Improving Flow Completion Time for Short Flows in Datacenter Networks

Sijo Joy, Amiya Nayak
School of Electrical Engineering and Computer Science, University of Ottawa, Ottawa, Canada
{sjoy028, nayak}@uottawa.ca

***Abstract*—Today's cloud datacenters host wide variety of applications which generate diverse mix of internal datacenter traffic. In a cloud datacenter environment 90% of the traffic flows, though they constitute only 10% of the data carried around, are short flows with sizes up to a maximum of 1MB. The rest 10% constitute long flows with sizes in the range of 1MB to 1GB. Throughput matters for long flows whereas short flows are latency sensitive. Datacenter Transmission Control Protocol (DCTCP) is a transport layer protocol that is widely deployed at datacenters nowadays. DCTCP aims to reduce the latency for short flows by keeping the queue occupancy at the datacenter switches under control while ensuring throughput requirements are met for long flows. But, DCTCP congestion control algorithm treats short flows and long flows equally. We demonstrate that treating them differently, by reducing the congestion window for short flows at a lower rate compared to long flows at the onset of congestion, we could improve the flow completion time for short flows by up to 25%, thereby reducing their latency up to 25%. We have implemented a modified version of DCTCP for cloud datacenters, based on the DCTCP patch available for Linux, which achieves better flow completion time for short flows while ensuring that throughput of long flows are not affected.**

I. INTRODUCTION

Today, cloud datacenters houses computing platforms that host large web scale modern Internet applications such as search, social networking and advertisements selection. These modern day datacenters houses thousands of servers interconnected by the datacenter network. Datacenter network is the central nervous system that enables distributed applications hosted in servers of a datacenter to communicate and interoperate [1].

Datacenter traffic is often characterized and measured in terms of flows. A flow corresponds to a sequence of packets from an application residing on a source to an application on a destination host and is uniquely identified by canonical five-tuple (source IP, destination IP, source port, destination port and protocol). Datacenter traffic measurement studies [2]–[4] have shown that datacenter traffic is bimodal, comprising of short flows and long flows. Short flows are either queries, responses or control messages that get exchanged between the servers. Short flows, often termed as mice flows (or foreground flows), are latency sensitive and can be of maximum size 1MB. Short flows constitute 90% of the traffic flows inside the datacenter but only 10% of the data being exchanged are in short flows [2]–[4]. Long flows, otherwise called elephant flows, correspond to applications that perform large transfers typically backup, virtual machine migration and datastructure synchronization inside a datacenter. Long flows can be of sizes ranging from 1MB to about 1 GB. Though long flows constitute only 10% in terms of traffic flows in a datacenter they carry majority (90%) of the data that is being exchanged internally [2]–[4].

Datacenters hosting online, interactive and data intensive (OLDI) applications exert diverse demands on datacenter networking. A significant requirement among them is ensuring soft-real time latency guarantees expressed in terms of service level agreements (SLAs) are met, which is crucial in ensuring user satisfaction and increased return traffic [5], [6]. Google reported 25% reduction in search query traffic as a result of an increase of 0.5 seconds in search page generation [5]. Amazon experienced 1% reduction in sales for every 100 milliseconds of latency [6]. Usage of commodity networking gear, especially commodity switches, to build datacenter networks because of the economies of scale [3], [7] add to the demands on datacenter networking. Commodity switches come with shallow buffers which are shared by multiple ports. For example it is common place to have 30-40 ports sharing 4MB of shared buffer in the Top of the Rack (ToR) switch in a typical datacenter network. Shared and shallow buffered nature of commodity switches make them more vulnerable to packet loss due to queue overflow resulting from congestion. Another aspect which adds on to the demands on datacenter networking is the partition/aggregate structure employed by OLDI applications. Partition/aggregate is an application design pattern where a request is split by the front end server or aggregator and assigned to worker nodes. Aggregator collects replies from worker nodes and forms the reply for the user. In such an environment, the time taken for a worker node to send data across to aggregator significantly impacts the latency perceived by the user. In this scenario, the link from the aggregator to the switch becomes the bottle neck link that would experience network congestion depending on the workload. In summary, shallow buffering combined with the partition/aggregate structure of the applications in cloud datacenter result in: (a) *incast impairment:* concurrent fan-in burst from worker nodes exceeding the buffer capacity causing congestion collapse at bottleneck link, (b) *queue build up:* long flows building up queue at the switch interface causing the short flows to experience longer delays even when there is no loss of packet, and (c) *buffer pressure:* long flows occupying major portion of the switch buffer without leaving enough buffer room to accommodate packet bursts which could result in packet loss.

To meet the needs of short flows in a cloud datacenter, which employs partition/aggregate structure and commodity switches, a transport protocol that ensures low queueing delay, and thereby low latency and burst tolerance is required. At the same time, the protocol should guarantee that high throughput requirements of long flows are met. This exemplifies the fact that from datacenter networking perspective it is important to determine the nature of traffic flows inside datacenter and treat them based on their type in order to ensure that the requirements of specific traffic types are met.

DCTCP [8], the first transport protocol designed for datacenter environment, employs a mechanism based on Explicit Congestion Notification (ECN) [9] to estimate the extent of congestion. DCTCP then uses this estimate to scale the congestion window thereby reacting to the extent of congestion rather than blindly reacting to the presence of congestion by halving the congestion window as in traditional TCP. This way DCTCP helps to keep queue occupancy low and at the same time ensures throughput is not affected by reacting to the extent of congestion not to the presence of it. Our work shows that we can go step further incorporating flow awareness in to DCTCP. Flow awareness can be used to adapt the congestion algorithm in DCTCP for meeting low latency requirement of short flows in a much better fashion. High performance datacenter transport mechanism is characterized as one which tries to achieve at least one of the following while not compromising on others - minimizing flow completion time, maximizing throughput or reducing deadline miss rate [8]. Our work focuses on minimizing flow completion time.

Proposed scheme, termed as *Flow Aware DCTCP* (*FA-DCTCP*), is designed ensuring that it is easier to implement and deploy with minimal changes at the end-hosts and without requiring any changes to the network fabric. *FA-DCTCP* requires only 52 lines of code change to congestion control algorithm at DCTCP sender and do not require any change to DCTCP receiver or ToR switch. Another point to note is that *FA-DCTCP*, similar to DCTCP, is tailored for the datacenter environment and deals with traffic internal to datacenter. Characteristics of datacenter environment differ significantly from the wide area networks. Especially the network will be under single administrative control and will be mostly homogeneous. *FA-DCTCP* leverages this aspect to its advantage in identifying flow types and also for ensuring easier deployability.

The key contributions of our work are: (1) proposal for a datacenter transport protocol *FA-DCTCP* that: (a) incorporates flow awareness into DCTCP, (b) treats short and long flows differently, enabling short flows to complete faster and achieve better latency without affecting the throughput of long flows, (c) incrementally deployable in datacenters, and (2) prototype based evaluation experiments to demonstrate benefits of the proposed scheme.

The remainder of the paper is organized as follows: Section II discusses background and related work. Section III lists details of *FA-DCTCP* and its implementation. Section IV covers experimental setup and evaluation results. Section V concludes the paper.

II. Background and Related Work

In this section, we highlight the need for a mechanism that leverages flow awareness while retaining simplicity and ease of deployment offered by a solution requiring minimal end-host changes and no changes to the network fabric.

State-of-the-art in low latency datacenter networking falls in to three categories: (1) *Reducing queue length* (DCTCP [8], HULL [10]), (2) *Prioritizing short flows by making transport protocol aware of flow deadlines* (D3 [11], D2TCP [12]), and (3) *Using multipath nature of datacenter networks* (DeTail [13]).

A. DCTCP

A main point to note about DCTCP, in the context of our work, is that it treats short and long flows in the same manner. i.e. DCTCP reduces congestion windows pertaining to short and long flow to the same extent, based on ECN marked packets received from the source for that particular flow.

B. High-bandwidth Ultra-Low Latency

HULL [10] caps link utilization at less than link capacity and makes use of *Phantom Queues* (virtual queues) associated to each egress switch port to ECN mark packets. HULL tries to eliminate buffering and ensures almost zero queue occupancy there by allowing latency sensitive short flows to avoid the delay that could result from having to wait at the switch's egress queues. HULL requires extensive changes in the end-hosts for supporting packet pacing and also requires changes in in-network switches for phantom queues which are not trivial to implement and deploy.

C. Deadline Driven Delivery Protocol

D3 [11] is a clean slate approach, the key idea behind the control protocol is the use flow deadlines associated with datacenter flows to schedule flows that are about to hit their deadlines over others that have their deadlines at a later point of time. D3 employs explicit rate control to reserve bandwidth in accordance to their flow deadlines and is based on the assumption that flow size and deadline information are available at flow initiation time. D3 being a clean slate approach which tries to align the datacenter network to application requirements, requires changes to applications, end-hosts, and network elements.

D. Deadline-aware Datacenter TCP

D2TCP [12] adds deadline awareness to TCP's reactive and distributed approach in order to better meet deadlines and achieve high bandwidth for long flows. D2TCP congestion avoidance algorithm makes use of gamma-correction function, with ECN feedback and deadline information as inputs, to manipulate the congestion window size. D2TCP prioritizes near-deadline flows and ensures that far-deadline flows back off aggressively in comparison to near-deadline flows.

E. Detail

DeTail [13] proposes a clean slate approach that makes use of cross-layer, in-network mechanisms to prioritize latency-sensitive flows, and evenly balance traffic across multiple paths, thereby reducing the long tail of flow completion times. DeTail requires custom switch fabric and also requires changes to the entire network stack starting from physical layer all the way up to application layer.

Schemes DCTCP and HULL try to achieve lower latencies for short flows, by reducing the time short flows have to wait at the egress queues. These schemes focused on ensuring low switch buffer occupancies could benefit from short flow prioritization as shown by the deadline aware transport protocols (D3 and D2TCP). Deadline aware protocols are designed under the assumption that applications passes on details about the deadline and size of the flows to the transport layer, which is not the case in the current datacenter environment. This is where the scheme proposed in this paper assumes significance, which taps in to the information readily available at the transport layer (IP, port number) and the ability to monitor TCP sockets to identify the nature of the flows (short or long) and leverage this to provide differential treatment to short and long flows. In essence, this work proposes to incorporate differential treatment of short and long flows by leveraging the flow awareness in to a mechanism which is focused on reducing queue length and fairness. Evaluation results show that by doing so it is possible to achieve better latency for short flows without compromising on the throughput for long flows.

III. FA-DCTCP: Design and Implementation Details

FA-DCTCP builds on DCTCP. DCTCP design treats short flows and long flows equally; whereas *FA-DCTCP* incorporate flow prioritization into DCTCP, by adding the ability to treat short flows and long flows differently. Essence of the idea behind *FA-DCTCP* is to reduce the congestion window for short flows gracefully compared to long flows, thereby improving latency for short flows. In order to guarantee that the throughputs of long flows are not affected congestion windows for long flows are reduced at the same rate as DCTCP. In short in the presence of congestion, long flows back off aggressively, whereas short flows back off gracefully in comparison, allowing short flows to complete faster. *FA-DCTCP* changes are limited to TCP stack in end-hosts and do not require any software or hardware changes in switches. *FA-DCTCP* requires only a configuration change (ECN settings) in the switches. This makes deployment of the proposed scheme fairly straight forward requiring only upgrades to the TCP stacks at the end-hosts. We implemented *FA-DCTCP* on top of the publically available DCTCP Linux source code [14]. This required only 52 lines of code change to the TCP implementation in Linux kernel. *FA-DCTCP* implementation is available at https://github.com/sijojoy/FA-DCTCP.

Main aspects that need to be taken care towards implementing *FA-DCTCP* are: (1) identifying short and long flows at the transport protocol layer in end-hosts, (2) designing a congestion control algorithm for adjusting congestion windows for short and long flows differently at the sender so that short flows complete faster compared to DCTCP. Specifics on how *FA-DCTCP* addresses these aspects are discussed below.

A. Identifying short and long flows at datacenter end-hosts

This could be done in either of the two ways listed below:

1) Distinguishing flows using the TCP port numbers corresponding to applications generating long flows: This approach is based on the luxuries a datacenter environment provides in terms of being a single administrative domain and the ability to take advantage of application awareness as pointed out in [10], [11], [15], [16]. Datacenter being a single administrative domain facilitates employment of schemes that could determine the type of a flow *a priori* by looking at the application port number. Martin Casado et al [15] points out that long flows can be determined *a priori* without actually trying to detect them from network effects since as an artifact of datacenter environment design, long flows are either related to VM cloning, migrations, backup, datastructure refresh, or file transfer, all of which can be identified from the edge (at the end-hosts) looking at the TCP headers. Another work justifying this approach is the Google's B4 architecture [17] which identifies the long and short flows *a priori* before applying specific traffic engineering rules pertaining to them.

2) Monitoring TCP socket buffer per TCP connection: This approach is based on the strategy proposed in Mahout [16] to distinguish between short and long flows. Idea here is to identify long flows by setting a threshold on the TCP socket buffer usage per TCP connection. For instance, *FA-DCTCP* implementation uses an approach where flows whose TCP socket buffer usage exceeds the threshold set at 1MB are marked and flagged as long flows. Threshold of 1MB is selected based on the datacenter traffic measurement studies [2]–[4], [8] that had pointed out that short flows range up to sizes 1MB and long flows 1MB to 1GB.

B. Congestion Control Algorithm

FA-DCTCP congestion control algorithm is built on top of DCTCP's by adding flow awareness to it. Switches in this setting are ECN configured to mark packets when the buffer occupancy reaches a stipulated marking threshold, *K*. An exponentially weighted moving average of the level of congestion, denoted as α, is maintained at the sender. α gives an estimate of the fraction of packets marked, updated once for every window of data, and is calculated using (1):

$$\alpha = (1-g) \times \alpha + g \times F \qquad (1)$$

where F indicates the fraction of packets marked in the last window and is the measure of packets that encountered congestion during that period. g the exponential averaging factor, a real number between 0 and 1, is the weight given to new samples compared to previous ones in the calculation of α. Based on α, congestion window $cwnd$ is resized using (2):

$$cwnd = cwnd \times (1- \alpha^{\beta} /2) \qquad (2)$$

where β is the congestion window scaling factor used to incorporate flow awareness in to congestion algorithm. β is used to modulate congestion window differently for short and long flows. β is set to 1 for long flows; ensuring congestion window for long flows are scaled in the same fashion as in DCTCP. In the case of short flows β is set to 2 during high levels of congestion ($\alpha > 0.6$) and is set to 3 otherwise ($\alpha \leq 0.6$). This way the congestion windows for short flows are scaled down at a lower rate in comparison to long flows. FA-DCTCP algorithm is outlined below.

Algorithm 1 FA-DCTCP congestion window calculation

```
CurrentCwnd: Current value of congestion window
Alpha: Fraction of the packets that are ECN marked
NewCwnd: Congestion window calculated by the algorithm
Input: < CurrentCwnd, Alpha >
Output: < NewCwnd >
// if flow is of type long (i.e. an elephant flow)
if FLOW_TYPE == ELEPHANT then
//calculate CWND in the same fashion as DCTCP, using
//DCTCP decrease law
   $CwndNew = CurrentCwnd \times (1 - (Alpha/2))$
else // if flow is of type short (i.e. a mice flow) calculate
     //CWND using FA-DCTCP decrease law
  if (Alpha > .6) then
  //during high levels of congestion, reduce congestion
  //window at a higher rate
     $CwndNew = CurrentCwnd \times (1 - ((Alpha)^2/2))$
  else
  //during low levels of congestion, reduce congestion
  // window at a lower rate
     $CwndNew = CurrentCwnd \times (1 - ((Alpha)^3/2))$
  end if
end if
```

Similar to DCTCP, *FA-DCTCP* algorithm comes to play only during the congestion phase; rest of the TCP operation like slow start, additive increase in congestion avoidance phase are retained as it is in original TCP implementation. An underlying assumption used in the *FA-DCTCP* design based on the literature [8], [10]–[13] on datacenter networking transport protocols is that it is safe to assume that the short flows and long flows would not use the same TCP connection, i.e., they will not be interleaved over the same TCP connection in any case.

Choosing FA-DCTCP congestion window scaling factor: Congestion window scaling factor (α^β) influences the behaviour of the *FA-DCTCP* algorithm in terms of the ability to achieve better performance, improved latency measured in terms of Flow Completion Time (FCT), while ensuring it does not lead to congestion collapse (indicated by queue over-flow). Choice of β was based on experimental evaluations where the objectives were to satisfy the following three criteria: (1) Short flows should not drive network to congestion collapse by virtue of reducing congestion windows sparsely. (2) At the onset of extreme congestion, *FA-DCTCP* should cut congestion window for short flows aggressively (but still at a rate less than long flows) and should default to DCTCP as α approaches 1. (2) Should achieve significant FCT improvement (above 10%) for short flows. Experimental evaluations were conducted for various values of β ranging from 2 to 6 as per guidelines outlined in [12]. Criteria (1) and (2) is measured in terms of queue overshoot frequency. Criterion (3) was evaluated by comparing FCT.

Fig. 1 plots the congestion window scaling factor *(α^β)* selected and used by *FA-DCTCP* based on the experimental evaluation. Straight line in the middle of the plot, which corresponds to $\beta = 1$, applies to long flows. Curve below the straight line applies to short flows. The portion of curve where $\beta = 3$ applies to short flows during mild congestion and the

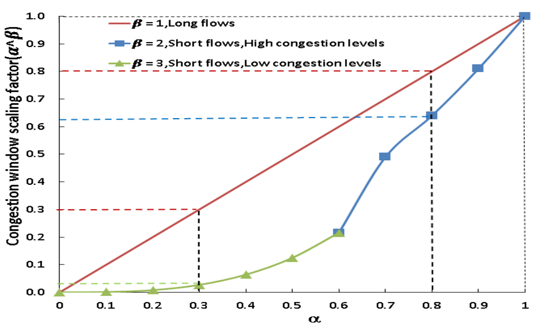

Fig. 1 Plot of α vs Congestion window scaling factor (α^β)

portion corresponding to $\beta = 2$ applies to short flows during higher levels of congestion. During mild and minor level of congestions both long flows and short flows reduces their congestion window slowly. In comparison, rate of reduction for short flows are lower than that for long flows by a power of three allowing short flows to grab more bandwidth and complete faster. For higher levels of congestion, rate of reduction of congestion window for short flows are lower than that of long flows by a power of two. As α nears 1 congestion window scaling factor for both short and long flows converge to 1.

IV. Experimental Setup, Evaluation Criteria and Results

Evaluation tests were performed using Mininet [18], a high fidelity network emulator that facilitates repeatable, realistic network experiments. Mininet is a Container Based Emulator and can be used to create a full-fledged virtual network, running real Linux kernel, on a single machine. Mininet is well suited for experiments that are network-limited, i.e. class of experiments that are constrained by network properties such as bandwidth, latency, and queueing. Handigol et al [18] points out that testing how a new version of TCP congestion control fares in a specific topology on 100 Mbps links would exemplify an excellent use case for Mininet, since results from such a test will wholly be dependent on link bandwidth and latency. As proven in [18], Mininet is well suited for recreating the DCTCP baseline results. All these reasons contributed to selecting Mininet for validating characteristics of the proposed *FA-DCTCP* mechanism. Tests were performed using Mininet running on an Ubuntu LTS box with Intel Core i5, two 2.9 GHz cores and 16 GB RAM.

A. Test topology, parameter settings and test workloads

Evaluation is done based on test topology depicted in Fig. 2, a benchmark topology which is often used in evaluation of datacenter transport protocols, derived based on the DCTCP paper [8] and Handigol et al.'s work on reproducible network research [18]. As alluded to in DCTCP paper [8] and related datacenter networking literature [11], the bottleneck in datacenter networks is the shallow buffered ToR switch. Test topology helps to emulate the bottleneck switch and link precisely for the evaluation purpose.

Ten hosts S_n (n = 1 to 10) act as senders and R represents the receiver connected to a 100Mbps switch. Switch interface connected to R acts as bottleneck link. Test set up parameter settings are listed in TABLE I. Link latencies are set to 225 µs

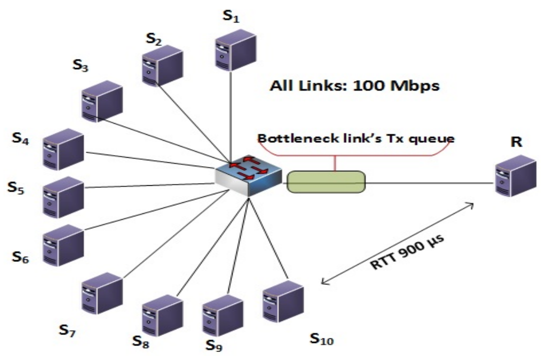

Fig. 2 Topology for Flow aware DCTCP evaluation experiments

achieving an average round trip time (RTT) of 900µs which is typical of datacenter network RTTs. Switch buffer size is set to 4MB to conform to the shallow buffered nature of ToR switches in datacenter. ECN is configured in Mininet via Linux Traffic Control's Random Early Detect queuing discipline (RED qdisc) and marking threshold of 5 packets is set. According to [8], 5 packets suffice to hit the theoretical minimum buffer size to achieve 100% throughput at 100Mbps. Another key DCTCP parameter the weighted averaging factor, g, is set to 1/16 as recommended in [8].

TABLE I
TEST SETUP PARAMETER SETTINGS

Parameters	**Value**
Link speed	100Mbps
Link Latency	225µs
Round Trip Time	900µs
Switch Buffer Size	4MB
ECN Marking Threshold	5 Packets
Exponential averaging factor	1/16

In order to validate the test setup and to ensure that we are exactly recreating DCTCP behavior correctly in the test topology, two long flows are initiated from hosts S_1 and S_2 using Iperf [19] traffic generator to receiver R. Instantaneous queue occupancy is then monitored for 120 seconds (Fig. 3) and compared against the result presented in Fig. 6(b) from [18] and Fig. 1 from [8] confirming the evaluation setup's correctness.

To evaluate and compare performance of *FA-DCTCP* to DCTCP each of the 10 senders originates 3 short flows, i.e. 30 short flows in total, following a Poisson distribution over the test duration which spans 120 seconds. Two long lived flows, originated from S_1 and S_2, span the test duration. Test traffic is generated using Iperf traffic generator.

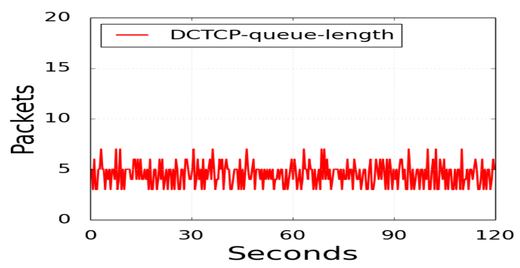

Fig. 3 DCTCP instantaneous queue occupancy from the baseline test performed to validate evaluation framework

B. Evaluation Criteria

FCT of short flows, throughput of long flows and switch queue occupancy are the metrics of interest in the evaluation. To be acceptable, FCT for short flows should demonstrate significant improvement, while ensuring that throughput of long flows are not significantly affected and queue occupancy remains low as in DCTCP. Tests were conducted with short flows of sizes 250KB, 500KB and 1MB. These tests were repeated with each algorithm, DCTCP and *FA-DCTCP,* with short flows of sizes 250KB, 500KB and 1MB, for 50 times. In order to ascertain the proposed scheme's impact on long flow throughput tests were conducted by varying long flow sizes from 10MB to 1GB.

C. Results Discussion

1) Instantaneous queue occupancy: Objective of the test is to ascertain the capability of *FA-DCTCP* to keep instantaneous queue occupancies low (near and around the marking threshold). Fig. 4 depicts the instantaneous queue occupancies at the bottleneck link which are almost the same for both DCTCP and *FA-DCTCP* around 5 packets. This serves to exemplify the capability of *FA-DCTCP* to ensure low buffer occupancy similar to DCTCP, which is vital in ensuring low latency for short flows.

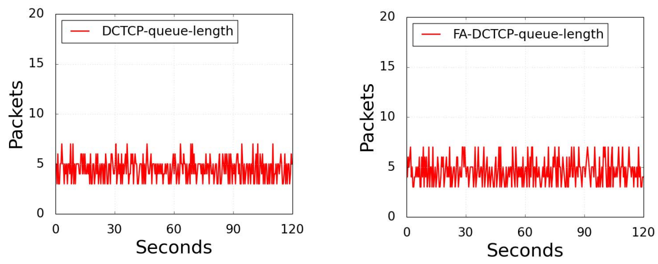

Fig. 4 DCTCP and FA-DCTCP - Instantaneous queue occupancies

2) Average flow completion time for short flows: Fig. 5 captures the average flow completion time for 30 short flows. Tests were conducted for short flow sizes 250KB, 500KB and 1000KB, and each test were repeated 50 times. With *FA-DCTCP,* average flow completion time for short flows improves by 79ms which corresponds to an improvement of up to 25.5% in average latency observed by short flows. This significant reduction in average latency serves to highlight the usefulness of *FA-DCTCP* as a good datacenter transport protocol.

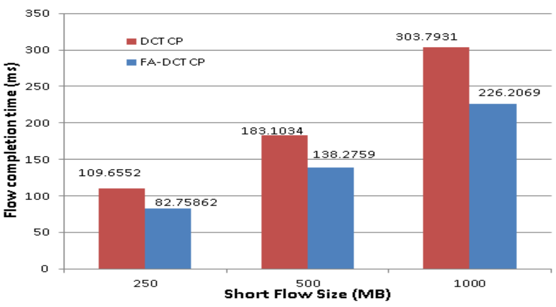

Fig. 5 Average flow completion time for short flows

3) 99th percentile of flow completion time for short flows: This metric corresponds to the tail latency in a cloud datacenter and is of much significance to applications such as

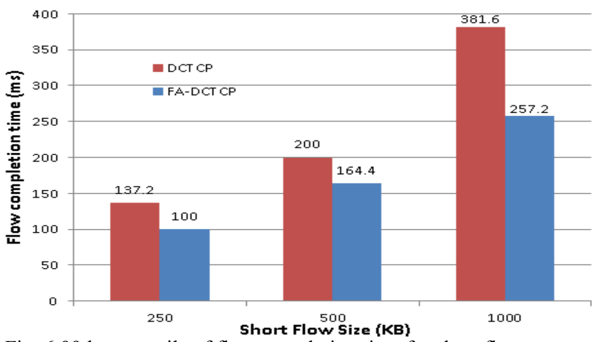

Fig. 6 99th percentile of flow completion time for short flows

search and social networking page generation where the quality of page rendered depends majorly on the tail latency. As depicted in Fig. 6, *FA-DCTCP* improves 99th percentile flow completion time by 124.4ms which corresponds to a reduction of 32.5% in the tail latency.

4) Throughput of long flows: Throughput measurements for long flows from the evaluation tests depicted in Fig. 7 shows that *FA-DCTCP* introduces only a minor negligible reduction of 0.04% in long flow throughput.

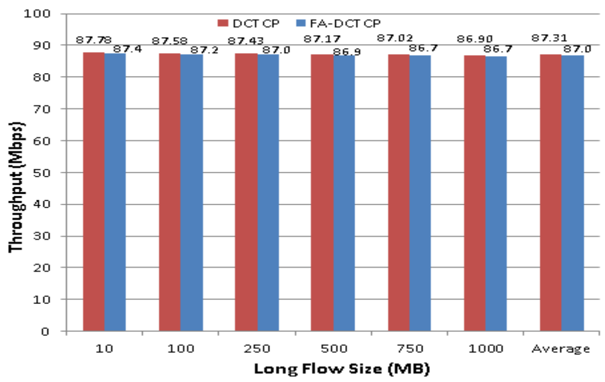

Fig. 7 Throughput for Long Flows

V. Conclusion

Our work show that, via less proportionate reduction of short flow congestion window in comparison to long flow's, by incorporating flow awareness we could ensure improved latency for short flows while not degrading throughput of long flows. From the evaluation results it is evident that the proposed scheme *FA-DCTCP* helps in significantly reducing average FCT (up to 25.5%), and 99th percentile FCT (up to 32.5%) of short flows in comparison to DCTCP. *FA-DCTCP* also ensures that the queue occupancies at the bottleneck links remain low. There is a slight negligible decrease (.04%) in throughput for long flows with this scheme; however, this tradeoff is well worth the benefit achieved in terms of faster completion of latency sensitive short flows.

References

[1] D. Abts and J. Kim, "High performance datacenter networks: Architectures, algorithms, and opportunities," in *Synthesis Lectures on Computer Architecture*, 2011, pp. 1–115.

[2] T. Benson, A. Akella, and D. A. Maltz, "Network Traffic Characteristics of Data Centers in the Wild," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, 2010, pp. 267–280.

[3] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "VL2: A Scalable and Flexible Data Center Network," in *Proceedings of the ACM SIGCOMM Conference on Data Communication*, 2009, pp. 51–62.

[4] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The Nature of Data Center Traffic: Measurements & Analysis," in *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference*, 2009, pp. 202–208.

[5] "Marissa Mayer at Web 2.0." [Online]. Available: http://glinden.blogspot.ca/2006/11/marissa-mayer-at-web-20.html.

[6] "Amazon found every 100ms of latency cost them 1% in sales." [Online]. Available: http://blog.gigaspaces.com/amazon-found-every-100ms-of-latency-cost-them-1-in-sales/.

[7] M. Al-Fares, A. Loukissas, and A. Vahdat, "A Scalable, Commodity Data Center Network Architecture," in *Proceedings of the ACM SIGCOMM Conference on Data Communication*, 2008, pp. 63–74.

[8] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data Center TCP (DCTCP)," in *Proceedings of the ACM SIGCOMM Conference*, 2010, pp. 63–74.

[9] K. Ramakrishnan, S. Floyd, and D. Black, " The Addition of Explicit Congestion Notification (ECN) to IP.", RFC3168, 2001.

[10] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda, "Less is More: Trading a Little Bandwidth for Ultra-low Latency in the Data Center," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, 2012, p. 19.

[11] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better Never Than Late: Meeting Deadlines in Datacenter Networks," in *Proceedings of the ACM SIGCOMM Conference*, 2011, pp. 50–61.

[12] B. Vamanan, J. Hasan, and T. N. Vijaykumar, "Deadline-aware Datacenter TCP (D2TCP)," in *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, 2012, pp. 115–126.

[13] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz, "DeTail: Reducing the Flow Completion Time Tail in Datacenter Networks," in *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, 2012, pp. 139–150.

[14] "Datacenter TCP." [Online]. Available: http://simula.stanford.edu/~alizade/Site/DCTCP.html.

[15] M. Casado and J. Pettit, "Of Mice and Elephants," 2013. [Online]. Available: http://networkheresy.com/2013/11/01/of-mice-and-elephants/.

[16] A. R. Curtis, W. Kim, and P. Yalagandula, "Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection," in *INFOCOM, Proceedings IEEE*, 2011, pp. 1629–1637.

[17] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a Globally-deployed Software Defined Wan," in *Proceedings of the ACM SIGCOMM Conference on SIGCOMM*, 2013, pp. 3–14.

[18] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Reproducible Network Experiments Using Container-based Emulation," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, 2012, pp. 253–264.

[19] "Iperf - The TCP/UDP Bandwidth Measurement Tool." [Online]. Available: https://iperf.fr/.