generates another file containing flow-wise average throughput and delay.

We then parse the data from these files using Python to calculate the optimum triplet that corresponds to low latency for short flows and high throughput for long flows. We use this triplet for further performance analysis and comparison with DCTCP. Performance analysis graphs have been generated using Plotly [4] since it could be easily interfaced with Python.

V. Results

A. Testbed

To test the performance of our algorithm, we use NS2 to simulate Datacenter traffic. Our topology consists of 8 senders, 1 switch and 1 receiver. Each sender sends either query, short or long flow. The distribution of these among the 8 senders was varied to test the stability of the algorithm. One of the flow distribution consists of 3 query flows, 3 short flows and 2 long flows as shown in fig. 2

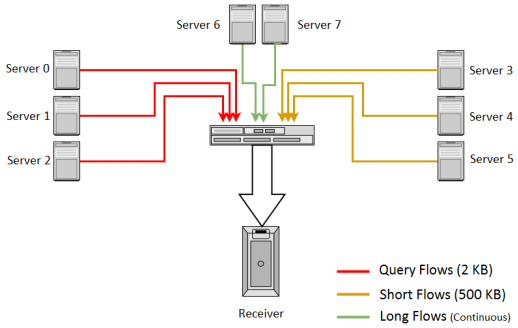

Fig. 2. Topology used for performance analysis

Parameter	Value
Link Speed	10 Gbps
Round Trip Time	100 μs
Query flow size	2 KB
Short flow size	500 KB
Query flow arrival interval	1 ms
Short flow arrival interval	10 ms
Simulation time	1 sec
ECN marking threshold	65 packets
Switch buffer capacity	250 packets
Packet size	1460 bytes

TABLE I
PARAMETERS USED FOR THE SIMULATION

Table I captures the different parameters set for the topology. The flow sizes and intervals are chosen so as to reflect Datacenter Hadoop traffic patterns as described in [7]. Based on

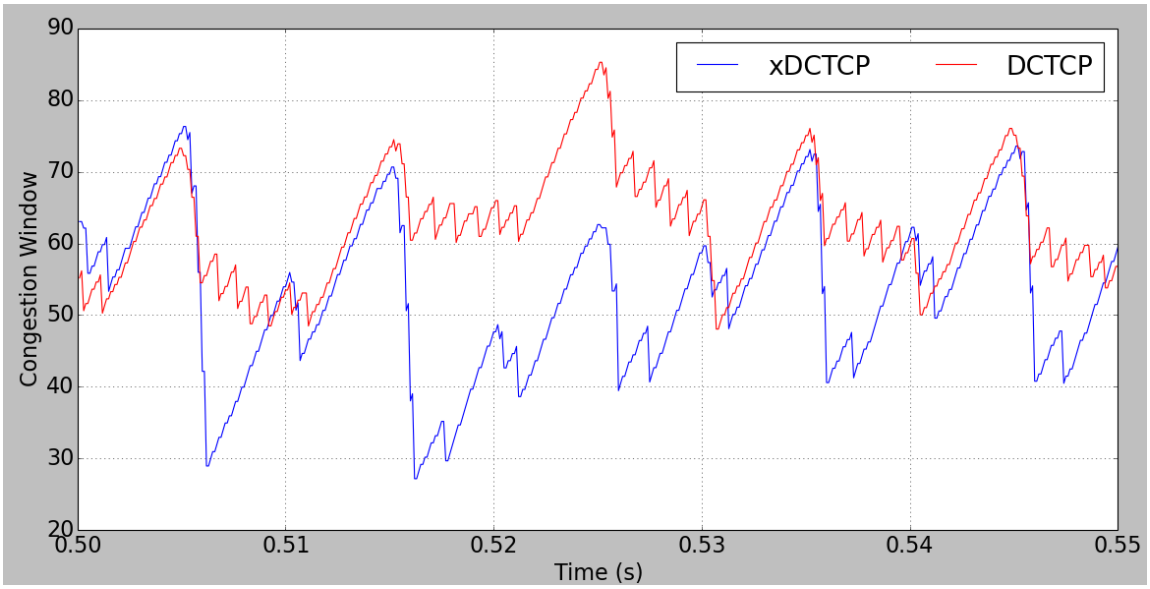

Fig. 3. Variation of congestion window for a long flow over time for DCTCP and xDCTCP

these parameters and the upper bound for g as described in eq. 5 in §III-B, we obtain the upper bound of g as 0.112. We vary the values of $g, dctcp\_ns\_, dctcp\_nl\_$ according to following relations:

$$0.05 \leq g \leq 0.11 \text{ in steps of } 0.005 \quad (6)$$

$$0.1 \leq dctcp\_ns\_ \leq 0.5 \text{ in steps of } 0.1 \quad (7)$$

$$0.5 \leq dctcp\_nl\_ \leq 1.0 \text{ in steps of } 0.1 \quad (8)$$

B. Performance Analysis

For each of the 420 possible triplet combinations, we obtain the average delay and average throughput stats for each flow as shown in fig. 1. The optimum triplet value is chosen as described in §III-B. We obtain the optimum point at $[g = 0.09, dctcp\_ns\_ = 0.3, dctcp\_nl\_ = 1.0]$. This is used for further performance analysis and comparison with DCTCP.

At the optimum point, we get 65.955μs delay for short flows and 1850.26 Mbps throughput for long flows while the corresponding values for original DCTCP are 70.65 μs and 1849.88 Mbps. This is an improvement of 6.646% in terms of latency while throughput shows a marginal 0.02% increase.

Fig. 3 shows a comparison of congestion window variation for a long flow for DCTCP and xDCTCP. It is evident that xDCTCP responds to congestion more aggressively for longer flows by cutting the congestion window by a larger value.

Fig. 4 gives a comparison of switch buffer occupancy over time for DCTCP and xDCTCP. The switch buffer occupancy is much less than the maximum queue size (250 packets) which eliminates the problem of packet loss due to buffer overflow and reduces queuing delay. The

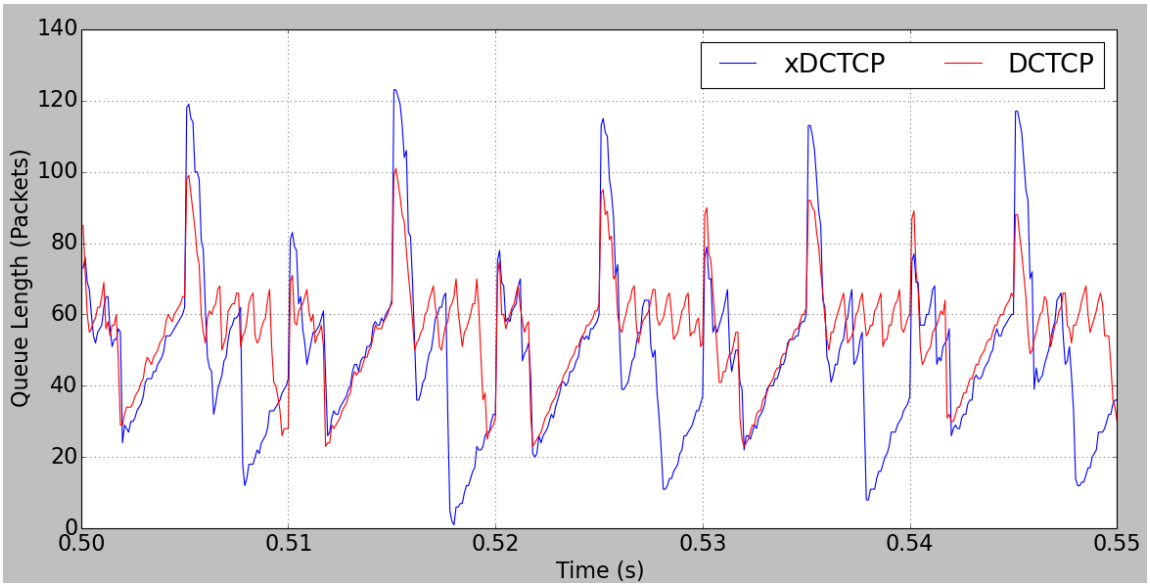

Fig. 4. Variation of queue length at switch over time for DCTCP and xDCTCP

occasional spikes in queue length are a result of the burst traffic of the short flows occupying the queue. This burst causes congestion and results in reduction of long flows' window size as can be seen from 3. These spikes are higher for xDCTCP than DCTCP because we are reducing the window size for short flows by a smaller value.

C. Stability Analysis

To test the consistency of the algorithm proposed, we used several different combinations of traffic flows i.e. variations in number of query, short and long flows in the network. We obtain consistently similar values of $[g, dctcp\_ns\_, dctcp\_nl\_]$ for various topology configurations and traffic patterns that satisfy our design goal.

VI. Challenges

During the implementation stage of DCTCP protocol for reproducing the results of the original paper, we had initially chosen to work with Mininet and OpenVSwitch (OVS). However,when we tried to replicate the original DCTCP code [3] on Mininet, the "dctcp" flags failed to initialize. This was because Mininet created a separate network namespace for each of the host network interfaces which didn't reflect the DCTCP changes we had applied to the host machine's kernel.

Similar problems were encountered when we tried network virtualization directly using `ip netns` linux command which decouples the `/proc` of the VM host. Hence, we proceeded to implement the protocol with separate Virtualbox VMs and connecting them with an OVS for checkpoint 1.

By checkpoint 1, we were successful in implementing DCTCP using 2 VMs on our own system. But this approach wasnt scalable because of the limitations of our system and changes to DCTCP would have required re-

compilation of kernel on each of the VMs individually. So, post checkpoint 1, we decided to shift to NS2 for xDCTCP implementation and simulation.

VII. Conclusion

In our performance analysis, we notice that for several possible topology configurations, the optimum values are consistently $[g = 0.09, dctcp\_ns\_ = 0.3, dctcp\_nl\_ = 1.0]$. This shows the stability of the algorithm over a wide range of traffic patterns in datacenters.

We show through results that our approach aids short flows in completing faster by paving path for them in event of congestion over long flows by cutting their window size by a larger factor. This we have achieved without compromising on the throughput. We have successfully reduced the average delay of short flows by 6.646% over DCTCP with a marginal 0.02% increase in throughput. This improvement is a great boost for highly latency sensitive flows in the data center where even minute delays can be expensive.

Thus, we conclude that differentiating long and short flows ina datacenter and reducing the window size of longer flows by greater factor results in lower latency for short flows.

References

[1] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). *ACM SIGCOMM computer communication review*, 41(4):63–74, 2011.

[2] Information Sciences Institute. The Network Simulator - ns-2, http://www.isi.edu/nsnam/ns/, December 2015.

[3] Mohammed Alizadeh. Data Center TCP, http://simula.stanford.edu/~alizade/Site/DCTCP.html, November 2015.

[4] Plotly Technologies Inc. Collaborative data science, https://plot.ly, 2015.

[5] Mohammad Alizadeh, Adel Javanmard, and Balaji Prabhakar. Analysis of dctcp: stability, convergence, and fairness. In *Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, pages 73–84. ACM, 2011.

[6] Balajee Vamanan, Jahangir Hasan, and TN Vijaykumar. Deadline-aware datacenter tcp (d2tcp). *ACM SIGCOMM Computer Communication Review*, 42(4):115–126, 2012.

[7] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C Snoeren. Inside the social network's (datacenter) network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 123–137. ACM, 2015.