TCP improvements for Data Center Networks

Tanmoy Das and Krishna M. Sivalingam

Department of Computer Science and Engineering, Indian Institute of Technology Madras, Chennai 600036, India

Emails: {tanmoy.justu@gmail.com, krishna.sivalingam@gmail.com, skrishnam@cse.iitm.ac.in}

***Abstract*—Data centers are used to host computation-intensive distributed applications. Since all the applications are distributed in nature, their performance is significantly affected by the communication network. These applications generally create two kinds of flows: large data flows that require high throughput and small control flows that require low delay. These two contradictory requirements are not satisfied efficiently by conventional TCP protocols. The situation becomes worse due to the bursty nature of the data center traffic. DCTCP, a modified version of TCP that was specifically designed for data centers, was originally proposed by Alizadeh et al. in 2010. DCTCP uses an Active Queue Management (AQM) policy where the router marks a packet using Explicit Congestion Notification (ECN), when the number of packets in the queue is more than a marking threshold (K). DCTCP provides lower end-to-end delay in comparison with conventional TCP. However, it tends to provide lower throughput since it uses very small buffer space. In this paper, we have modified the existing DCTCP congestion control algorithm to increase the throughput without significantly increasing delay. We also present an algorithm for dynamic delayed ACK timeout calculation. The performance of TDCTCP (our modified DCTCP algorithm), DCTCP and the TCPNewReno have been studied using OMNeT++ simulator based models. It is shown that TDCTCP provides 15% more throughput and 15% more fairness when compared to DCTCP. Queue length and end-to-end delay increases slightly for TDCTCP. Additionally, TDCTCP provides more stable throughput than DCTCP and TCPNewReno. Though TDCTCP's delay is slightly higher than that of DCTCP, it provides much better throughput than DCTCP for the same marking threshold value.**

I. Introduction

A data center is a cluster of large number of computers operated and managed by a single authority. Data centers are used to host large distributed and computation-intensive applications. These days, private companies and universities are building their own data centers [1]. The main reason behind this is cheap networking equipment, lower operating costs, improved data protection, information security and management. Most of these applications follow the Partition/Aggregate pattern such as Google MapReduce [2]. This kind of application divides a processes into multiple sub-processes and assigns them to different machines.

Data centers are managed by a single authority so that new networking techniques can be easily implemented without any concern about interoperability. Distributed applications like MapReduce require high bandwidth to communicate between different components of the computing process. Data centers also need to be highly available and highly fault-tolerant. Thus, new network topologies that provide high bandwidth and high fault tolerance are required for data centers. Some of the proposed topologies for data center are DCell [3], BCube [4] and Fattree [1].

In order to design efficient networking mechanisms for data centers, the analysis of data center traffic characteristics is important [5]–[7]. The flows in a data center tend to be bursty and they are either large data flows that require high throughput (*elephant* flows) or short control flows that require low delay (*mice* flows) [7], [8]. Flows in a data center also suffer from the *incast* problem [9]. When a lot of flows are destined to a single machine, severe congestion occurs in the last node before the destination. This situation is known as incast problem.

Data center flows typically operate using the Transmission Control Protocol (TCP). TCP provides reliable and ordered bi-directional delivery of stream of bytes from one application to the other application residing on the same or two different machines. However, TCP is unable to provide high throughput and simultaneously low delay as required by data center flows. Hence, there is a need to re-design TCP specifically to handle data center traffic. Data center specific TCP modifications have been reported in [10]–[12].

In this paper, we consider the Data Center TCP (DCTCP) protocol [12] that uses Explicit Congestion Notification (ECN) [13] to indirectly control the occupied buffer size in the router in order to provide low delay and high throughput. This is done by using a buffer size threshold (K). In DCTCP [12], the performance of DCTCP in comparison with TCPNewReno is demonstrated. DCTCP provides similar throughput and fairness as TCPNewReno for high values of K, while providing lower delay. However, DCTCP does not provide good throughput and fairness when the value of K is small (as we demonstrate later). We have also shown that the variation of throughput for DCTCP is very high.

The objective of this paper is to design an improved version of DCTCP (called TDCTCP) that will provide low end-to-end delay while improving throughput and fairness. We describe three enhancements: (i) modified congestion avoidance mechanism: our modified congestion control mechanism will use the congestion indicator (explained later) of DCTCP algorithm to control the congestion window. It will help the modified algorithm to react better to the current congestion state of network and provide better throughput. (ii) resetting congestion indicator: After every delayed ACK timeout, we set the congestion indicator to the value of 0. This ensures that modified algorithm does not use a stale value of congestion indicator. As the stale value of congestion indicator remains very high, it restricts the increment of congestion window and

978-1-4673-5494-3/13/$31.00 © 2013 IEEE

hence causes successive delayed ACK timeout. Our modification provides more stable throughput. (iii) Dynamic Delayed ACK timeout calculation: Normally, TCP uses a very high delayed ACK timeout. However, this value is not suitable for all TCP flows. In actual network, the situation changes from time to time. Hence, we need to calculate the delayed ACK timeout dynamically to better adapt to the network conditions. This ensures better fairness.

The DCTCP, TDCTDP and TCPNewReno variants were implemented using the OMNeT++ simulator [14]. Single-bottleneck and multi-bottleneck network topologies were used. The proposed TDCTDP mechanism provides 15% higher throughput and fairness than that of DCTCP for the same value of K. So for smaller value of K, TDCTCP provides better throughput and fairness. This enhancement in performance reduces the buffer space requirement in the switches. Additionally, TDCTCP provides more stable throughput (measured in terms of throughput variance) than DCTCP and TCPNewReno. However, TDCTCP's queue length and delay increases slightly in comparison with DCTCP.

II. Related Work

There have been many recent attempts to modify the TCP congestion control algorithm to better suit data center traffic. Some of these modified TCP protocols are listed below:

In the approach called Incast-TCP [10], receiver side congestion control is implemented to mitigate the effect of incast problem. As stated earlier, this problem is encountered when multiple sources send packets to the same destination. This can lead to congestion and packet drops at the router before the destination. In this approach, the receive window size of these connections are dynamically adjusted to alleviate the congestion, based on the ratio of incoming throughput to expected throughput. The design rationale is that implementing receiver based systems will find easier acceptance.

In the Multipath TCP (MPTCP) approach [11], multiple paths between the sender and the receiver are utilized. The protocol divides the TCP flow into a number of sub-flows and each of them follows a different path. This is based on the planned IETF multipath-TCP extensions. The sender tries to take advantage of the possibility that there will be a small number of lightly loaded paths and transmit the subflows on these paths.

In the Data Center TCP (DCTCP) approach [12], Explicit Congestion Notification (ECN) [13] is used to indirectly control the occupied router buffer size. The objective is to provide low delay as well as high throughput. We first explain the ECN mechanism.

III. Data Center TCP

Explicit Congestion Notification (ECN): Normally routers drop packets to handle network congestion; these dropped packets serve as an indication of congestion to the transport layer protocols. Explicit Congestion Notification (ECN) [13] provides a way for the routers to mark a packet to indicate impending congestion instead of dropping them.

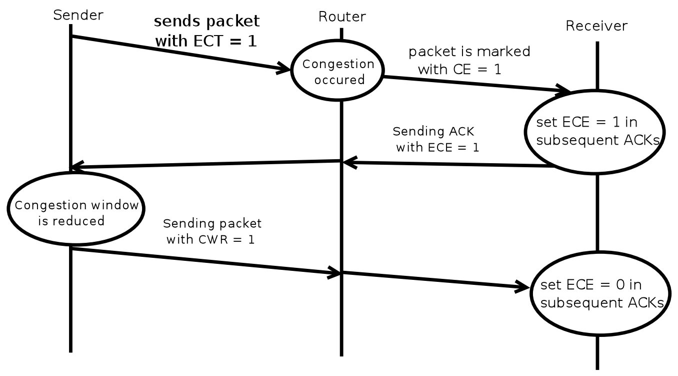

Sender
sends packet with ECT = 1
Router
Congestion occured
packet is marked with CE = 1
Receiver
set ECE = 1 in subsequent ACKs
Sending ACK with ECE = 1
Congestion window is reduced
Sending packet with CWR = 1
set ECE = 0 in subsequent ACKs

Fig. 1: The operation of Explicit Congestion Notification (ECN) mechanism.

ECN uses 2-bits of the diffserv field in the IP packet: (i) 0: non-ECN traffic (ii) 1: ECN capable traffic or ECT(1); (iii) 2: ECN capable traffic or ECT(2) (iv) 3: Congestion Encountered (or CE).

Transport layer protocols can request ECN-enabled traffic by setting the Diffserv field in the IP header to either ECT(1) or ECT(2). TCP also uses two control flags in the TCP header to support ECN. These are: *ECE:* ECN-Echo flag and *CWR:* Congestion Window Reduced. The operation of ECN is presented in Fig 1. The sender indicates to the intermediate routers that a packet belongs to a ECN-capable flow by setting ECT=1. When the packets from the ECN-enabled traffic traverse a congested router that implements some kind of AQM policy, the router marks those packets by setting the CE field to. The receiver, on seeing the CE bit set to 1, sets the ECE bit to 1 in its subsequent ACK packets. When the TCP sender receives an ACK with ECE = 1, it treats this as a packet drop and reduces its congestion window and the slow start threshold (*ssthresh*) by a factor of two. However, the sender will not reduce its congestion more than once in a single window of data.

To implement DCTCP, routers need to implement a very simple AQM policy. Routers need to detect the number of packets in the buffer. When the number of packets is greater than a threshold value (K), routers mark the arriving packets with CE = 1.

DCTCP Congestion Control: The TCP protocol reduces its congestion window by a factor of two when it receives an ACK with ECE = 1. This reaction is very pessimistic and can lower throughput. The DCTCP sender calculates the fraction of CE-marked packets seen by the receiver (α). The sequence of CE-marked packets can be easily conveyed back to the sender if the receiver sends one ACK for every packet. But this will increase the number of ACKs sent and degrade the performance. Hence, the DCTCP sender uses the *delayed ACK* mechanism to reduce the number of ACKs sent. To relay the sequence of CE marked packets back to the sender, the DCTCP receiver uses the automaton presented in Fig 2. As seen, the DCTCP receiver sends one ACK for every m (generally m is equal to 2) packets when the state does not change and sends one immediate ACK whenever the state changes. If every ACK reaches the sender, it can easily reconstruct the series of CE marked packets seen by the receiver.

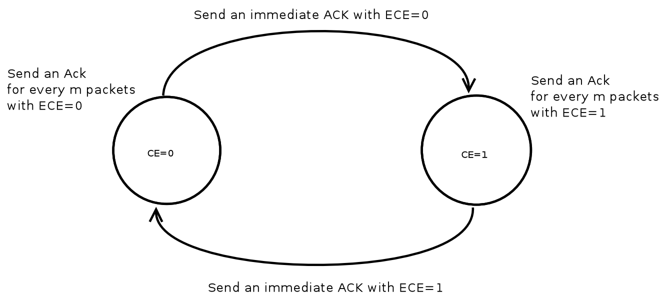

Send an immediate ACK with ECE=0
Send an Ack for every m packets with ECE=0
Send an Ack for every m packets with ECE=1
CE=0
CE=1
Send an immediate ACK with ECE=1

Fig. 2: Automaton Used by DCTCP receiver.

The DCTCP sender computes: $\alpha = (1 - g)\alpha + gF$, where F is the fraction of packets marked in one window and g is a constant. The sender reduces the congestion window using the formula, cwnd $=$ cwnd$(1 - \alpha/2)$ whenever it receives an ACK with ECE set to 1; otherwise, it behaves like a normal TCP sender.

In [12], DCTCP is shown to provide similar throughput and fairness as TCPNewReno for high values of K; while providing lower delay. We later show that DCTCP does not provide good throughput and fairness when the value of K is small. We have also demonstrated that the variation of throughput for DCTCP is very high. We seek to address this problem in this paper.

IV. Proposed TDCTCP Protocol

The proposed three modifications to the DCTCP protocol are presented below. Our modified TCP protocol is referred to as TDCTCP. The details and the rationale for each modification are also described.

Algorithm 1 Modified DCTCP Congestion Avoidance

```
1: Procedure CONGESTION AVOIDANCE
2:     If ECE = 0 Then
3:         Increase cwnd by $MSS * (1 + \frac{1}{1+\alpha/2})$ in every RTT
4:     Else ECE = 1
5:         $cwnd = cwnd * (1 - \alpha/2)$
6:     End If
7: End Procedure
```

A. Modification of Congestion Avoidance

In DCTCP, the sender calculates the fraction of marked packets in one congestion window (denoted as α). This serves as an indication of current level of network congestion. When the value of α is small, it indicates that the network is not congested and when high, that the network is highly congested. DCTCP increases its congestion window by 1 MSS (Maximum Segment Size), when ECE = 0 in congestion avoidance state. In our proposed modification, we increase the congestion window according to the level of congestion in the network. The modification is presented in Alg 1. As shown, congestion window is increased according to the value of α in congestion avoidance state. In every RTT, it is increased by $MSS * (1 + \frac{1}{1+\alpha/2})$ (we tried different versions and this formula works better than the others). When the network is less congested, the increment is high; and vice-versa. The highest value of increment is 2 MSS when $\alpha = 0$ and lowest value is 1.67 MSS when $\alpha = 1$.

As a result of this modification, the algorithm uses the available buffer space more efficiently than DCTCP. As a result, the achieved throughput is also higher with TDCTCP. We demonstrate this later in the performance analysis section (Figs. 13a and 13b).

B. Resetting α after Delayed ACK timeout

The TCP protocol uses *delayed ACK timeout* to decrease the number of ACKs it has to send to the sender. When a TCP receiver receives a packet, instead of sending an immediate ACK for that packet, the receiver waits for a fixed period of time. If a packet arrives within the time period, the receiver sends one ACK for both the packets. Otherwise it sends one ACK for the first packet at the end of the time period.

The value of α indicates the level of network congestion as explained earlier. However, when a TCP flow experiences a delayed ACK timeout, it does not send any packet to the receiver for a long time because this timeout is generally higher compared to the flow's RTT. In this situation, the value of α is not updated and it does not reflect the present level of network congestion. DCTCP's congestion window is affected by this old value. This results in a high variation in DCTCP's congestion window.

In the second modification, the value of α is reset to 0 after every delayed ACK timeout. There are two benefits resulting from this modification: 1) The old value of α gives incorrect estimation of network congestion and after delayed ACK timeout, we need to calculate it from scratch. 2) Since the old value of α remains very high, it hinders the increment of congestion window and this results in successive delayed ACK timeouts. Our modification ensures that the TDCTCP congestion window does not get affected by the old value of α. Thus it ensures better throughput while increasing throughput stability.

Fig. 3 presents the results for a single bottleneck link of capacity 1 Gbps, shared by 10 flows; $K = 16$ and buffer size is 700 KBytes. The variation in congestion window size is found to be less for all the flows, when using the proposed modification. Detailed analysis is presented in the Section V.

C. Dynamic Delayed ACK timeout calculation

DCTCP uses very small buffer space compared to TCP-NewReno. Due to this reason, DCTCP sender's congestion window remains very small. Hence, most of the times the congestion window was reduced to 1 segment causing delayed

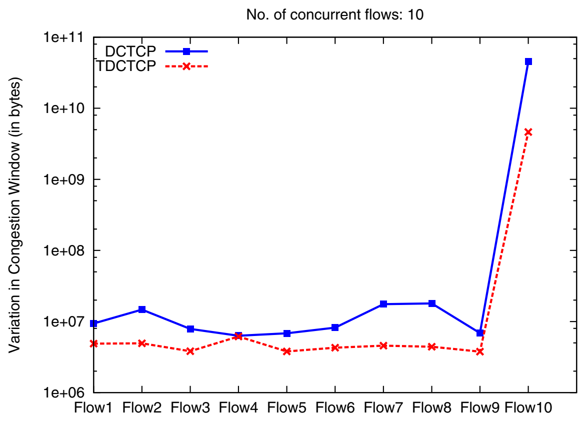


Fig. 3: Single bottleneck (10Gbps) with 10 concurrent flows. K=16 and buffer space=700 KB

ACK timeouts. As mentioned in Section IV-B, TCP's delayed ACK timeout is very high compared to RTT. This high value of delayed ACK timeout hampers the throughput and fairness of DCTCP flows. A low delayed ACK timeout value can be set in a static manner; however, this value may not be suitable for all DCTCP flows. To calculate the delayed ACK timeout dynamically, the third modification is proposed. This modification is based on the following assumptions:

- Packet arrival follows an exponential distribution.
- Packet loss probability in the network is small.

The first assumption is for modeling convenience and is only an approximation to the arrival process. The second assumption holds for DCTCP since it indirectly controls the occupied buffer size in the routers so that packet drop does not happen often.

Algorithm 2 Dynamic Delayed ACK Timeout Calculation

```
1: Procedure CALCULATEDELAYEDACKTIMEOUT
2:     arr_rate= $\frac{\text{Total bytes received}}{MSS*\text{Total time elapsed}}$
3:     max_interval = 99% of EXP(arr_rate)
4:     return min(DELAYED_ACK_TIMEOUT, max(TCP_CLOCK_GRANULARITY, constant*max_interval))
5: End Procedure
```

The proposed modification is presented in Alg 2. At first, we calculate the arrival rate (*arr_rate*) in number of packets of the DCTCP flow as: Total bytes received$/(MSS *$ Total time elapsed) in Step 2. The error in this arrival rate calculation is very small since there is virtually no packet drop as assumed earlier. We use this computed arrival rate as the parameter of exponential distribution. The maximum interval time is set to the 99^{th}-percentile of the exponential distribution with parameter *arr_rate* in Step 3. The delayed ACK timeout is set to some *constant* times the maximum interval. It is upper bounded by 200 ms as specified in [15] and lower bounded by the TCP clock granularity. We have empirically set this *constant* value to 4, based on simulation experiments.

The modification is illustrated with an example. If the total bytes received in 10 ms is 1000 bytes, then the arrival rate in number of packets is $1000/(10 * 10^{-3} * 1460) = 95.6$ pps, where MSS is taken to be 1460 bytes. The maximum inter-arrival time is set to 99% of EXP(95.6) = 0.048 seconds. The delayed ACK timeout value is set to min(0.2, max(4*0.048, 0.01)) = 0.192s (or 192 milliseconds), where TCP clock granularity is taken to be 0.01s (10 milliseconds).

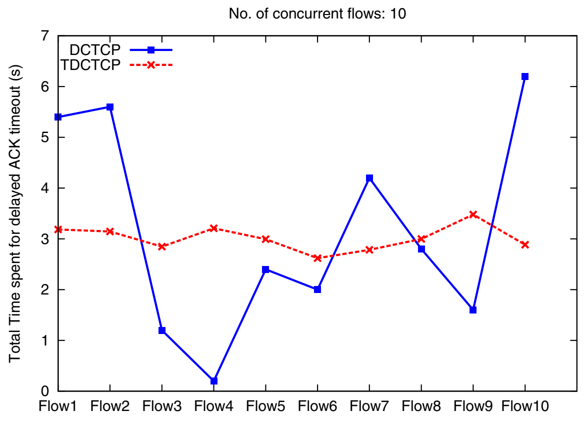


Fig. 4: Single bottleneck (10Gbps) with 10 concurrent flows; K=16 and buffer space=700 KB.

Fig 4 presents the results for a single bottleneck link of capacity 1 Gbps, shared by 10 flows; $K = 16$ and buffer size is 700 KBytes. The total time spent due to delayed ACK timeout varies from 0.2s to 6.2s for DCTCP whereas for TDCTCP this value varies from 2.8s to 3.2s. This low variation helps TDCTCP to achieve good fairness value as presented in Section V-B.

This section has presented the three modifications present in the proposed TDCTCP algorithm. The next section presents the simulation based performance study.

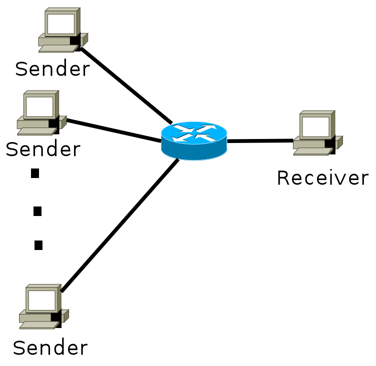
V. Performance Evaluation

This section presents the details of the performance analysis done using discrete-event simulation models. The DCTCP and TDCTCP algorithms have been implemented using OMNET++ 4.1 and its INET framework. For comparison, the TCPNewReno protocol has also been studied. The INET [16] framework is built on OMNeT++ simulation framework. It is an open-source communication network simulation package. It includes implementation of different wired and wireless network protocols like Ethernet, IPv4, IPv6, TCP, UDP, etc.

The relevant system parameters are summarized in Table I. Two different bottleneck network topologies were studied: single bottleneck and multiple bottleneck, as presented in Fig 5. Each source node's application generated packet arrivals based on a Poisson process with a rate of 4 Gbps to emulate a backlogged sender. Though the application generates this amount of data, the amount of data that is forwarded by TCP

Parameter	Description
Number of senders	10 and 15
Link capacity	1 Gbps and 10 Gbps
Buffer Size	700KB and 1.5MB
K	Threshold value for buffer
Path MTU	1500 bytes

TABLE I: Parameters



(a) Single bottleneck

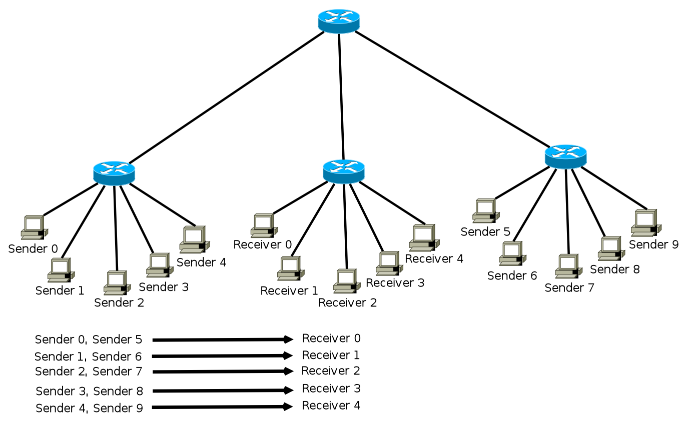

Sender 0
Sender 1
Sender 2
Sender 3
Sender 4
Receiver 0
Receiver 1
Receiver 2
Receiver 3
Receiver 4
Sender 5
Sender 6
Sender 7
Sender 8
Sender 9
Sender 0, Sender 5 → Receiver 0
Sender 1, Sender 6 → Receiver 1
Sender 2, Sender 7 → Receiver 2
Sender 3, Sender 8 → Receiver 3
Sender 4, Sender 9 → Receiver 4
(b) Multiple bottleneck

Fig. 5: Bottleneck topologies studied.

will be limited by its TCP buffer size. This, in turn, is limited by TCP'S data sending rate.

DCTCP used shared buffer switches and port in these shared switches had maximum buffer capacity of 700 KB [12]. The buffer capacity was set to either 700 KB and 1.5 MB. All the experiments were repeated for 10 times; the results are presented with the 95% confidence interval.

The performance metrics studied are:

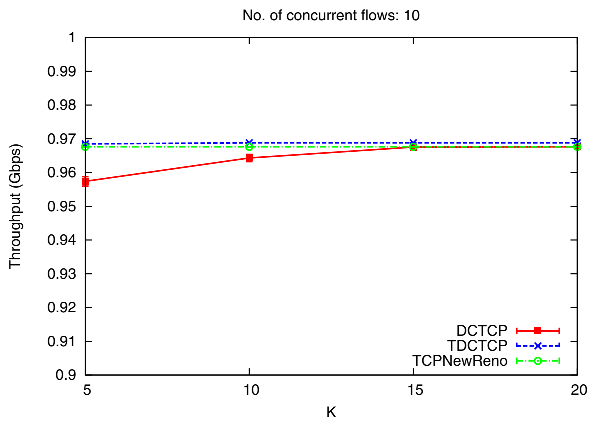
- **Packet delay**: the end-to-end delay experienced by the packets in the network. The variation in packet delay is also studied.
- **Throughput**: the total throughput achieved by all the TCP flows collectively. We calculated throughput as the ratio of the total number of bytes sent by all the flows to the maximum finish time among all the flows. The variation in throughput is also presented.
- **Fairness**: Jain's fairness index (JFI) [17] is a metric to measure the fairness achieved by concurrent flows that are sharing bandwidth. If there are N number of flows each having allocated bandwidth of $x_1, x_2, \ldots x_N$, then:

$$\text{JFI} = \frac{\left(\sum_{i=1}^{N} x_i\right)^2}{N\left(\sum_{i=1}^{N} x_i^2\right)}$$

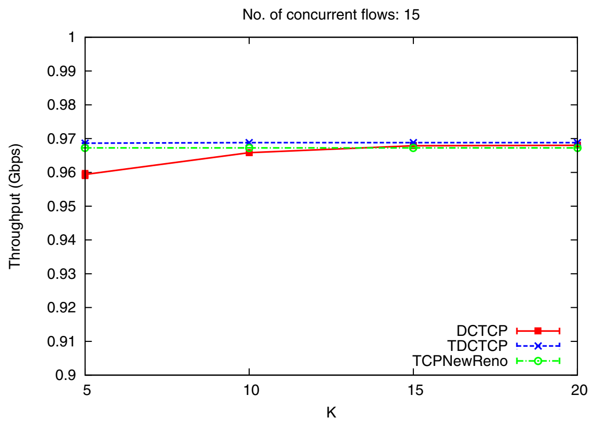
- **Queue Length**: the average number of packets in the queue(s).

A. Throughput

In this subsection, we present the results related to throughput of different TCP algorithms.



(a) 10 flows



(b) 15 flows

Fig. 6: Throughput Single bottleneck (1Gbps); Buffer space=700 KB.

1) Single-Bottleneck Network: The throughput of the three TCP variants for 1 Gbps link capacity is presented in Fig 6. As observed, there is very little difference in throughput among the algorithms. This result conforms to that reported in [12]. The throughput with 10 Gbps link capacity is presented in Fig 7. In this case, there is an improvement in the throughput provided by TDCTCP with respect to DCTCP. TCPNewReno provided the maximum throughput (in most cases). For $K = 6$, TDCTCP provides 26% more throughput than DCTCP, when buffer space is 700 KB and 37% more throughput, when buffer space is 1.5 MB. For $K = 10$, TDCTCP provides 21% more throughput, when buffer space is 700 KB and 19% more throughput, when buffer space is 1.5MB, than DCTCP. For $K \geq 24$, all the three algorithms have approximately same throughput. But TDCTCP gives the same throughput as TCPNewReno at around $K = 16$, whereas DCTCP gives the same throughput as TCPNewReno at around $K = 24$.

2) Multi-Bottleneck: The throughput of the three TCP variants for the multi-bottleneck network are shown in Fig 8. Here too, TCPNewReno provides the highest throughput. However, TDCTCP provides 18% more throughput than DCTCP for $K = 6$; and for $K = 10$, TDCTCP provides 9% more throughput than DCTCP. TDCTCP provides the same throughput as TCPNewReno for $K \geq 14$ whereas DCTCP achieves the same throughput as TCPNewReno for $K \geq 24$.

Summary: Though there is no difference in throughput when link capacity is 1 Gbps, there is a significant difference in throughput when link capacity is 10 Gbps and the value of K is small. TDCTCP gives the same throughput as the TCPNewReno for much lower values of K compared to DCTCP. This increase in throughput can be attributed to the modification as discussed in Section IV-A. Thus, we can use TDCTCP if we want lower delay and low buffer occupancy instead of DCTCP.

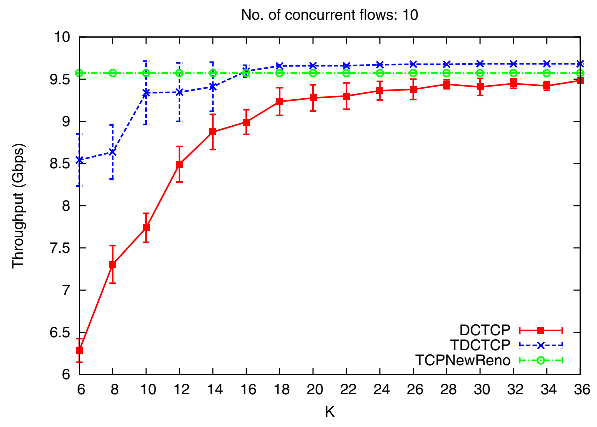
B. Fairness

In this subsection we present the results of fairness provided by the three TCP variants.

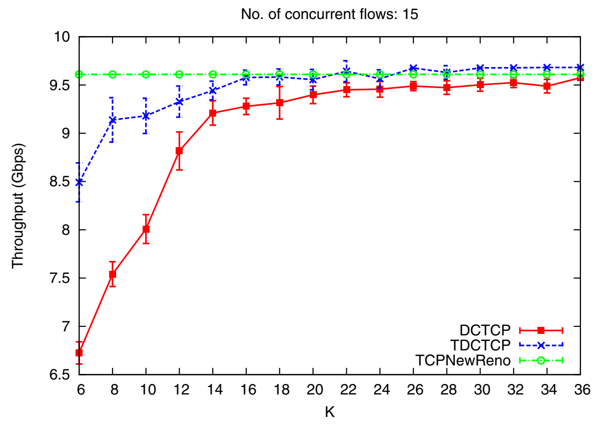
1) Single-Bottleneck Network: As in the case of throughput, there is very little difference in fairness, when the link capacity is 1 Gbps, as seen in Fig. 9(a). For 10 Gbps link capacity, there is a significant difference in fairness as seen in Fig. 9b. TDCTCP's fairness is close to 1.0 for almost all values of K. But DCTCP's fairness is very poor for low values of K. For $K = 6$, TDCTCP's fairness is 20% better than DCTCP's fairness when buffer space is 700 KB and it is 19% better than DCTCP's fairness when buffer space is 1.5 MB. For $K = 10$, TDCTCP's fairness is 18% better than DCTCP's fairness when buffer space is 700 KB and it is 15% better than DCTCP's fairness when buffer space is 1.5 MB. DCTCP's fairness improves as the value of K increases. DCTCP gives the same fairness as TDCTCP/TCPNewReno only when $K \geq 24$.

2) Multi-Bottleneck Network: Fig. 10 presents the fairness results for the three TCP variants. For $K = 6$, TDCTCP's fairness is 16% better than DCTCP's fairness and for $K = 10$, TDCTCP's fairness is 15% better than DCTCP's fairness. TDCTCP achieves same fairness as TCPNewReno for almost every value of K and DCTCP achieves same fairness as TCPNewReno for $K \geq 26$.

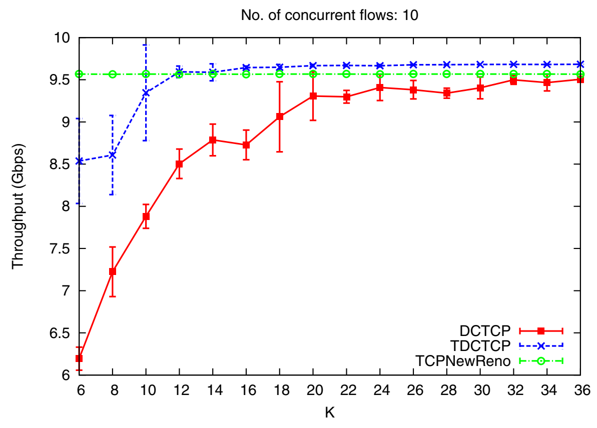
Summary: TDCTCP provides better fairness in every scenario, while DCTCP provides good fairness only for the case where link capacity is 1 Gbps. TDCTCP achieves this



(a) 10 flows; Buffer space=700 KB



(b) 15 flows; Buffer space=700 KB



(c) 10 flows; Buffer space=1.5 MB

Fig. 7: Throughput in Single bottleneck (10Gbps) network, varying number of flows and buffer size.

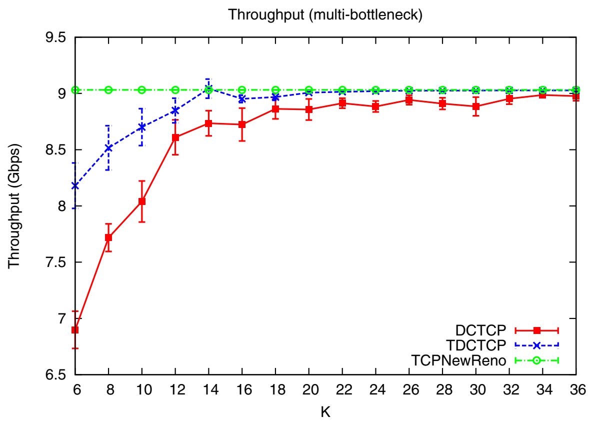


Fig. 8: Throughput in Multi-Bottleneck (10Gbps) network. Buffer space=700 KB

improvement in fairness value due to the modifications as discussed in Sections IV-B and IV-C.

C. Delay

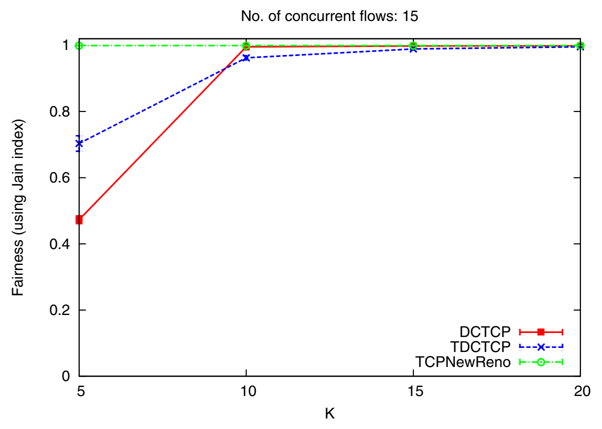
This section compares the delay performance of the three TCP variants. Results are presented only for link capacity of 10 Gbps since there was no significant difference in the performance of DCTCP and TDCTCP with 1 Gbps links.

Fig 11 and 12 present the average packet delay results. The delay provided by TCPNewReno is very high compared to DCTCP and TDCTCP; and is approximately 10 times more than that of DCTCP and TDCTCP. This is due to the higher buffer size used by TCPNewReno. Thus, even though it can obtain higher throughput values as seen earlier, the higher delay is a significant disadvantage. The delay provided by TDCTCP is 1 to 2 μs higher than DCTCP in every scenario as seen in the graphs. This is because TDCTCP uses more buffer space due to the modification discussed in Section IV-A. However, the increase in delay is negligible compared to the throughput and fairness gains seen in the earlier sections.

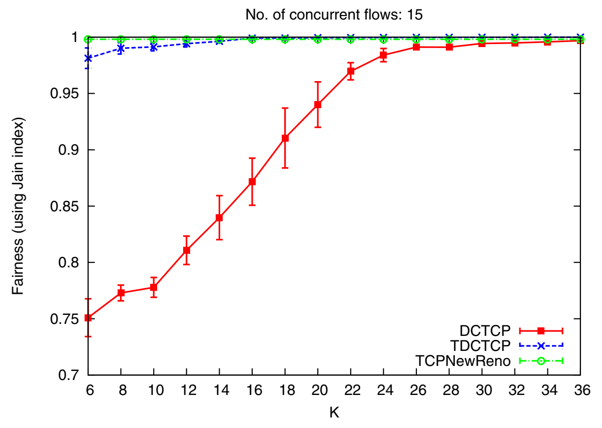
D. Queue Length

In this subsection we present the results related to queue length. All the results are collected from single-bottleneck network with link capacity of 10 Gbps. The results are presented in Fig 13.

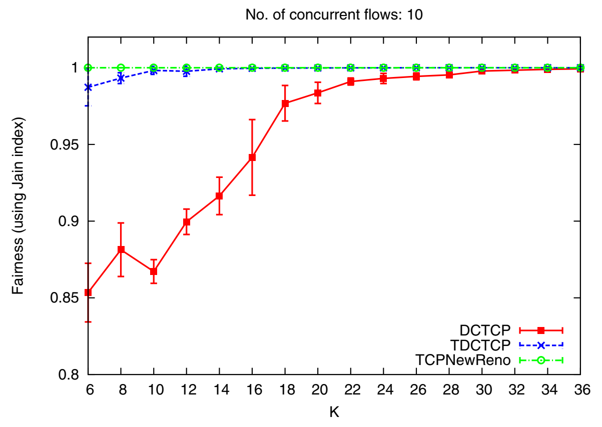
As seen in Fig 13a, for $K = 6$, TDCTCP's mean queue length is approximately 1.01 packets more than that of DCTCP. For $K = 16$, TDCTCP's mean queue length is 1.61 packets more than that of DCTCP. For $K = 24$, TDCTCP's mean queue length is 1.67 packets more than that of DCTCP. Similar trends are observed in Fig. 13b. This increase in queue length for TDCTCP can be attributed to the first modification presented in Section IV-A. Varying the congestion window based on congestion levels, leads to more packets in transit. As a result, the queue length at the link increases. It is also seen that TCPNewReno's mean queue length is 10 to 20 times



(a) 1 Gbps; 15 flows; Buffer space=700 KB



(b) 10 Gbps; 15 flows; Buffer space=700 KB



(c) 10 flows; Buffer space=1.5 MB

Fig. 9: Fairness measured, for single bottleneck network.

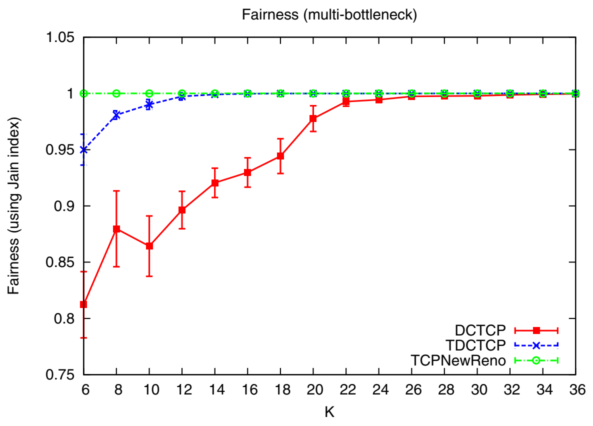


Fig. 10: Fairness in Multi-Bottleneck (10Gbps) network; Buffer space=700 KB

more than that of DCTCP and TDCTCP, due to its larger buffer size.

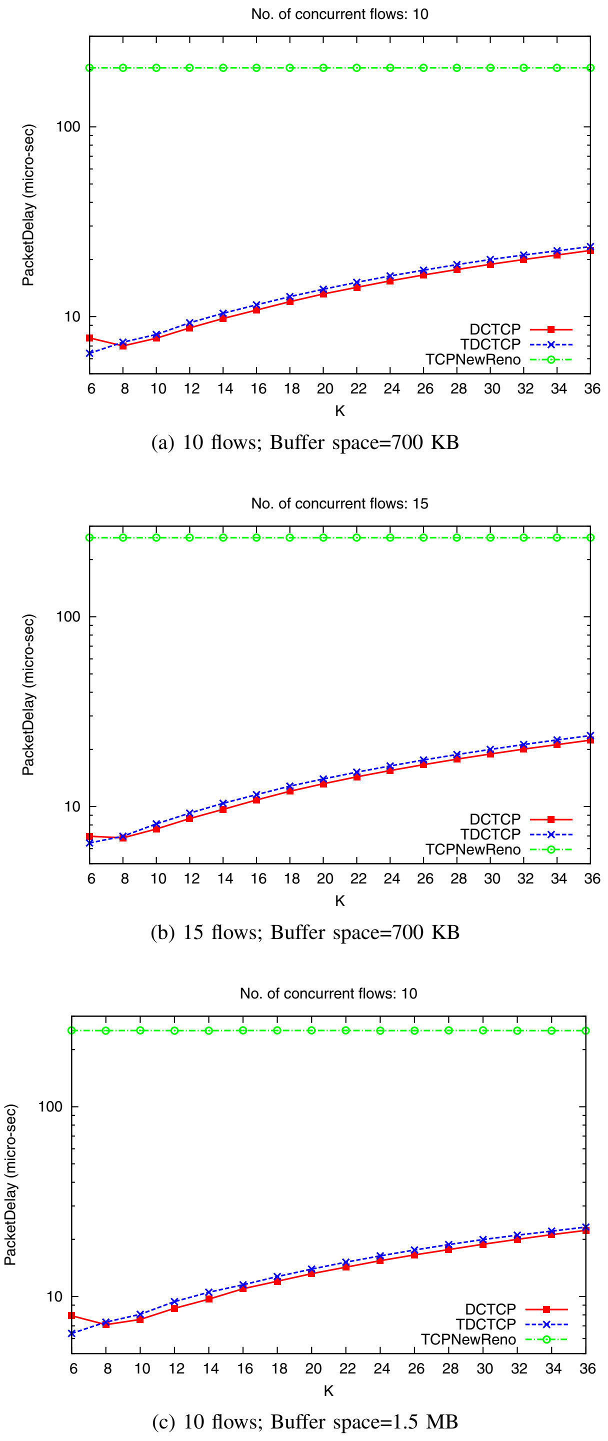
E. *Variation in Delay and Throughput*

In this subsection we present the results related to variation in delay and throughput. All the results presented in this subsection are collected from a single-bottleneck network with link capacity of 10 Gbps.

Variation in Delay: In Fig 14a and 14b, variation in delay for DCTCP is 1.27 μs more than that of TDCTCP at $K = 8$. At $K = 16$, the variation in delay for DCTCP and TDCTCP are almost same. At $K = 24$, TDCTCP's variation in delay is 0.95 μs more than that of DCTCP and at $K = 32$, TDCTCP's variation in delay is 1.02 μs more than that of DCTCP. TCPNewReno's variation in delay is roughly 10 times more than that of DCTCP and TDCTCP.

Variation in Throughput: Fig. 15 presents the variation of throughput for the three TCP variants. It is seen that DCTCP is almost 5 times higher than that of TDCTCP at $K = 16$. At both $K = 24$ and $K = 32$, DCTCP's variation of throughput is roughly 4 times higher than that of TDCTCP. TCPNewReno's variation in throughput is same as TDCTCP's variation in throughput and almost 4 times lower than that of DCTCP's at $K = 8$ in Fig 15a. TCPNewReno's variation in throughput is roughly 2 times lower than that of DCTCP's and almost 2 times higher than that of TDCTCP's in Fig 15a. But in case of 15 concurrent flows, TCPNewReno's variation in throughput increases as seen in Fig 15b.

Summary: The variation of delay for TDCTCP is slightly higher than that of DCTCP. TCPNewReno's variation in delay is 10 times higher than that of DCTCP and TDCTCP. The variation of throughput for DCTCP is 4 to 5 times higher than that of TDCTCP. TCPNewReno's variation in throughput is lower than DCTCP's variation in throughput but it is higher than TDCTCP's variation in throughput. TDCTCP achieves more stable throughput due to the modifications as discussed in Sections IV-B and IV-C.



(a) 10 flows; Buffer space=700 KB

(b) 15 flows; Buffer space=700 KB

(c) 10 flows; Buffer space=1.5 MB

Fig. 11: Delay performance for single bottleneck network, with 10 Gbps link capacity.

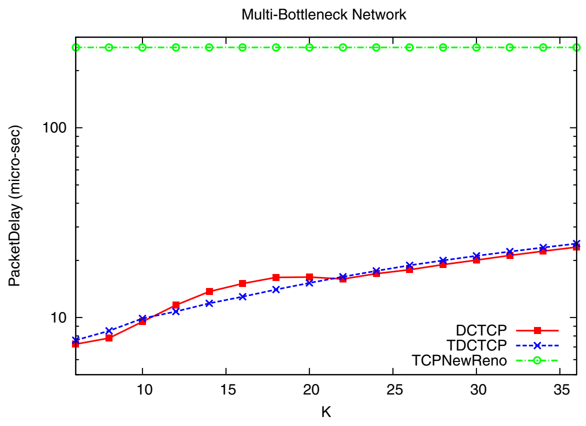
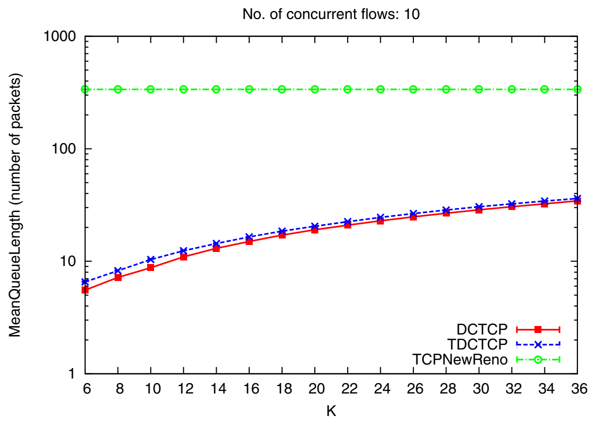
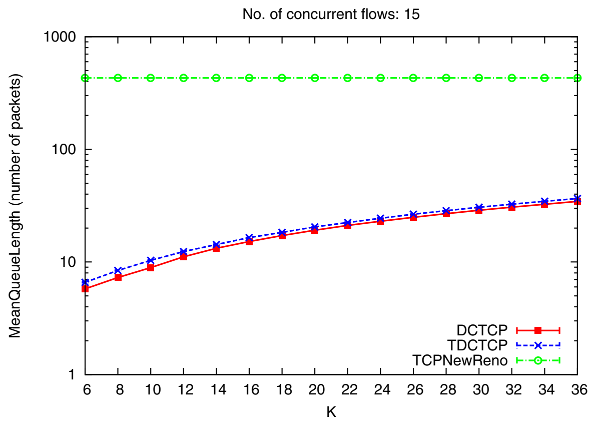


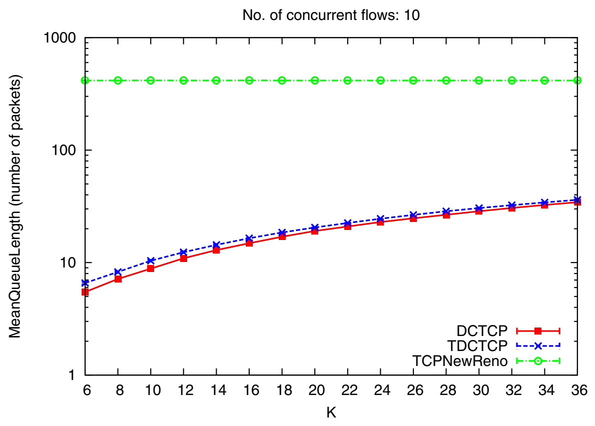
Fig. 12: Packet delay in Multi-Bottleneck (10Gbps). Buffer space=700 KB



(a) 10 flows; Buffer space=700 KB



(b) 15 flows; Buffer space=700 KB



(c) 10 flows; Buffer space=1.5 MB

Fig. 13: Mean Queue Length in Single-bottleneck (10Gbps) network.

VI. Conclusions

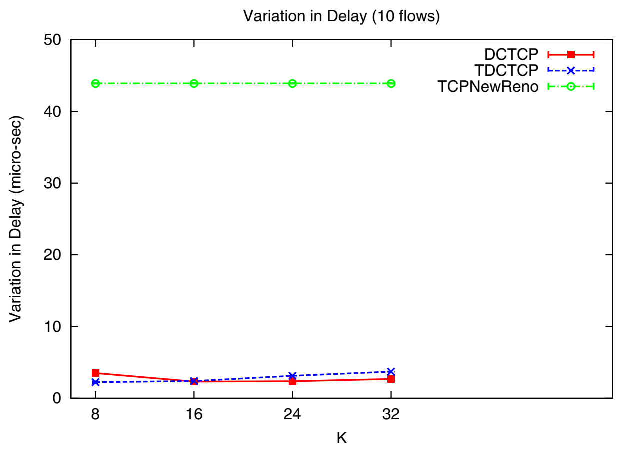
In this paper, we have attempted to improve the performance of the DCTCP [12], a variant of TCP that was specifically designed for data centers. Our improved TCP variant is called TDCTCP. Changes were introduced in DCTCP's congestion control algorithm and in dynamic delayed ACK timeout calculation. We tested the performance of DCTCP, TDCTCP and TCPNewReno using OMNeT++ simulator. We used single-bottleneck and multi-bottleneck network to evaluate the performance of these three algorithms. It was shown using simulation based experiments that TDCTCP provided 15% higher throughput and improved fairness when compared to DCTCP. Though TDCTCP provides good fairness for every value of K, DCTCP's fairness is very poor when the value of K is small. TDCTCP's performance converges to TCPNewReno's performance for smaller values of K, when compared to DCTCP. TDCTCP also provides more stable throughput than DCTCP and TCPNewReno. However, TDCTCP's queue length is slightly more than that of DCTCP in the network with 10 Gbps links. As a result, the packet delay also increases slightly. In summary, TDCTCP's delay is slightly higher than that of DCTCP, but TDCTCP's throughput is higher than that of DCTCP for the same value of K.

As part of future work, the performance of TDCTCP and DCTCP can be studied using testbed implementation. The performance can also be analyzed for other data center network topologies such as Fattree [1], DCell [3] or BCube [4]. A mathematical model of the congestion control algorithm and stability study of the congestion window are also worthy of further study. We also need to evaluate the performance of TDCTCP for different kinds of flow dynamics.

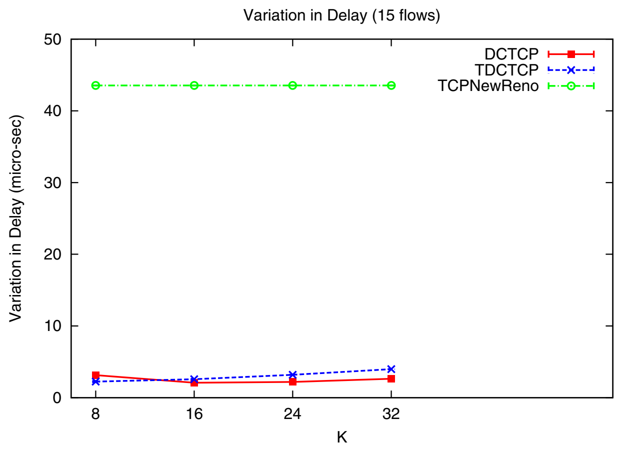
Acknowledgments: This work was supported by India-UK Advanced Technology Centre of Excellence in Next Generation Networks, Systems and Services (IU-ATC).

References

[1] M. Al-Fares, A. Loukissas, and A. Vahdat, "A Scalable, Commodity Data Center Network Architecture," in *Proc. of ACM SIGCOMM*,

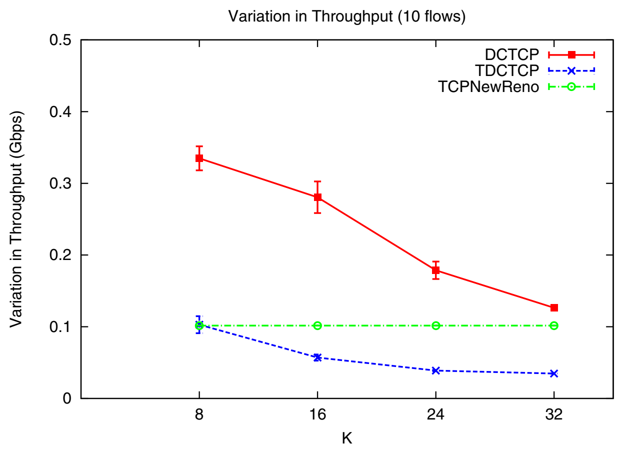


(a) 10 flows

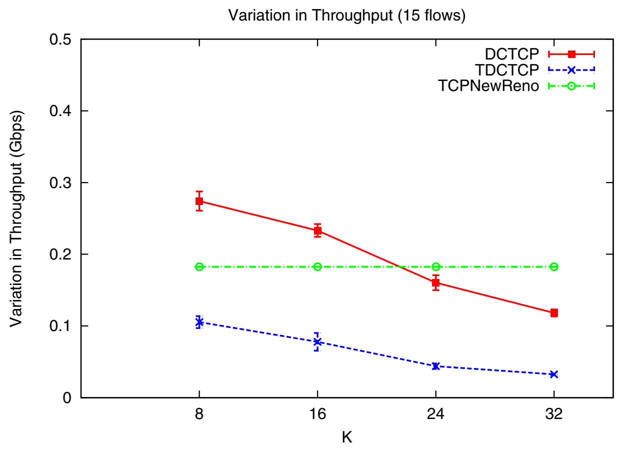


(b) 15 flows

Fig. 14: Variation in Delay in Single-bottleneck (10Gbps); Buffer space=700 KB.



(a) 10 flows



(b) 10 flows

Fig. 15: Variation in Throughput in Single-bottleneck (10Gbps); Buffer space=700 KB.

Seattle, USA, Aug. 2008.

[2] "Mapreduce," Dec. 2011, http://en.wikipedia.org/wiki/MapReduce.

[3] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "DCell: A Scalable and Fault-Tolerant Network Structure for Data Centers," in *Proc. of ACM SIGCOMM*, Seattle, USA, Aug. 2008.

[4] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers," in *Proc. of ACM SIGCOMM*, Barcelona, Spain, Aug. 2009.

[5] T. Benson, A. Anand, A. Akella, and M. Zhang, "Understanding data center traffic characteristics," in *Proc. of the 1st ACM workshop on Research on enterprise networking*, Barcelona, Spain, Aug. 2009.

[6] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. of ACM SIGCOMM Internet Measurement Conference (IMC)*, Melbourne, Australia, Nov. 2010.

[7] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *Proc. of ACM SIGCOMM Internet Measurement Conference (IMC)*, Chicago, Illinois, USA, Nov. 2009.

[8] 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 *Proc. of ACM SIGCOMM*, Barcelona, Spain, Aug. 2009.

[9] Y. Chen, R. Griffith, J. Liu, and R. H. Katz, "Understanding TCP Incast Throughput Collapse in Datacenter Networks," in *Proceedings of ACM workshop on Research on enterprise networking (WREN)*, Barcelona, Spain, Aug. 2009, pp. 73–82.

[10] H. Wu, Z. Feng, C. Guo, and Y. Zhang, "ICTCP: Incast Congestion Control for TCP in data center networks," in *Proc. of the ACM CoNEXT*, Philadelphia, PA, Nov. 2010.

[11] C. Raiciu, C. Pluntke, S. Barre, A. Greenhalgh, D. Wischik, and M. Handley, "Data center networking with multipath TCP," in *Proc. of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks*, Monterey, California, Oct. 2010.

[12] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center TCP (DCTCP)," in *Proc. of ACM SIGCOMM 2010*, New Delhi, India, Aug. 2010.

[13] K. Ramakrishnan, S. Floyd, and D. Black, "RFC 3168 : The Addition of Explicit Congestion Notification (ECN) to IP," http://www.ietf.org/rfc/rfc3168.txt, Sept. 2001.

[14] "OMNeT++ Network Simulator," http://www.omnetpp.org/, Sep. 2012.

[15] R. Braden, "RFC 1122: Requirements for Internet Hosts – Communication Layers," http://tools.ietf.org/html/rfc1122, Oct. 1989.

[16] "INET Framework," Dec. 2011, http://inet.omnetpp.org/.

[17] "Fairness measure," Dec. 2011, http://en.wikipedia.org/wiki/Fairness_measure.