Explicit Multipath Congestion Control for Data Center Networks

Yu Cao[1,2,4], Mingwei Xu[1,4], Xiaoming Fu[3], Enhuan Dong[1,4]

[1]Tsinghua University, China
[2]National Digital Switching System Engineering & Technological Research Center, China
[3]Institute of Computer Science, University of Goettingen, Germany
[4]Tsinghua National Laboratory for Information Science and Technology (TNList), China
{caoyu08, xmw, dongenhuan}@csnet1.cs.tsinghua.edu.cn
fu@cs.uni-goettingen.de

ABSTRACT

The vast majority of application traffic in modern data center networks (DCNs) can be classified into two categories: throughput-sensitive large flows and latency-sensitive small flows. These two types of flows have the conflicting requirements on link buffer occupancy. Existing data transfer proposals either do not fully utilize the path diversity of DCNs to improve the throughput of large flows, or cannot achieve a controllable link buffer occupancy to meet the low latency requirement of small flows. Aiming to balance throughput with latency, we develop the eXplicit MultiPath (XMP) congestion control scheme for DCNs. XMP comprises two components: the BOS algorithm brings link queue buffers consumed by large flows under control, while the TraSh algorithm is responsible for shifting traffic from more congested paths to less congested ones, thus achieving high throughput. We implemented XMP and evaluated its performance on traffic shifting, fairness, goodput, buffer occupancy and link utilization by conducting comprehensive experiments and simulations. The results show that XMP outperforms existing schemes and achieves a reasonable tradeoff between throughput and latency.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols; C.4 [**Performance of Systems**]: Design studies, Modeling techniques

Keywords

multipath; congestion control; data center networks

1. INTRODUCTION

To meet the requirements of cloud applications such as massive data processing and large-scale distributed computation, many organizations have built their own private networks [10, 12, 15]: Data Center Networks (DCNs). A DCN commonly comprises tens of thousands of machines interconnected with commodity switches. DCNs are very different from the Internet in many respects [2, 7], such as bandwidth, latency, topologies and traffic patterns. Because of these differences, DCNs attract considerable attention and become a research hotspot.

There are mainly two types of application demands [4, 30] for high performance communications in DCNs. One is bulk data transfer, such as virtual machine migration and data storage synchronization for the purpose of management, which generates long-lived large flows. As evidenced in [7], most of the bytes are in the top 10% of large flows and at least 20% of the flows are more than 11 seconds long. The other is many-to-one or one-to-many short-lived communication, such as Map-Reduce applications [9], which creates lots of highly dynamic small flows. On one hand, large flows want to achieve as high throughput as possible, resulting in a full buffer occupancy in link queues. Small flows, on the other hand, are much more concerned with latency, because they each have only tens of kilobytes of data for transfer, and more importantly, they are commonly constrained by a completion deadline specified by latency-sensitive applications [30, 33]. Unlike in the Internet, packet queuing delay predominates Round Trip Time (RTT) in DCNs. Taking Gigabit links for example, one buffered packet will increase RTT by $12\mu s$, which accounts for more than one tenth of inner-rack RTT. The requirement of low latency implies that link buffer occupancy should be as low as possible, which will decrease link utilization. Therefore, one of the major challenges facing data transfer in DCNs is the tradeoff between throughput and latency.

Towards addressing this challenge, we argue it is a reasonable choice to maintain a low level of link buffer occupancy. First, the requirement of low latency has a higher priority than that of high throughput. Considering user experiences and commercial revenues, many online services hosted in DCNs require the flows to accomplish data delivery within the deadline of tens of milliseconds [30]. Thus, large flows have to suffer from bandwidth loss due to fairly low link buffer occupancy. As an extreme example, Alizadeh *et al.* [5] proposed to sacrifice about 10% of bandwidth for reducing packet queuing delay down to zero. Second, link queues with plenty of free buffer space can effectively accommodate the traffic burstiness caused by Incast-like communication

CoNEXT'13, December 09–12, 2013, Santa Barbara, California, USA.

http://dx.doi.org/10.1145/2535372.2535384.

patterns [31]. Finally, the throughput of large flows can be improved by employing multipath data transfer schemes, rather than using saturating link queues.

A remarkable feature of DCNs is the richness of paths between hosts. For example, the Fat Tree architecture [2] achieves a bandwidth oversubscription ratio of 1:1. It has $k^2/4$ equal-cost paths between a pair of inter-pod hosts, where k is the amount of ports per switch. By exploiting path diversity, Raiciu *et al.* [25] demonstrated that Multipath TCP (MPTCP) [1] is promising for significantly improving the performance of bulk data transfer in DCNs. An MPTCP flow can split its traffic into several subflows, each of which goes through a different path available between the source and the destination. The congestion control algorithm of MPTCP, called Linked Increases (LIA) [34], is responsible for adaptively adjusting the transmission rate of each subflow in an attempt to shift traffic from more congested paths to less congested ones, thus improving throughput and link utilization.

We believe the server side (e.g., content providers) will be highly incentivized to utilize high capacity links instead of keeping their customers constantly complaining Quality of Experience (QoE) dissatisfactions. In case they encounter frequent single path saturation or server overloading, the natural choice for servers will be to upgrade their infrastructure. As a result, MPTCP could leverage the wide existence of multiple paths at the core and the aggregation layer in DCNs to achieve higher throughput.

However, LIA is designed originally for the Internet and by nature based on TCP-Reno. Thus, it does not take into account the tradeoff between throughput and latency. On one hand, LIA exhausts link buffer to achieve a full link utilization, resulting in a considerably large RTT and a high packet loss probability which both adversely impact the performance of small flows. On the other hand, if LIA wants to limit its buffer occupancy below an acceptable level, its 50% window reduction in response to congestion will cause link under-utilization.

DCTCP [4] was developed to accommodate the conflicting requirements of throughput-sensitive large flows and latency-sensitive small flows. However, it is a *single-path* data transfer scheme. As shown in our simulations, DCTCP cannot fully utilize path diversity in DCNs, thus wasting some links, especially when several flows collide on the same link.

This paper aims to design a new multipath congestion control scheme for DCNs under the constraint of a controllable link buffer occupancy. LIA follows an ad-hoc design based on the three predetermined performance goals given in [34]. In contrast, our scheme, eXplicit MultiPath (**XMP**) congestion control, is developed using the network utility maximization model [21, 22]. First, we propose the Buffer Occupancy Suppression (BOS) algorithm, which employs the Explicit Congestion Notification (ECN) mechanism [17] to control link buffer occupancy. Next, we find the utility function of BOS and then "multi-path-lize" it in the context of MPTCP. Finally, based on the above results, we construct the Traffic Shifting (TraSh) algorithm to couple subflows so as to move the traffic of an MPTCP flow from its more congested paths to less congested ones. The BOS and the TraSh algorithm together constitute the XMP scheme. BOS brings link queue buffers consumed by XMP under control so as to meet the low latency requirement of small flows and to reduce packet loss probability. TraSh provides for XMP the ability to shift traffic between multiple paths, thus improving the throughput of large flows.

Our previous work in [8] employed the network utility maximization model to establish a general framework for designing multipath congestion control algorithms. This paper uses this framework to develop XMP with specifically addressing DCN's distinct characteristics. Our main original contribution is exploiting the BOS and the TraSh algorithm together to achieve a high throughput for large flows and a low latency for small flows, forming a practical multipath congestion control scheme (XMP) for DCNs. Note that, in this paper, large flows use MPTCP (with XMP or LIA) while small flows use TCP to transfer data.

TraSh sits on BOS. This makes TraSh differ from LIA. Although XMP and LIA both can achieve traffic shifting, their performance is different. For example, Raiciu *et al.* proposed that an MPTCP flow with LIA needs 8 paths [25] to obtain a good utilization in the Fat Tree network. Our simulations show that XMP doesn't need so many subflows. The increase in the amount of subflows will make more small flows miss their completion deadline.

We implemented XMP in both MPTCP_v0.86[1] [27] and the NS-3.14 simulation platform [29]. Our source code can be found in [28]. By conducting comprehensive experiments and simulations, we evaluated the performance of XMP on traffic shifting, fairness, goodput, buffer occupancy and link utilization. Our results show that XMP outperforms existing schemes and can effectively balance throughput with latency.

The remainder of the paper is organized as follows. In Section 2, we propose the BOS algorithm and the TraSh algorithm. In Section 3, we present the implementation of XMP in details. Section 4 and Section 5 are dedicated to experiments and simulations, respectively. We briefly overview related work in Section 6. Finally, we discuss the future work and conclude the paper in Section 7.

2. ALGORITHMS OF XMP

Our goal is to achieve effective traffic shifting under the constraint of a controllable buffer occupancy in link queues. Next we will elaborate on the design principles of XMP towards this goal.

2.1 Suppressing Link Buffer Occupancy

A straightforward method for suppressing link buffer occupancy is to employ Active Queue Management algorithms, such as Random Early Detection (RED) [11], to throttle the transmission rate of sources with the aid of ECN. The ECN+RED has been implemented in many types of forwarding devices as well as host protocol stacks. Traditionally, the congestion control loop works as follows. RED performs the Exponentially Weighted Moving Average (EWMA) algorithm to estimate the average link buffer occupancy, according to which RED assigns a probability to marking the arriving packet with the Congestion Experienced (CE) codepoint in the IP header. When receiving a CE signal, the destination feeds it back to the source by setting the ECN Echo (ECE) codepoint in the TCP header of *each* acknowledgement packet. The source reduces its congestion window (*cwnd*) by half in response to ECE signals. And meanwhile it sets the Congestion Window Reduced (CWR) codepoint

[1]It is based on Linux Kernel 3.5.

in the TCP header of the sending packet in order to inform the destination of ceasing sending ECE signals.

Based upon the following three reasons, we do not copy the above-mentioned standard control process in our scheme. First, a path in DCNs is often dominated by very few large flows [7]. This leads to quite a low degree of statistical multiplexing on links. Since we want to maintain a low level of buffer occupancy, halving *cwnd* will cause the average packet arrival rate to be far away from link capacity, hence an under-utilization of network resources. Second, the main cause of packet loss events in DCNs is the temporal traffic burstiness [7, 24, 31] due to Incast-like communication patterns. This implies the average queue length estimated by EWMA is not an appropriate congestion metric for switches, especially in the network with ultra-low RTT (several hundreds of microseconds) and a low degree of statistical multiplexing. Third, it is difficult to determine the desirable parameters for RED to avoid instability [23] when the level of link buffer occupancy is low.

Kuzmanovic [20] first proposed to use the instantaneous queue length as the congestion metric to mark packets. This idea was lately adopted by DCTCP, which translates the amount of ECN marks received by the source into the congestion extent of networks so as to proportionally reduce *cwnd*. Although DCTCP is quite an ingenious method that can solve the tradeoff between throughput and latency in DCNs, we found that DCTCP is sensitive to the parameter settings and may converge to the state of unfair bandwidth allocation. An example is given in Figure 1(a) and 1(b). We think the reasons are two-fold. First, a relatively small reduction in *cwnd* increases the time that the competing flows take to converge to the fairness state. If the global synchronization [11] happens [2] before the end of convergence, every flow will stay in the unfairness state until the occurrence or the disappearance of a flow breaks the current global synchronization. Second, the *cwnd* reduction factor and the amount of marked packets may have a mutual influence on each other for DCTCP. This interaction effect may lead to an inconsistent congestion estimate that the different flows make for the same bottleneck link.

Actually, we can achieve a comparable link utilization to DCTCP using a constant factor to reduce *cwnd*, providing that the marking threshold K is set to an appropriate value. Taking the case in Figure 1 for example, the Bandwidth Delay Product (BDP) is about $1Gbps \times 225\mu s/(8 \times 1500) \approx 19$ packets. So if $K = 20$, halving *cwnd* still can fully utilize link capacity, as shown in Figure 1(d). It is interesting that the result in Figure 1(c) is also pretty good. We think it can be explained by two facts. First, the minimal *cwnd* is about $(19 + 10)/2 \approx 15$ packets, which is only 4 packets lower than the BDP. Second, a smaller K leads to a smaller RTT, hence a faster growth rate of *cwnd*, which significantly compensates for utilization loss due to halving *cwnd*. As mentioned before, packet queuing delay predominates RTT in DCNs. In the above example, changing K from 20 down to 10 will decrease RTT by $10 \times 1500 \times 8/1Gbps \approx 120\mu s$, which accounts for more than half of RTT.

Generally, suppose *cwnd* is reduced by a factor of $1/\beta$ in response to ECN marks. In order to achieve a full link utilization, the marking threshold K should satisfy $(K + BDP)/\beta \leq K$. So we have

$$K \geq \frac{BDP}{\beta - 1}, \quad \beta \geq 2. \tag{1}$$

A larger β, in general, corresponds to a smaller lower-bound of K, hence a smaller latency and a sufficient free space in link queues to accommodate the burstiness of small flows. However, similar to DCTCP, if β is too large, the convergence time will significantly increase and the bandwidth may be unfairly shared by competing flows. Considering *cwnd* changes with packet granularity, we think it is sufficient to choose an appropriate β from the integers between 3 and 5. In DCNs, hosts are connected to switches commonly using links of $1Gbps$ and RTT is less than $400\mu s$ [31, 36]. So the BDP is about 33 packets. We choose $\beta = 4$ and $K = 10$.

Motivated by these observations, we propose the following Buffer Occupancy Suppression algorithm (**BOS**) as a starting point to design XMP.

1) **Packet Marking Rules**: The switch marks the arriving packet with the CE codepoint if the *instantaneous* queue length of the outgoing interface is larger than K packets; otherwise, it does not perform packet marking.

2) **ECN Echo**: The destination feeds its received CEs back to the source through the ECE and the CWR codepoint of acknowledgement packets. The two bits can encode at most 3 CEs.

3) **Slow Start**: The source increases *cwnd* by one if receiving an acknowledgement packet whose ECE and CWR are both zero; otherwise, it stops increasing *cwnd*, and meanwhile enters Congestion Avoidance.

4) **Congestion Avoidance**: The source decreases *cwnd* by a factor of $1/\beta$ if receiving an acknowledgement packet whose ECE and CWR are *not* both zero; otherwise, it increases *cwnd* by δ on the end of a round. Note that the source decreases *cwnd* only once in a round, as needed.

We define a "round" as the interval from the time of sending out a specified packet to the time of receiving its acknowledgement. Please see Section 3 and Figure 2 for more details. Note that the standard process of ECN is *changed* in BOS. Specifically, instead of setting ECE in every acknowledgement packet until receiving CWR from the source, the destination echoes exactly each received CE to the source. Thus, the standard function of the CWR codepoint is no longer required. We employ it together with ECE to encode the amount of CEs. Two bits are sufficient since the destination sends back to the source one cumulative acknowledgement for at most every two consecutively received packets[3].

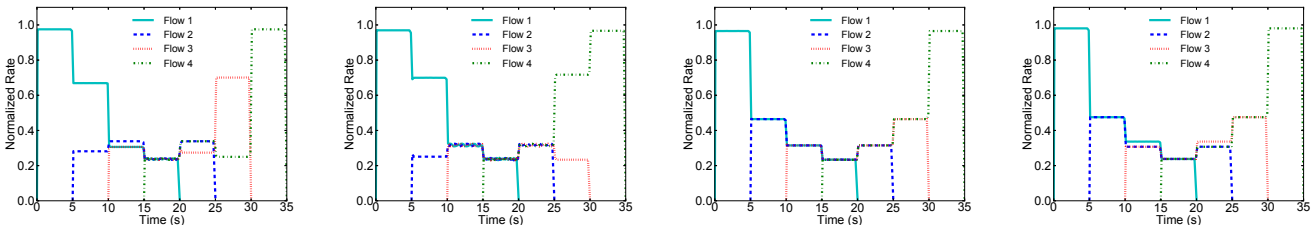
We use differential equations to formulate the *cwnd* evolution of BOS in the phase of congestion avoidance:

$$\frac{dw(t)}{dt} = \frac{\delta}{T}(1 - p(t)) - \frac{w(t)}{T\beta}p(t), \tag{2}$$

where $w(t)$ is the size of *cwnd*, T is the time interval of a round (roughly one RTT) and $p(t)$ is the probability that at least one packet is marked in a round. Since BOS attempts to hold the queue length at about K packets, the queuing

[2]It is common when using Droptail queues in the condition of the low degree of statistical multiplexing on links.

[3]This is due to the Delayed ACKs mechanism. In contrast to our two-bit encoding method, DCTCP uses a state machine to infer the amount of CEs.



(a) DCTCP, $K = 10$. (b) DCTCP, $K = 20$. (c) Halving cwnd, $K = 10$. (d) Halving cwnd, $K = 20$.

Figure 1: Four flows compete for a bottleneck link with capacity of $1Gbps$. We start or stop a flow with an interval of $5s$. Whenever the number of flows changes, the current equilibrium state is broken and then a new equilibrium state will be established. RTT is about $225\mu s$ with no packet queuing delay. If the instantaneous queue length in switches is larger than K, the arriving packet is marked; otherwise, it is not marked.

delay doesn't change much. Thus, we assume T is a constant related to the path. At the equilibrium point, we have

$$\tilde{p} = \frac{1}{1 + \tilde{w}/\delta\beta}. \tag{3}$$

So the utility function [21] of BOS is

$$U(x) = \frac{\delta\beta}{T} \log\left(1 + \frac{T}{\delta\beta}x\right), \tag{4}$$

where $x = \tilde{w}/T$ is the bandwidth obtained by a flow. Clearly, $U(x)$ is increasing, strictly concave and twice continuously differentiable in the nonnegative domain.

It is worth further explaining $p(t)$. Traditionally, a packet marking/dropping probability $q(t)$ is assumed to quantify the congestion extent of networks. Given that $q(t)$ is sufficiently small and the packets are marked/dropped independently of each other, equation $\tilde{p} = 1 - (1 - \tilde{q})^{\tilde{w}} \approx \tilde{q}\tilde{w}$ holds. In DCNs, however, a window of packets issued by a source commonly arrives at switches in batches. Once one of the packets is marked, all the succeeding ones are also most likely marked. This implies that the assumption of independent packet marking does not hold. Thus, we use $p(t)$, instead of $q(t)$, as the congestion metric in DCNs.

2.2 Coupling Subflows for Traffic Shifting

Based on the BOS algorithm, we next present how to achieve traffic shifting by means of coupling the subflows belonging to an MPTCP flow.

An MPTCP flow can split its traffic into several subflows, each of which goes through a different path. If the subflows perform congestion control independently of each other, the fairness will be violated. Thus, we think that a major objective of multipath congestion control is to couple all the subflows belonging to a flow to achieve both the efficiency and the fairness goal. In [8, Section I], we provided two examples for explaining these two goals. Not only does coupling subflows make every flow compete for bandwidth in a consistent way, irrespective of the amount of subflows, but also it gives multihomed end-hosts an ability to shift traffic from more congested paths to less congested ones. Thus, MPTCP is considered promising for load balancing in DCNs.

We employ the network utility maximization model to couple subflows. Suppose flow s transfers data along path r[4] at rate $x_{s,r}$. So its total rate is $y_s = \sum_{r \in R_s} x_{s,r}$ and it can obtain an utility $U(y_s)$. Each link l has a finite capacity of c_l. All the paths are formulated by matrix $\mathbf{A}$, where $a_{l,r} = 1$ if path r goes through link l, and $a_{l,r} = 0$ otherwise. Path set R_s is given by matrix $\mathbf{B}$, where $b_{s,r} = 1$ if path r is used by flow s, and $b_{s,r} = 0$ otherwise. The objective of congestion control is to determine rate $\mathbf{X}$ for maximizing the total utility subject to link capacity constraints:

$$\begin{aligned} &\max_{\mathbf{X} \geq \mathbf{0}} \sum_{s \in S} U(y_s) \\ &s.t.\ \mathbf{Y} = \mathbf{B}\mathbf{X} \\ &\quad\ \mathbf{A}\mathbf{X} \leq \mathbf{C}. \end{aligned} \tag{5}$$

Based upon this model, we established a general framework in [8] for designing multipath congestion control algorithms. The framework consists of three key components as follows. Please refer to [8] for more details.

1) **Congestion Metric**: It determines how the source quantifies the congestion extent of paths using congestion signals that are explicitly or implicitly "announced" by links.

2) **Adjustable Parameters**: The source changes *cwnd* of each subflow in response to congestion signals. In other words, the source needs a set of parameters as a knob to controlling the bandwidth obtained by each subflow.

3) **Congestion Equality Principle**: If each flow strives to equalize the congestion extent that it perceives on every available paths by means of traffic shifting, network resources will be fairly and efficiently shared by all the flows. This principle guides the source to adjust the knob parameters along the direction of convergence.

Following the framework, we first choose $p(t)$ as the congestion metric. According to (4), we construct the utility function of XMP as follows:

$$U(y_s) = \frac{\beta}{T_s} \log\left(1 + \frac{T_s}{\beta}y_s\right), \tag{6}$$

where $T_s = \min\{T_{s,r}, r \in R_s\}$ and $T_{s,r}$ is the smoothed RTT of path r measured by flow s. The derivative

$$U'(y_s) = \frac{1}{1 + y_s T_s/\beta} \tag{7}$$

can be interpreted as the expected congestion extent as though flow s transferred data at rate y_s along a virtual

[4]We use r to denote both the path and the subflow running on it.

single path whose RTT is T_s and whose capacity is the sum of all the paths used by flow s.

Suppose subflow r changes its *cwnd* according to the BOS algorithm with parameter $\delta_{s,r}$ and parameter β. So at the equilibrium point we have

$$\tilde{p}_{s,r} = \frac{1}{1 + x_{s,r}T_{s,r}/\delta_{s,r}\beta}. \quad (8)$$

Clearly, a larger $\delta_{s,r}$ can help flow s obtain more bandwidth on path r. Conversely, a smaller $\delta_{s,r}$ may lead to less bandwidth. In other words, $\delta_{s,r}$ indicates how aggressively flow s competes for bandwidth with other flows on path r. Thus, we choose $\delta_{s,r}$ as the knob to controlling the bandwidth obtained by flow s on path r.

The Additive-Increase Multiplicative-Decrease rule makes TCP flows converge to the equilibrium point that satisfies both fairness and efficiency in a distributed way. Similarly, the Congestion Equality Principle guides each MPTCP flow to shift its own traffic among multiple paths with only local knowledge on network status, achieving both fairness and efficiency in the context of multipath data transfer. Following this principle, if $\tilde{p}_{s,r} > U^{'}(y_s)$, $\delta_{s,r}$ should be decreased so as to offload a part of traffic; if $\tilde{p}_{s,r} < U^{'}(y_s)$, $\delta_{s,r}$ should be increased so as to moderately onload more traffic. Theoretically, δ will ultimately converge to the point at which all the paths used by flow s are equally congested[5], i.e. $\tilde{p}_{s,r} = U^{'}(y_s)$ for each $r \in R_s$. Letting (7) = (8) yields

$$\delta_{s,r} = \frac{T_{s,r}x_{s,r}}{T_s y_s}. \quad (9)$$

According to this result, we design the Traffic Shifting algorithm (**TraSh**) as follows.

1) **Initialization**: At step $t = 0$, flow s sets $\delta_{s,r}(0) = 1$ for each $r \in R_s$; go to 2).

2) **Rate Convergence**: At step t, flow s performs the BOS algorithm independently on each subflow until achieving rate convergence, namely $x_{s,r}(t) = \frac{\beta\delta_{s,r}(t)\left(1-p_{s,r}(t)\right)}{T_{s,r}p_{s,r}(t)}$ for each $r \in R_s$; go to 3).

3) **Parameter Adjustment**: At step t, flow s calculates the total rate $y_s(t) = \sum_{r\in R_s} x_{s,r}(t)$ and finds out $T_s = \min\{T_{s,r}, r \in R_s\}$. It then adjusts $\delta_{s,r}$ for each $r \in R_s$ using $\delta_{s,r}(t+1) = \frac{T_{s,r}x_{s,r}(t)}{T_s y_s(t)}$; go to 4).

4) **Iteration**: $t \leftarrow t + 1$; go to 2).

PROPOSITION 1. *If $p_{s,r}(t) < U^{'}(y_s(t))$, then $\delta_{s,r}(t+1) > \delta_{s,r}(t)$.*

PROOF. Substituting (7) and (8) into $p_{s,r}(t) < U^{'}(y_s(t))$ yields $\frac{T_{s,r}x_{s,r}(t)}{\delta_{s,r}(t)} > T_s y_s(t)$. Thus, $\frac{T_{s,r}x_{s,r}(t)}{T_s y_s(t)} = \delta_{s,r}(t+1) > \delta_{s,r}(t)$ holds. □

Since $\delta_{s,r}$ controls the bandwidth obtained by flow s on path r, Proposition 1 proves that TraSh follows the Congestion Equality Principle and it can effectively achieve traffic shifting.

[5]An MPTCP flow should give up the path whose congestion extent is always higher than that of its other paths. In practice, however, it is more reasonable to set 2 packets as the lower-bound of *cwnd*.

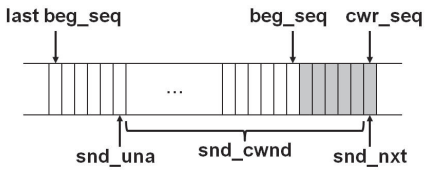


Figure 2: A round. *snd_una* **is the highest unacknowledged sequence number.** *snd_nxt* **is the sequence number of the next packet that will be sent.** *beg_seq* **records a specified packet sequence number. When** *snd_una* > *beg_seq*, **a round ends. And meanwhile,** *beg_seq* **is updated using** *snd_nxt*. *snd_cwnd* **is the congestion window.** *cwr_seq* **is used to avoid undesired reduction in** *snd_cwnd* **when ECN signals are received.**

3. IMPLEMENTATION OF XMP

For an MPTCP flow, each subflow adjusts its *cwnd* according to the BOS algorithm while its parameter δ is tuned by the TraSh algorithm. These two algorithms together constitute the XMP scheme. Next, we present the implementation details of XMP in Linux Kernel. The pseudo-code of XMP is given by Algorithm 1.

First of all, XMP requires the switches are ECN-enabled. Currently, most types of ECN-switches support RED, which uses the EWMA algorithm to estimate the average queue length of each outgoing interface and marks packets according to two configurable thresholds [11]. By employing RED with the two configuration tricks, we can implement our packet marking rules. One is to set parameter W_q [11] of EWMA to 1.0. The other is to configure both the high and the low marking threshold to be K.

For brevity and clarity, we denote the variables of subflow r in the form of array elements. During the phase of congestion avoidance, XMP adjusts congestion window *snd_cwnd*[r] and other related parameters once per round. To identify the end of each round, we use variable *beg_seq*[r] to record a specified packet sequence number, as illustrated in Figure 2. When subflow r receives an acknowledgement number that is larger than *beg_seq*[r], its current round ends. And meanwhile, *beg_seq*[r] is updated using the current *snd_nxt*[r] for preparation of identifying the next round.

The destination uses *cnt_ce*[r] to record the amount of received CEs that are waiting for returning to the source. As proposed in BOS, the destination feeds CEs back to the source through encoding the ECE and the CWR codepoint in the TCP header. Since at most 3 CEs can be piggybacked on each acknowledgement packet, the Delayed ACKs mechanism does not matter much to XMP.

XMP reduces *snd_cwnd*[r] only once in a round when receiving CEs on subflow r. To this end, XMP performs the following operations.

- When subflow r receives the first acknowledgement packet that carries CEs[6], it reduces *snd_cwnd*[r]. And meanwhile, it records the current *snd_nxt*[r] in *cwr_seq*[r] and changes the state from $NORMAL$ to $REDUCED$.
- Subflow r ignores all the CEs carried by the succeeding acknowledgements if it is in the state of $REDUCED$.

[6]Namely, the ECE and the CWR codepoint are not both zero.

- Subflow r changes its state from $REDUCED$ to $NORMAL$ when $snd\_una[r] \geq cwr\_seq[r]$.

The idea behind these operations is based on two observations. First, it takes about one RTT for congestion signals to reach the source. Likewise, it also takes time for the reduction in *cwnd* to kick in. Thus, in order to prevent CEs from triggering an undesired successive reduction in *cwnd*, we limit the frequency of reduction to at most once per RTT (round). Second, the amount of CEs received by subflow r in a round is no more than $snd\_cwnd[r]$ which equals the interval between $snd\_una[r]$ and $cwr\_seq[r]$, as illustrated in Figure 2. By the time $snd\_una[r] \geq cwr\_seq[r]$ holds, all the CEs issued in the previous round will have been fed back to the source. At this point, the state is changed from $REDUCED$ to $NORMAL$ for preparation of the possible next reduction in *cwnd*.

Theoretically, the TraSh algorithm involves two levels of convergence. At the inner level, each subflow $r \in R_s$ achieves rate convergence, given a fixed $\delta_{s,r}$. At the outer level, each $\delta_{s,r}$ is tuned using converged subflow rates until traffic shifting stops. The two levels interact with each other. In practice, however, the rate convergence is not a strict condition on which it is allowed to adjust δ. XMP uses a round as the control interval on each subflow so as to quickly response to the changes in the extent of network congestion. Specifically, subflow r updates $instant\_rate[r]$ by dividing $snd\_cwnd[r]$ by $srtt\_us[r]$ and adjusts $delta[r]$ using Equation (9) on the end of each round. $instant\_rate[r]$ serves as the converged rate of subflow r. $srtt\_us[r]$ is the smoothed RTT measured on subflow r. Linux provides a flag, $TCP\_CONG\_RTT\_STAMP$, for congestion control modules since version 2.6.22. By enabling this flag, XMP can measure RTT with microsecond granularity.

XMP provides parameter *mptcp_xmp_reducer* as the user interface to configure β in the BOS algorithm. Since most of hosts in DCNs are connected to switches using links of $1Gbps$ and RTT is commonly less than $400\mu s$ [31, 36], the BDP is about 33 packets. Considering β should not be too large, we think it is a reasonable choice to configure $\beta = 4$ and $K = 10$.

Algorithm 1: The pseudo-code of XMP

At receiving a new *ack* **on subflow** r**:**

```
/* perform per-round-operations                    */
if ack > beg_seq[r] then
    instant_rate[r] ← snd_cwnd[r]/srtt_us[r];
    total_rate ← ∑{instant_rate};
    min_rtt ← min{srtt_us};
    delta[r] ← snd_cwnd[r]/(total_rate × min_rtt);
    if state[r] = NORMAL and
    snd_cwnd[r] > snd_ssthresh[r] then
        /* congestion avoidance                    */
        adder[r] ← adder[r] + delta[r];
        snd_cwnd[r] ← snd_cwnd[r] + ⌊adder[r]⌋;
        adder[r] ← adder[r] − ⌊adder[r]⌋;
    beg_seq[r] ← snd_nxt[r]; // for next round
/* perform per-ack-operations                      */
if state[r] = NORMAL and
snd_cwnd[r] ≤ snd_ssthresh[r] then
    snd_cwnd[r] ← snd_cwnd[r] + 1; // slow start
if state[r] ≠ NORMAL and ack ≥ cwr_seq[r] then
    state[r] ← NORMAL;
```

At receiving ECE or CWR on subflow r**:**

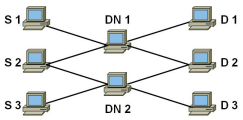
```
if state[r] = NORMAL then
    state[r] ← REDUCED;
    cwr_seq[r] ← snd_nxt[r];
    /* reduce the congestion window                */
    if snd_cwnd[r] > snd_ssthresh[r] then
        tmp ← snd_cwnd[r]/mptcp_xmp_reducer;
        snd_cwnd[r] ← snd_cwnd[r] − max{tmp, 1};
        snd_cwnd[r] ← max{snd_cwnd[r], 2};
    /* avoid re-entering slow start                */
    snd_ssthresh[r] ← snd_cwnd[r] − 1;
```

4. EXPERIMENTS

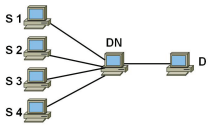
To verify the performance of XMP on traffic shifting and fairness, we built a testbed using several personal computers equipped with multiple Gigabit network interface cards. These computers are all connected to a Gigabit switch. They logically constitute two network topologies shown in Figure 3 by means of different routing configurations. In Figure 3, Si and Di denote the source and the destination, respectively, which are indexed with i and use CentOS 6.4 as operating systems. DNi serves as a switch which runs FreeBSD 9.1 along with DummyNet [26] to construct a two-way bottleneck link and to perform packet marking operations. We configured the sending/receiving buffer of each flow to be sufficiently large so that the transmission rate is limited only by congestion windows. Because the average RTT is roughly $1.8ms$ in our testbed, we set the capacity of bottleneck links to $300Mbps$ so as to achieve a BDP of about 45 packets. DummyNet has a built-in RED algorithm. However, it performs packet discarding to implicitly inform sources of congestion signals. So we had to modify the source code of DummyNet to implement packet marking. In the experiments, we set K to 15 and the queue size to 100 packets. For brevity and clarity, we identify each flow by the number of its corresponding source. Moreover, the subflows of a flow are numbered, starting at 1, in sequence from left to right or from the top down.

The first experiment aims to test the ability of XMP to shift traffic between two paths. In Figure 3(a), Flow 1, Flow 2 and Flow 3 are simultaneously started at $0s$. Flow 2 establishes two subflows which go through $DN1$ and $DN2$, respectively. Two background flows come up on $DN1$ at $10s$ and on $DN2$ at $20s$, respectively, both of which last for $10s$. Figure 4 shows the subflow rates of Flow 2. By comparing Figure 4(a) to 4(b), we found that a larger β may adversely impact the performance of XMP. This is because a larger β makes each flow relinquish less bandwidth for reallocation, leading to an increased convergence time of traffic shifting. If the global synchronization happens before the end of convergence, the process of traffic shifting will stall. Besides, Figure 4(a) demonstrates the rate compensation effect. That is to say, when one subflow decreases the rate, the other subflow will increase its rate as compensation. A more complicated scenario is given in the next section.

Unlike small flows which care more about latency (or completion time), large flows need to fairly share network resources. XMP is specifically designed for assuring fairness of throughput for large flows. To verify it, we conducted the second experiment in the topology of Figure 3(b), where four

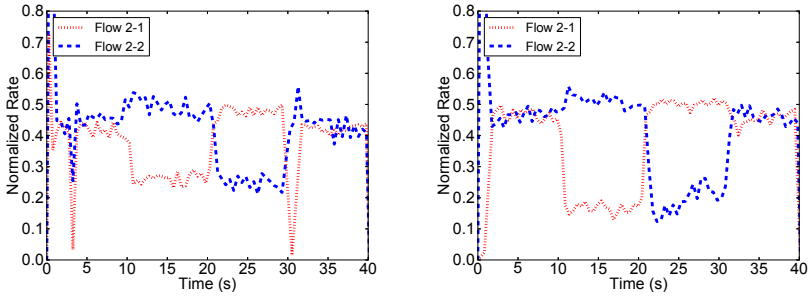


(a) Traffic Shifting



(b) Fairness

Figure 3: Testbed



(a) $\beta = 4$ (b) $\beta = 6$

Figure 4: Two background flows run on $DN1$ from $10s$ to $20s$ and on $DN2$ from $20s$ to $30s$, respectively. They make Flow 2 dynamically shift its traffic from one subflow to the other.

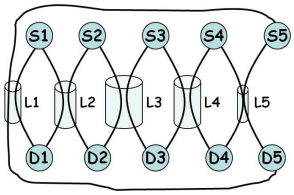
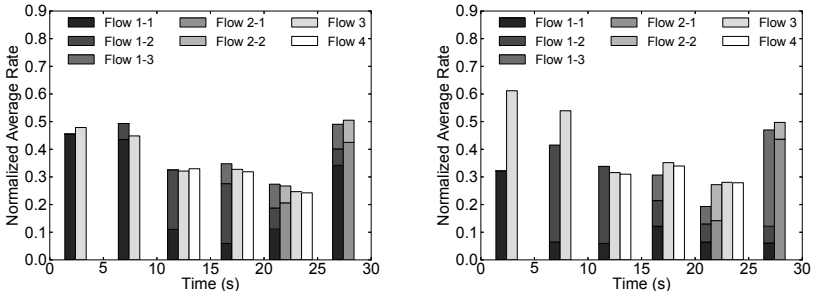


Figure 5: Torus Topology



(a) $\beta = 4$ (b) $\beta = 6$

Figure 6: Four flows compete for one link.

flows compete for one bottleneck link. Flow 1 has three subflows which are established at $0s$, $5s$ and $15s$, respectively. Flow 2 has two subflows which are established simultaneously at $20s$. Flow 3 and Flow 4 both have single subflow and they are started at $0s$ and $10s$, respectively, and are shut down simultaneously at $25s$. The experiment results are shown in Figure 6. Clearly, with $\beta = 4$, all the flows can fairly share the bottleneck link, irrespective of how many subflows each flow has. In contrast, Figure 6(b) shows the fairness of XMP declines when increasing β.

5. SIMULATIONS

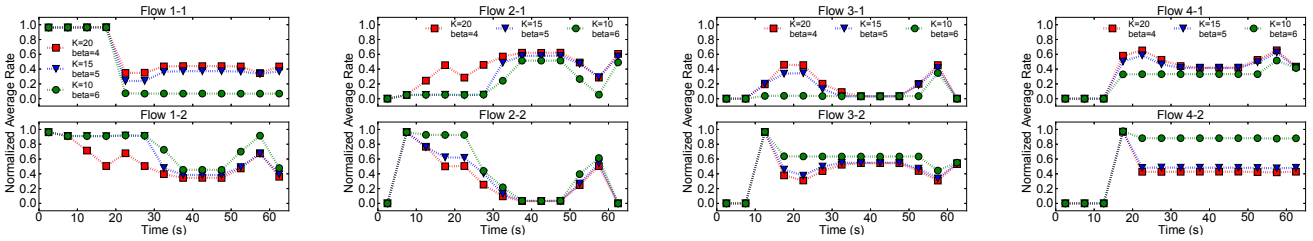
In order to further evaluate the performance of XMP in a larger network topology, we implemented MPTCP and DCTCP in the NS-3.14 simulation platform. XMP and LIA were also implemented as the congestion control algorithm of MPTCP. Note that in all the simulations, we configured the sending/receiving buffer of each flow to be sufficiently large so that the transmission rate is limited only by congestion windows.

5.1 Rate Compensation

The rate compensation effect is an important metric to evaluate the performance of an multipath congestion control algorithm on traffic shifting. Benefiting from the design principle of coupling subflows through the knob parameters, an MPTCP flow can adaptively increase the transmission rate on less congested paths so as to compensate for the decrease in the rate on more congested paths, thus achieving a higher throughput. Rate compensation between subflows leads to an interesting phenomenon: a congestion event occurring in one place may cause the flows in other places to change their rate. In other words, the impact of a congestion event is spread from its occurring location to other places in the network. This process is similar to the attenuated Dominos.

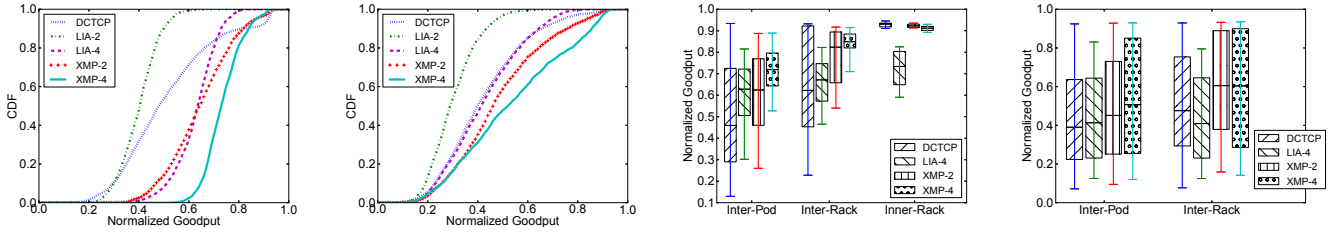
We constructed a scenario to demonstrate the performance of XMP on rate compensation. In Figure 5, from left to right, the capacity of bottleneck links is $0.8Gbps$, $1.2Gbps$, $2Gbps$, $1.5Gbps$ and $0.5Gbps$, respectively. The flows indexed from 1 to 5 are started one by one with an interval of $5s$. Each of these five flows has two subflows that go through the different bottleneck links. After $25s$, the other four background flows are added to link $L3$ one by one with an interval of $5s$. The background flows are used to make $L3$ become increasingly congested. After $45s$, the background flows are shut down sequentially. After $60s$, link $L3$ is closed. We configured RTT of every path to be $350\mu s$. So the BDP is between 15 and 60 packets. We conducted the simulation three times with $\beta = 4$, $\beta = 5$ and $\beta = 6$, respectively. According to Equation (1), we set the corresponding marking threshold to $K = 20$, $K = 15$ and $K = 10$, respectively, and the queue size to 100 packets.

Figure 7 shows the average subflow rates of each flow during every interval of $5s$. As expected, with $L3$ becoming increasingly congested from $25s$ to $45s$, Flow 2-2 and Flow 3-1 both continuously decrease their rate. Meanwhile, Flow 2-1 and Flow 3-2 gradually increase rates to compensate for



(a) Flow 1 (b) Flow 2 (c) Flow 3 (d) Flow 4

Figure 7: Rate Compensation Effect. For each flow, if the rate curve of one subflow is concave, that of the other subflow is convex, and vice versa.



(a) Permutation (b) Incast (c) Permutation (d) Incast

Figure 8: Goodput Distributions. In (c) and (d), each vertical bar is marked with three short horizontal lines that represent the 10th, 50th and 90th percentile, respectively, from the bottom up. Each long vertical line shows the minimal and the maximal value. XMP-x, LIA-y: x and y mean the amount of subflows established by each large flow in the corresponding scheme.

bandwidth loss on their respective sibling subflow, which in turn leads to a decrease in the rate of Flow 1-2 and Flow 4-2, and so on. After $45s$, the changes in the rates of each flow are just the reverse, since $L3$ gradually returns to the initial uncongested status due to the four background flows ending. Finally, when $L3$ is closed at $60s$, the rate of Flow 2-2 and Flow 3-1 sharply declines to zero while the rate of Flow 2-1 and Flow 3-2 shows a significant increase. Also, we found that the rate of Flow 1-1 and Flow 4-2 remains nearly unchanged from $25s$ to $65s$[7]. This indicates that the rate compensation effect is of the attenuation property. In summary, for an MPTCP flow, if the rate curve of one subflow is concave, that of the other subflow is convex, and vice versa. This simulation proves that XMP can effectively achieve rate compensation, or, in other words, traffic shifting.

5.2 Performance Evaluation in DCNs

5.2.1 *Settings and Traffic Patterns*

Considering many DCNs are built using multi-rooted tree architectures [7], we conducted performance evaluation in a Fat-Tree topology which follows the same structure as [2]. Specifically, the network consists of 80 8-port switches and 128 hosts. The link capacity is $1Gbps$. The one-way link delay is $20\mu s$, $30\mu s$ and $40\mu s$ in the rack, the aggregation and the core layer, respectively. So RTT with no queuing delay is between $105\mu s$ and $435\mu s$. Since the BDP is less than 37 packets, we set β to 4, K to 10 and the queue size to 100. Instead of the ECMP routing scheme employed in [25], we adopted the Two-Level Routing Lookup proposed in [2] to forward packets. We assigned multiple addresses to each host so that an MPTCP flow can establish multiple subflows that go through different paths.

According to the simulations in [2, 25] and the findings in [7, 36], we constructed three traffic patterns as follows. Each pattern generates more than 2000 large flows which totally transfer about $600GB$ of data.

- **Permutation**: Every host transfers data to one other host chosen at random with the constraint that each host serves as the destination of only one flow. After all the flows finish data transfer, a new "permutation" is started. This is the simplest randomized pattern that may saturate the whole network [25]. The flow size is uniformly distributed between $64MB$ and $512MB$.

- **Random**: Every host transfers data to one other host chosen at random with the constraint that each host serves as the destination of no more than 4 flows. When a host finishes data transfer, it immediately chooses another host at random to issue a new flow. The flow size is a random variable that follows the Pareto distribution with the shape parameter of 1.5, the mean of $192MB$ and the upper bound of $768MB$.

- **Incast**: A "Job" is defined as follows. First, 9 hosts are chosen at random, one of which serves as a client and the others as servers. Next, the client issues a request to each server simultaneously. Each request is a small flow whose size is a constant, $2KB$. After receiving the request, the server immediately issues back to the client a response which is also a small flow with the size of $64KB$.

[7]The rate of Flow 5-1 and Flow 5-2 also remains unchanged. Their rate curves are not ploted.

Finally, when the client receives all the responses, the Job ends. After that, a new Job will be started. There are totally 8 Jobs running simultaneously. Additionally, each host issues a large flow, following the Random pattern, to generate traffic on background[8]. Note that all the small flows use TCP to transfer data.

5.2.2 Goodput

We define "Goodput" as the average data transfer rate of a large flow over its whole running time. The average goodput over all the large flows is listed in Table 1, where the trailing digit of each scheme name denotes the amount of subflows established by a large flow. The goodput distributions are shown in Figure 8(a) and 8(b).

Table 1: Average Goodput ($Mbps$)

	Permutation	Random	Incast
DCTCP	513.6	440.5	423.7
LIA-2	400.8	310.0	302.7
LIA-4	627.3	434.5	425.4
XMP-2	644.3	497.9	483.7
XMP-4	735.6	542.9	535.7

From these results, we can draw the following conclusions. First, even if each flow establishes 2 subflows, XMP can increase goodput by more than 13%, compared to DCTCP. Second, XMP-2 significantly outperforms LIA-2 and achieves a comparable goodput with LIA-4. We think the primary factor that adversely impacts the goodput of LIA is the minimal retransmission timeout (RTO_{min}) of $200ms$. Third, there is only a minor increase (10%) in goodput for XMP, if the amount of subflows changes from 2 to 4. In contrast, doubling the amount of subflows can significantly increase the goodput of LIA (by more than 40%). In other words, LIA can further improve throughput by establishing more subflows. This result is consistent with [25], where Raiciu *et al.* proposed that MPTCP with LIA needs 8 paths to obtain a good utilization in Fat Tree networks. For XMP, it is not necessary to establish so many subflows. Finally, by comparing Incast with Random, we found that Jobs cause only a slight decrease in the goodput of large flows.

Figure 8(c) and 8(d) provide more information on goodput distributions. According to the location of the source and the destination, we classified large flows into three categories: Inter-Pod, Inter-Rack[9] and Inner-Rack. DCTCP can achieve the highest goodput for inner-rack flows. However, its goodput drops significantly for the other two categories of flows. This is because DCTCP is sensitive to the changes in network congestion. The increase in path hops will lead to a higher packet marking probability, hence more frequent congestion window reduction. In contrast, XMP can compensate for the adverse impact of sensitiveness by means of establishing multiple subflows. Also, we found that LIA achieves a comparable performance with other algorithms for inter-pod flows. The poor performance of LIA for inner-rack flows is mainly caused by the packet loss recovery time of $200ms$, which is two thousand times larger than RTT of inner-rack flows.

[8]We posed on large flows an additional constraint that the source and the destination are not located in the same rack.
[9]The source and the destination are located in the different racks of the same pod.

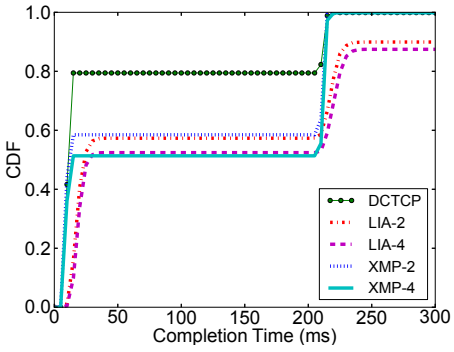


Figure 9: Job Completion Time Distributions. Note that large flows use MPTCP or DCTCP while small flows use TCP to transfer data.

In order to further evaluate the impact of XMP on other congestion control schemes, we constructed three coexistence scenarios using the Random pattern as follows: half of flows use XMP and the other half of flows use 1) LIA or 2) TCP or 3) DCTCP. Each MPTCP flow establishes 2 subflows. The results in Table 2 show that XMP fairly shares network resources with DCTCP. We think the main reasons are two-fold. First, these two schemes are both based on ECN, following the similar congestion control way. Second, the number of subflows are small for XMP. So the advantages of MPTCP are not fully exploited. By referring to the results in Table 1, we believe, with the number of subflows increasing, XMP will obtain more bandwidth than DCTCP.

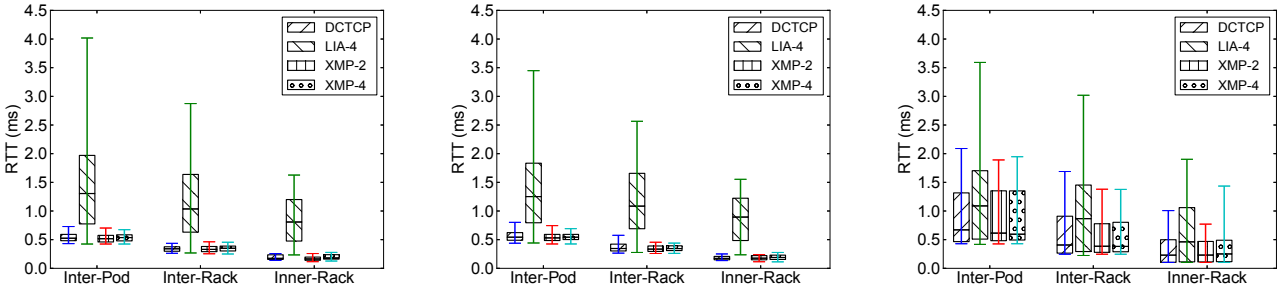
Table 2: Average Goodput ($Mbps$) in the Random Pattern. XMP coexists with other three schemes, respectively.

Queue Size	50 packets	100 packets
XMP : LIA	463.4 : 314.3	423.2 : 388.3
XMP : TCP	522.9 : 175.3	501.8 : 243.4
XMP : DCTCP	485.4 : 485.3	481.4 : 493.5

By comparing Table 2 with column Random in Table 1, we found that XMP has little impact on other schemes, and vice versa. It is worth noting that the queue size is an important factor for the throughput of LIA and TCP. A larger queue size implies that LIA/TCP can use a larger congestion control window to transfer data, which partly offsets link under-utilization caused by packet losses, thus helping LIA/TCP increase throughput. Also, a higher level of link buffer occupancy leads to more ECN signals issued by switches. So XMP has to relinquish some bandwidth to LIA/TCP.

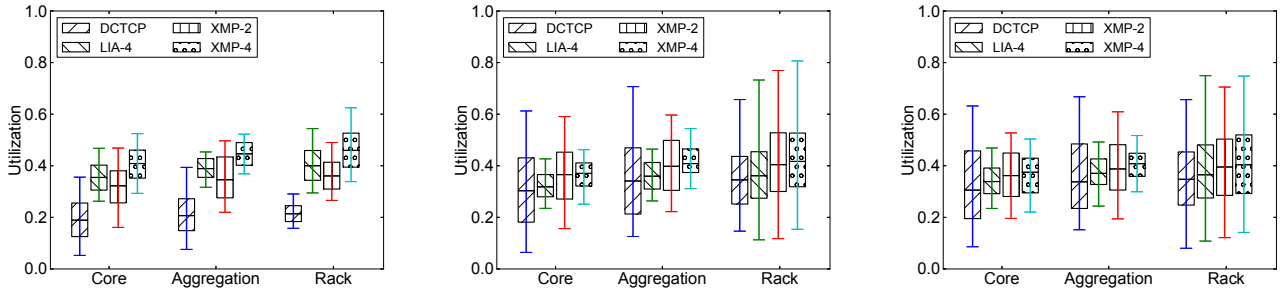
5.2.3 RTT and Job Completion Time

In order to meet the low latency requirement of small flows, the switches should maintain a low level of buffer occupancy in link queues. This is because packet queuing delay predominates RTT in DCNs. For example, in the Fat-Tree network, one buffered packet will increase RTT by $12\mu s$, which accounts for more than one tenth of inner-rack RTT. Thus, RTT can be employed to estimate the level of buffer occupancy in link queues. From Figure 10, we con-



(a) Permutation (b) Random (c) Incast

Figure 10: RTT Distributions



(a) Permutation (b) Random (c) Incast

Figure 11: Link Utilization Distributions. A shorter vertical line implies a more balanced link utilization.

cluded that due to the similar packet marking rules, XMP and DCTCP both can effectively control link buffer occupancy under a low level. Moreover, the amount of subflows has very little impact on RTT. The relatively large RTT in the Incast pattern is because small flows use TCP to transfer data. In contrast, LIA leads to a very large RTT.

The completion time of Jobs is primarily determined by two factors: RTT and packet loss probability. The packet marking rules force large flows to relinquish a majority of link buffer space to small flows, thus reducing packet losses and queuing delay. As shown in Table 3, XMP and DCTCP both make Jobs ending as early as possible. Compared to DCTCP, XMP roughly doubles the completion time of Jobs. This is because MPTCP fully utilizes all the available paths between hosts to transfer data. So small flows have no opportunity to exclusively run on some paths, hence a longer completion time. Note that XMP outperforms LIA on completion time while outperforming both LIA and DCTCP on throughput. Since XMP aims to achieve tradeoff between throughput and latency, the results in Table 3 are acceptable. Also, the results show that more than one tenth of Jobs cannot end within $300ms$ if LIA is used by large flows. In contrast, this ratio is nearly close to zero for XMP as well as DCTCP.

From Figure 9, we can further find some interesting results. First, each CDF curve has two distinct jumps with an interval of about $200ms$. This is due to $RTO_{min} = 200ms$. In other words, most of small flows experience at least one TCP collapse caused by incast. Of course, this result highly depends upon the load level of networks. Second, DCTCP achieves the best performance and it outperforms XMP by roughly 25% before the first jump. As explained before, the reason is that MPTCP saturates the whole network. However, XMP achieves a comparable performance with DCTCP after the second jump. Third, doubling the amount of subflows leads to that roughly 8% of Jobs experience the second TCP collapse. Thus, we think it might not be a good practice to establish too many subflows for an MPTCP flow. Finally, before the first jump in Figure 9, XMP can achieve the completion time of less than $15ms$, which is half of that of LIA.

Table 3: Average Job Completion Time (ms)

	DCTCP	LIA-2	LIA-4	XMP-2	XMP-4
Time	52	156	180	93	109
$> 300ms$	0.1%	10.1%	12.5%	0.1%	0.2%

5.2.4 Link Utilization

We employed link utilization as the metric to evaluate the performance on efficiency. The utilization of link l is defined as $transferred/capacity$, where the numerator is the amount of bytes that go through link l during the whole simulation time and the denominator is the maximal amount of bytes that can be transferred by link l.

Figure 11(a) shows that the utilization of DCTCP is considerably below that of the other schemes. This is because a DCTCP flow can use only one path between the source and the destination, thus wasting many under-utilized links, es-

pecially when several flows collide on one link. Figure 11(b) and 11(c) show the similar distributions. This indicates that link utilization is primarily determined by the traffic of large flows. Compared to LIA, XMP increases link utilization by 10% in average. Although the peak utilization of DCTCP in the core and the aggregation layer is higher than that of XMP and LIA, the vertical lines of DCTCP are longer in Figure 11. In other words, DCTCP fails to achieve a balanced link utilization.

Note that, irrespective of which transport protocol is used, the total data in each simulation is about $600GB$ transferred by 2000 large flows. Because the throughput of DCTCP is smaller than that of MPTCP, the same data transferred with DCTCP take more time than the one with MPTCP.

6. RELATED WORK

Many multipath transport protocols were proposed in earlier years, such as pTCP [14], mTCP [37] and CMT-SCTP [16]. By incorporating many lessons learned from previous research efforts, Raiciu *et al.* proposed MPTCP [1] in recent years and demonstrated its pretty good performance in DCNs [25]. MPTCP has become a de facto standard of multipath proposals. However, the above schemes are designed originally for the Internet, rather than for DCNs.

The work in [7] analyzed the raw data collected from 10 DCNs and obtained some empirical results which are very useful for researchers to emulate DCNs. [7] found that many DCNs are built using 3-layer multi-rooted tree architectures [13]. Thus, we conducted the simulations in a Fat Tree topology [2]. Vasudevan *et al.* [31] studied the Incast issue in DCNs and proposed to reduce RTO_{min} for avoiding TCP collapse. We think this method may also help MPTCP improve its throughput. ICTCP [35] is a receiver-side scheme that eliminates Incast congestion by adjusting TCP receive window proactively before packet drops occur. Alizadeh *et al.* recognized the conflicting requirements between large and small flows in DCNs and proposed two solutions to address it. One is to sacrifice about 10% of bandwidth for reducing packet queuing delay down to zero [5]. The other is DCTCP [4], which employs ECN to limit the buffer occupancy of large flows. Similarly, D2TCP [30] uses ECN to make flows with tight deadlines obtain more bandwidth in DCNs. Besides, the work in [36] proposed to improve the data transfer performance of DCNs only by tuning ECN parameters. Note that the above transport protocols are all single-path schemes. ECN is also employed by NF-TCP [6] to make large flows such as P2P traffic submissive to latency-sensitive applications under congestion. However, it is also not a multipath scheme, and not designed for DCNs.

MPTCP can be regarded as a load balancing scheme deployed at end systems. This is because it can shift traffic from one path to the others. [19] improved this ability. Thus, when large flows use MPTCP to transfer data in DCNs, link utilization will be more balanced. Before that, flow scheduling is one of the possible solutions, such as Hedera [3].

The fluid approximation and the duality model [22] are both classic methods to solve congestion control issues. For example, [23] analyzed the linear stability of TCP/RED and Kelly *et al.* [18] presented a sufficient condition for the local stability of end-to-end congestion control. Low [21, 22] proposed a dual model to formulate TCP congestion control algorithms. Wang *et al.* [32] developed two distributed algorithms to maximize the aggregate source utility. We partially exploited these theoretical foundations on congestion control to design XMP.

7. CONCLUSIONS AND FUTURE WORK

In order to balance throughput with latency in DCNs, we developed XMP as a congestion control scheme of MPTCP. XMP primarily comprises two core algorithms. The BOS algorithm achieves controllable link buffer occupancy to meet the low latency requirement of small flows. The TraSh algorithm performs traffic shifting, thus improving the throughput of large flows. Our experiments and simulations demonstrate the performance of XMP in DCNs.

XMP has two configurable parameters: packet marking threshold K and congestion window reduction factor β. The parameter setting has impact on the performance of XMP, as demonstrated by our experiments. We think a deeper understanding on these impacts should be based on further theoretical analysis. Additionally, we noted the points raised by [19]. As the principle of TraSh is similar to LIA, TraSh may also be subject to the non-Pareto-optimal issue, and the proposed solution in [19] could be exploited to improve XMP as well. However, the focus of XMP is on the trade-off between throughput and latency. We plan to study the above issues in future work. Also, we will deploy XMP in a DCN testbed to verify and possibly improve its performance.

8. ACKNOWLEDGMENTS

The research is partially supported by the National Basic Research Program of China (973 Program) under Grant 2012CB315803, the National Natural Science Foundation of China (61073166 and 61133015), the National High-Tech Research and Development Program of China (863 Program) under Grant 2011AA01A101, as well as the EU-JAPAN GreenICN project under EC FP7 Grant No. 608518 and NICT Contract No. 167.

9. REFERENCES

[1] A. Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar. Architectural Guidelines for Multipath TCP Development. RFC 6182, Mar. 2011.

[2] M. Al-Fares, A. Loukissas, and A. Vahdat. A Scalable, Commodity Data Center Network Architecture. In *ACM SIGCOMM*, pages 63–74, 2008.

[3] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic Flow Scheduling for Data Center Networks. In *USENIX NSDI*, pages 19–19, 2010.

[4] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data Center TCP (DCTCP). In *ACM SIGCOMM*, pages 63–74, 2010.

[5] 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 *USENIX NSDI*, pages 19–19, 2012.

[6] M. Arumaithurai, X. Fu, and K. K. Ramakrishnan. NF-TCP: a Network Friendly TCP Variant for Background Delay-insensitive Applications. In *NETWORKING*, pages 342–355, 2011.

[7] T. Benson, A. Akella, and D. A. Maltz. Network Traffic Characteristics of Data Centers in the Wild. In *ACM IMC*, pages 267–280, 2010.

[8] Y. Cao, M. Xu, and X. Fu. Delay-based Congestion Control for Multipath TCP. In *IEEE ICNP*, pages 1–10, 2012.

[9] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *USENIX OSDI*, pages 137–150, 2004.

[10] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's Highly Available Key-Value Store. *ACM SIGOPS Operating Systems Review*, 41(6):205–220, Oct. 2007.

[11] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, Aug. 1993.

[12] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google File System. *ACM SIGOPS Operating Systems Review*, 37(5):29–43, Oct. 2003.

[13] 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 *ACM SIGCOMM*, pages 51–62, 2009.

[14] H. Y. Hsieh and R. Sivakumar. pTCP: an end-to-end transport layer protocol for striped connections. In *IEEE ICNP*, pages 24–33, 2002.

[15] M. Isard. Autopilot: Automatic Data Center Management. *ACM SIGOPS Operating Systems Review*, 41(2):60–67, Apr. 2007.

[16] J. R. Iyengar, P. D. Amer, and R. Stewart. Concurrent multipath transfer using SCTP multihoming over independent end-to-end paths. *IEEE/ACM Transactions on Networking*, 14(5):951–964, Oct. 2006.

[17] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168, Sept. 2001.

[18] F. Kelly and T. Voice. Stability of end-to-end algorithms for joint routing and rate control. *ACM SIGCOMM Computer Communication Review*, 35(2):5–12, Apr. 2005.

[19] R. Khalili, N. Gast, M. Popovic, U. Upadhyay, and J.-Y. Le Boudec. MPTCP is not Pareto-Optimal: Performance Issues and a Possible Solution. In *ACM CoNEXT*, pages 1–12, 2012.

[20] A. Kuzmanovic. The Power of Explicit Congestion Notification. In *ACM SIGCOMM*, pages 61–72, 2005.

[21] S. H. Low. A Duality Model of TCP and Queue Management Algorithms. *IEEE/ACM Transactions on Networking*, 11(4):525–536, Aug. 2003.

[22] S. H. Low and D. E. Lapsley. Optimization flow control-I: basic algorithm and convergence. *IEEE/ACM Transactions on Networking*, 7(6):861–874, Dec. 1999.

[23] S. H. Low, F. Paganini, J. Wang, and J. C. Doyle. Linear Stability of TCP/RED and a Scalable Control. *Computer Networks*, 43(5):633–647, Dec. 2003.

[24] A. Phanishayee, E. Krevat, V. Vasudevan, D. G. Andersen, G. R. Ganger, G. A. Gibson, and S. Seshan. Measurement and Analysis of TCP Throughput Collapse in Cluster-based Storage Systems. In *USENIX FAST*, pages 12:1–12:14, 2008.

[25] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley. Improving datacenter performance and robustness with multipath TCP. In *ACM SIGCOMM*, pages 266–277, 2011.

[26] The dummynet project. [Online]. Available: http://info.iet.unipi.it/∼luigi/dummynet/.

[27] The implementation of MPTCP. [Online]. Available: http://multipath-tcp.org/pmwiki.php?n=Main.Release86.

[28] The implementation of XMP. [Online]. Available: http://routing.netlab.edu.cn/tiki-index.php?page=Yu+Cao.

[29] The NS-3 network simulator. [Online]. Available: http://www.nsnam.org/.

[30] B. Vamanan, J. Hasan, and T. Vijaykumar. Deadline-Aware Datacenter TCP (D2TCP). In *ACM SIGCOMM*, pages 115–126, 2012.

[31] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G. R. Ganger, G. A. Gibson, and B. Mueller. Safe and Effective Fine-grained TCP Retransmissions for Datacenter Communication. In *ACM SIGCOMM*, pages 303–314, 2009.

[32] W. H. Wang, M. Palaniswami, and S. H. Low. Optimal flow control and routing in multi-path networks. *Performance Evaluation*, 52(2–3):119–132, Apr. 2003.

[33] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron. Better Never than Late: Meeting Deadlines in Datacenter Networks. In *ACM SIGCOMM*, pages 50–61, 2011.

[34] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley. Design, implementation and evaluation of congestion control for multipath TCP. In *USENIX NSDI*, pages 8–8, 2011.

[35] H. Wu, Z. Feng, C. Guo, and Y. Zhang. ICTCP: Incast Congestion Control for TCP in Data Center Networks. In *ACM CoNEXT*, pages 13:1–13:12, 2010.

[36] H. Wu, J. Ju, G. Lu, C. Guo, Y. Xiong, and Y. Zhang. Tuning ECN for Data Center Networks. In *ACM CoNEXT*, pages 25–36, 2012.

[37] M. Zhang, J. Lai, A. Krishnamurthy, L. Peterson, and R. Wang. A transport layer approach for improving end-to-end performance and robustness using redundant paths. In *USENIX Annual Technical Conference*, pages 99–112, 2004.