OFPT: OpenFlow based Parallel Transport in Datacenters

Bo Liu[1], Bo XU[1], Chao Hu[1,2], Hui Hu[1] and Ming Chen[1]
[1]College of Command Information Systems, PLA University of Science and Technology, Nanjing, China
[2]Key Laboratory of Computer Network and Information Integration (Southeast University), Ministry of Education, Nanjing, China
[e-mail: lbo.xidian@163.com]
*Corresponding author: Bo XU

Received July 4, 2015; revised May 6, 2016; accepted September 3, 2016; published October 31, 2016

Abstract

Although the dense interconnection datacenter networks (DCNs) (e.g. FatTree) provide multiple paths and high bisection bandwidth for each server pair, the single-path TCP (SPT) and ECMP which are widely used currently neither achieve high bandwidth utilization nor have good load balancing. Due to only one available transmission path, SPT cannot make full use of all available bandwidth, while ECMP's random hashing results in many collisions. In this paper, we present OFPT, an OpenFlow based Parallel Transport framework, which integrates precise routing and scheduling for better load balancing and higher network throughput. By adopting OpenFlow based centralized control mechanism, OFPT computes the optimal path and bandwidth provision for each flow according to the global network view. To guarantee high throughput, OFPT dynamically schedules flows with Seamless Flow Migration Mechanism (SFMM), which can avoid packet loss in flow rerouting. Finally, we test OFPT on Mininet and implement it in a real testbed. The experimental results show that the average network throughput in OFPT is up to 97.5% of bisection bandwidth, which is higher than ECMP by 36%. Besides, OFPT decreases the average flow completion time (AFCT) and achieves better scalability.

Keywords: Datacenter; Traffic Engineering; OpenFlow; Multipath; Flow completion time

This work is supported by the State Key Development Program for Basic Research of China under Grant No. 2012CB315806, the National Natural Science Foundation of China under Grant Nos. 61103225 and 61379149, Jiangsu Province Natural Science Foundation of China under Grant No. BK20140070, Jiangsu Future Networks Innovation Institute Prospective Research Project on Future Networks under Grant No. BY2013095-1-06.

http://dx.doi.org/10.3837/tiis.2016.10.009 **ISSN : 1976-7277**

1. Introduction

In the past decade, the Internet applications, computing and data storage are migrating to datacenters. Because mass data are stored in distributed circumstance, the data migration and some bandwidth-intensive applications (e.g. MapReduce) have triggered significant increase of inter-datacenter traffic. Some novel frameworks (e.g. FatTree [1]) have been presented to improve the connectivity and robustness of datacenter network, which engender multiple parallel transmission paths and high bisection bandwidth for each server pair. However, traditional TCP/IP architecture applies SPT, it makes some links congested while other links' loads are very light, and the average bandwidth utilization is only 20%~30%. Therefore, how to improve the load balancing as well as the throughput of DCNs is an urgent problem in DCNs.

Researchers have proposed many multipath transport schemes in DCNs to improve the load balancing, including flow-level schemes [1], [2], [3], packet-level schemes [4], [5], [6], [7] and flowlet (or flowcell)-level schemes [8], [9], [10]. Besides, Hedera [11] and MicroTE [12] have proposed centralized control mechanism in DCNs for improving the network throughput, and they proposed using OpenFlow [13] to improve the load balancing. By separating the control plane from data plane, OpenFlow enhances the instant management and provides fine-grained control on network resources distribution owing to the advantages of centralized control mechanism. Google firstly employs OpenFlow to reform its wide area network B4 [14] in 2010. After the reconstruction, the bandwidth utilization of B4 has increased from 20%~30% to nearly 100%, and B4 has realized load balancing and differentiated services. Therefore, applying OpenFlow and multipath transport mechanism to solve the DCN traffic engineering (TE) issue is an appropriate choice.

This paper targets on addressing the issues of DCNs via OpenFlow. There is serious packet disordering issue in packet-level schemes, which will seriously impair the user experience as well as the network throughput. Meanwhile, although flowlet-level scheme can efficiently improve datacenter's throughput, it will add more overheads on OpenFlow controller as well as more flow tables on OpenFlow switches, which will impair the scalability of DCNs. Therefore, we believe flow-level scheme is more suitable in OpenFlow network. Unfortunately, the performance of existing OpenFlow based flow-level schemes is still far from the optimal for short of the consideration on current load of links or precise routing and scheduling. For example, Hedera and MicroTE adopt ECMP routing for short flows, which will hurt the performance of short flows. In this paper, we propose OpenFlow based Parallel Transport (OFPT) framework, a flow-level multipath transmission scheme to improve load balancing and network throughput. Referring to the idea of parallel computing, we regard the end-to-end traffic as a task, while the available bandwidth of transmission links is abstracted as the resource pool. The end servers just inject the traffic into the network, and the OpenFlow controller conducts the optimal allocation of network resource. More specifically, by maintaining the global network view, including the available bandwidth of each link, the sending rate and routing information of each flow, OFPT computes the optimal path and bandwidth provision for each flow. The controller updates the network view periodically and schedules flows to lighter paths with SFMM for better load balancing. The main contributions of this paper are as follows:

1) We design OFPT framework, which has the following advantages: a) OFPT is a flow-level scheme, which can avoid packet disordering problem; b) OFPT adopts an accurate

routing algorithm, which can avoid flow collision that ECMP suffers; c) OFPT complies with the OpenFlow standards and does not need any modifications on switches, thus is easy to deploy; d) OFPT decreases FCT and increases the throughput as well.

2) We present the mathematical model of OFPT and the corresponding algorithms of routing and scheduling, which makes DCN achieving better load balancing and higher throughput.

3) We design the seamless flow migration mechanism, which can keep away from packet loss during flow rerouting.

The remainder of this paper is organized as follows. In Section 2, we discuss the related work about TE issues in DCNs. We describe the motivation in Section 3 and the framework of OFPT in Section 4, and present the mathematical optimization model and the corresponding algorithms of OFPT in Section 5. In Section 6, we introduce our experimental methodology and results. Finally, our work is concluded in Section 7.

2. Related Works

DCNs have three main goals: increasing the bisection bandwidth for high network throughput, achieving good load balancing for the increase of bandwidth utilization, and decreasing the average flow completion time for improving user experience. To realize these goals, some novel structures of datacenter network are presented to increase the bisection bandwidth, and multipath transport schemes are employed for load balancing in the dense interconnection DCNs. Moreover, OpenFlow based software-defined networks (SDN) is introduced to the innovation of DCNs.

Traditional DCNs generally adopt a hierarchical tree structure, which is composed of edge layer, aggregation layer and core layer. Racks of servers directly connect to top-of-rack switches (ToR switches) at the edge layer, and ToR switches connect to the aggregation switches (Agg switches) through a certain structure for bandwidth aggregation. Finally, aggregation switches connect to the core layer switches (Spine switches). This type of DCNs' structure has many shortcomings. For example, due to the low connectivity, datacenter networks have a limited bisection bandwidth, and the robustness and scalability are far away from practical requirements. Some researchers try to design new DCN fabric to increase the DCN's bisection bandwidth. These new datacenter fabrics can be divided into two classes: one is network-centric solution, in which the switches are only responsible for packet forwarding, such as FatTree and VL2; and the other is server-centric solution, such as BCube [15], in which the switches forward the packet and store some data as servers. The new DCNs have high connectivity among servers, so they efficiently increase the bisection bandwidth and improve the robustness. **Fig. 1** shows the k=4 FatTree DCN topology. We can find that there are four parallel transmission paths between any two servers in different pods.

Although there are multiple parallel paths between each server pair in dense interconnection datacenter networks, SPT only uses one path in data transmission and thus has a low throughput. Multipath transmission schemes have been proposed to solve this problem. ECMP [1] adopts random load balancing mechanism, in which each router chooses the next hop for each packet by hashing the five-tuple, including source/destination IP address, source/destination port number and the protocol of the transport layer. SPAIN [2] splits the multiple paths into different VLANs, and an offline network controller system pre-computes and pre-installs the transmission path for each flow. VL2 [3] adopt Valiant Load Balancing to spread traffic uniformly across network paths. However, none of them can fully utilize the bisection bandwidth for lack of precise routing and they are unaware of the current network

load. In DRB [4], for each outgoing packet, the source server selects one of the highest level switch (Spine switch) based on the measured latency introduced by the end-server network stack to be bouncing switch, and sends the packet to that switch. The bouncing switch then bounces the packet to the destination. FlowBender [5] is based on ECMP and uses end-host-driven rehashing to trigger dynamic flow-to-path assignment when detecting congestion. RPS [6] and DeTail [7] place the burden of load balancing the traffic upon the switches themselves at the packet level, which requires hardware changes at the switches and thus makes it hard to be deployed. Resorting to the multiple addresses servers, MPTCP [8] separates one (long) flow into multiple subflows and then delivers each subflow in different paths, and all subflows use a coupling congestion control algorithm. Presto [9] utilizes edge vSwitches to break each flow into flowcells, and distributes them evenly to near-optimally load balance the network. CONGA [10] splits TCP flows into flowlets, estimates real-time congestion on fabric paths, and allocates flowlets to paths with small load based on feedback from remote switches.

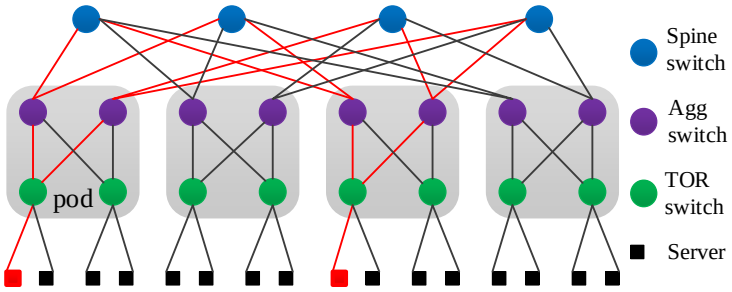


Fig. 1. *k*=4 FatTree datacenter network topology

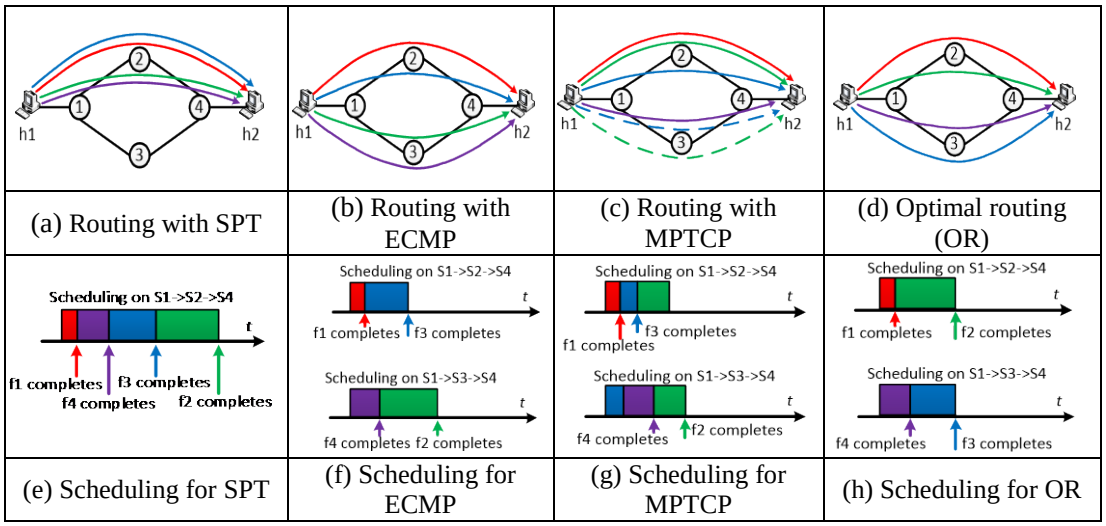
The advent of OpenFlow based SDN fabric provides a new way to solve the datacenter issues, such as TE, flexibility, robustness and rapid configuration. Compared with TCP/IP architecture, SDN adopts a centralized control mechanism, in which the logical control functions and the high-level network policy are accomplished by the controller, and the controller maintains the flow table structure on OpenFlow switches. In the OpenFlow network, packets are forwarded according to the flow tables, while the generation, maintenance and configuration of flow tables are implemented by the central controller. The improvement of Google's datacenter B4 via OpenFlow demonstrates that it is feasible to achieve better traffic scheduling through centralized control mechanism, and Hedera [11] and MicroTE [12] have also shown the OpenFlow based global resource allocation scheme can efficiently increase the network throughput.

3. Motivation

In this section, we introduce some key observations in achieving high throughput in high connectivity network topology. As shown in **Fig. 2**, each link's bandwidth is 100Mbps. There are four flows transmitting from h1 to h2, and the size of flows is shown in **Table 1.** We use four different routing schemes in the example, which are SPT, ECMP, MPTCP and the optimal routing scheme, respectively. The routing is shown in **2(a)~2(d)**, and correspondingly optimal scheduling (short job first, SJF) are shown in **2(e)~2(h)**. In MPTCP, we just divide the flows which size is more than 50Mb into two subflows due to only two paths between the server pair in our example.

Table 1. flows' size used in example

Flow number	f1	f2	f3	f4
Size (Mb)	20	80	60	40



(a) Routing with SPT | (b) Routing with ECMP | (c) Routing with MPTCP | (d) Optimal routing (OR)

(e) Scheduling for SPT | (f) Scheduling for ECMP | (g) Scheduling for MPTCP | (h) Scheduling for OR

Fig. 2. An explanatory example, where (a)~(d) indicate different routing schemes, and (e)~(h) indicate the optimal scheduling schemes for (a)~(d).

The first observation is that multipath transport scheme is more efficient than SPT in improving throughput and decreasing flow completion time. We can learn from **Fig. 2(e)~Fig. 2(h)** that SPT has the lowest throughput in four schemes, and especially its throughput is only half of the optimal, while ECMP and MPTCP achieve 83.3% and 90.9% of the optimal, respectively. What's more, the FCT of f2 is 2s in SPT while 1s in OR, and the FCT is 1.2s in both ECMP and MPTCP, so we can conclude that multipath transport scheme can efficiently decrease FCT.

The second observation is that global optimization is more efficient than distributed control. We can learn from **Fig. 2(f)** and **Fig. 2(h)** that ECMP gets only 83.3% of the optimal scheme in throughput. Moreover, the throughput in ECMP only achieves 50% of the optimal when the four flows route on S1→S2→S4 (or S1→S3→S4) at the same time due to random hashing.

The third observation is that both routing and scheduling are demanded and they should be integrated in improving throughput and decreasing FCT. We can learn that routing is essential for improving the network throughput and load balancing from **Fig. 2(f)** to **Fig. 2(h)**. If we schedule long flow first (LFF), the FCT of f1 and f2 in **Fig. 2(h)** will be 1.0s and 0.8s respectively, but they are 0.2s and 1.0s in the optimal scheduling, which demonstrates the FCT of f1 will increase by 400% in LFF. Therefore, scheduling is also essential in DCNs. MPTCP achieves better throughput and load balancing than ECMP, but it also impairs some short flow's FCT. For example, f4 should be scheduled before f3 based on SJF, but it is scheduled later than f3 in MPTCP, which results in the increase of f4's FCT (the FCT of f4 in ECMP is 0.4s while is 0.7s in MPTCP).

In conclusion, we should integrate centralized control mechanism, multipath transport, precise routing and scheduling in improving the throughput of DCNs. In our example, the highest throughput and the minimal FCT can be achieved by combining the optimal multiple transport scheme, routing and scheduling in **Fig. 2(d)** and **Fig. 2(h)**. We design OFPT framework to accomplish this target.

4. Design Overview

OFPT aims at optimally decomposing the traffic on all available paths to improve the network throughput. Specifically, for each flow OFPT determines which path carries it, when to start it and how much bandwidth should be allocated for it. Therefore, OFPT needs to perceive the load and available bandwidth of each link, and it also requires more detail of each flow, such as size and sending rate. OFPT assumes that the information about each flow can be derived from the upper layer applications or using the state-of-the-art prediction techniques, and it mandates that flows less than 100KB are considered as short flows. The key ideas of OFPT are:

To achieve scalability, OFPT mainly orchestrates long flows, while short flows are treated as background traffic. To decrease short flows completion time (SFCT), background traffic is transmitted along the preinstalled flow tables installed by controller, which can overleap the extra RTTs for installing flow tables. OFPT installs flow tables for short flows based on server-lever load balancing.

We describe the OFPT framework in algorithm 1, which is invoked whenever a new flow comes, an existing flow finishes or the pooling time is up. In more details, when a new flow comes, OFPT is triggered to compute the routing and bandwidth provision for it. When an existing flow finishes and the bandwidth is released, OFPT is also triggered to determine which flow uses the available bandwidth. To decrease AFCT, OFPT uses the well-known minimum remaining time first (MRTF) as the scheduling policy. Furthermore, OFPT carries out a periodic polling operation to improve the load balancing.

Algorithm 1: The OFPT Framework

Input:

$F_c \leftarrow \{f_{auc}\}$/* all uncompleted flows*/

$R_c \leftarrow \{(p_{fi}, b_{fi})\}$, $f_i \in F_c$/* current routing*/

Output:

F_u /* updated flows set*/

R_u /* updated routing*/

1: **while** $F_c \neq \emptyset$ **do**

2: R_u=Maximize_Throughput(F_c, R_c) /*compute the maximum throughput for each flow, and the routing and bandwidth provision */

3: Scheduling(R_u)

4: **end while**

5: **return** F_u, R_u

In algorithm 1, all the (long) flows which are uncompleted are put into the flow set F_c as the optimization object. We regard the F_c and the current routing set R_c as the input, and OFPT will compute the transmission path, bandwidth provision and sending order for each flow in F_c. There are two key issues that OFPT has to address, one is to allocate the maximum bandwidth provision for each flow, and the other is to realize transparent flow rerouting for end server.

5. Details of OFPT

In this section, we will present the key algorithms of OFPT. Firstly, we present the mathematical model of OFPT in 5.1, and then give the bandwidth distribution algorithm. In 5.2, we present the scheduling and incremental update algorithms, and we design the Seamless Flow Migration Mechanism in 5.3.

5.1 Maximize the network throughput

We use directed graph $G = (V, E)$ to describe the network model. We assume that the network is composed of a set of nodes V, and they connect to each other with a set of directed links E. Moreover, $f(e)$ and $c(e)$ represent the current traffic and the link capacity of link e ($e \in E$), respectively, and $NH(u, d)$ denotes the set of next hop nodes from u to d.

Given two servers s and d, all path $<s,u_0,\ldots u_k, d>$ from the source server s to the destination server d will be termed as ***reachable path*** if $(u_{j-1},u_j) \in E$ and $\{u_j\}=NH(u_{j-1}, d)$ for $j=1,2, \ldots,k$, and the set of reachable paths between s and d is denoted by P_{sd}. Besides, let W_{sd} represent the bisection bandwidth between s and d.

Optimization target 1: Optimal Load Balancing

The first target of OFPT is to improve the load balancing, which is to decrease the bandwidth utilization θ of a given link, i.e., we prefer to put more traffic on the lightest link first. We describe this purpose of traffic decomposition as the following optimization problem.

$$minimize \quad \theta$$

Subject to

$$\begin{cases} \sum_{e \in p} x_{sd}(p) + f(e) \le \theta c(e) & \forall e \in E \quad p \in P_{sd} \quad (1) \\ \sum_{p \in P_{sd}} x_{sd}(p) \le W_{sd} & \forall i \in V \quad d \in V \quad (2) \\ x_{sd}(p) \ge 0 \quad \forall p & (3) \end{cases}$$

where $x_{sd}(p)$ denotes the traffic constituent on the path p from s to d.

The first constraint condition demonstrates that the sum of the existing traffic $f(e)$ and all new arriving traffic portion $x_{sd}(p)$ on link e must be less than the product of the maximum link utilization θ and the capacity of the link $c(e)$. The second constraint condition means that the total arriving traffic cannot exceed the bisection bandwidth W_{sd}, and the third constraint condition is to ensure the non-negative characteristic.
Where $x_{sd}(p)$ denotes the traffic constituent on the path p from s to d.

Optimization target 2: Maximize Throughput

The second target of OFPT is to improve the network throughput, i.e., improving the utilization λ of a given bisection bandwidth. We introduce the following optimization problem to describe the goal of traffic decomposition.

$$maximize \quad \lambda$$

Subject to

$$\begin{cases} \sum_{e \in p} x_{sd}(p) \le c(e) - f(e) = b(e) & \forall e \in E \quad p \in P_{sd} \quad (4) \\ \sum_{p \in P_{sd}} x_{sd}(p) \le \lambda W_{sd} & \forall s \in V \quad d \in V \quad (5) \\ x_{sd}(p) \ge 0 \quad \forall p & (6) \end{cases}$$

where $x_{sd}(p)$ denotes the traffic constituent on the path p from s to d, and b(e) represents the available bandwidth of link e.

The first constraint condition indicates that the sum of the existing traffic $f(e)$ and all new arriving traffic $x_{sd}(p)$ on link e must be less than the capacity of the link $c(e)$. The second constraint condition means that the total arriving traffic cannot exceed the product of the maximum link utilization λ and the bisection bandwidth W_{sd}, and the third constraint condition ensures that the flow on any path is non-negative.

OFPT aims to minimize the bandwidth utilization of each link in the network for the given traffic, which is equivalent to keeping the capacities of the link fixed but scale the injected traffic. Therefore, the optimization target 1 is in accordance with the optimization target 2 in nature, and we choose optimization model 2 as our optimization model of traffic decomposition. In this model, we assumed that values $f(e)$ and W_{sd} are known, but OFPT needs to update their values based on instant workload. Fortunately, both $f(e)$ and W_{sd} are easy to calculate because OFPT can be easy to get each flow's information in OpenFlow network. Although the problem is a NP-complete problem and has an exponential number of variables, we can solve the problem with a primal-dual algorithm.

We introduce dual variables $l(e)$ for constraint (4) and z_{sd} for constraints (5), where $l(e)$ is the cost of link and z_{sd} is the path with the lightest load from s to d. The dual can be written as

$$minimize \sum_{e\in E} b(e)l(e)$$

Subject to

$$\begin{cases} \sum_{e\in p} l(e) \geq z_{sd} \qquad \forall e \in E \quad p \in P_{sd} & (7) \\ \sum_{s\in V}\sum_{d\in V} z_{sd} W_{sd} \geq 1 & (8) \\ l(e) \geq 0 \quad \forall e \in E & (9) \end{cases}$$

Let L_{sd} denote the lightest path from s to d which use the link cost $l(e)$ of e. The dual can now be rewritten as

$$minimize \sum_{e\in E} b(e)l(e)$$

Subject to

$$\begin{cases} \sum_{s\in V}\sum_{d\in V} W_{sd} L_{sd} \geq 1 & (10) \\ l(e) \geq 0 \quad \forall e \in E & (11) \end{cases}$$

In other words, given any non-negative set of link costs $l(e)$, $\sum_{e\in E} b(e)l(e) / \sum_{s\in V}\sum_{d\in V} W_{sd} L_{sd}$ is the upper bound of the dynamic routing problem. We use Fully Polynomial Time Approximation Scheme (FPTAS) to solve this problem, which is simple to implement and runs significantly faster than a general linear programming solver in OFPT.

FPTAS provides the following performance guarantees: for any $\varepsilon > 0$, the solution has objective function value within $(1+\varepsilon)$ – factor of the optimal, and the running time is at most a polynomial function of the network size and $1/\varepsilon$. FPTAS in our case is a primal dual algorithm, and we implement some optimization on FPTAS to decrease the running times via dynamic graph algorithms [16]. We optimize the primal dual algorithm with the following approaches.

1) In the FatTree network, given an arbitrary feasible traffic matrix, if a routing algorithm can evenly spread the traffic x_{ij} from server i to server j among all the possible uplinks at every

layer, then all the links, including all the downlinks, are not overloaded, which has been verified in [4]. Moreover, for FatTree network, once the spine switch is chosen, the routing path from the source to the destination is decided without ambiguity. Therefore, we just need to precisely compute the routing from the source server to the spine switch, thus it decreases the running time of the algorithm. For VL2, we can also generalize the similar conclusion according to the related algorithm presented in [4].

2) OFPT maintains path information in the controller, which can provide all path information for a flow directly rather than compute everything from scratch. For any $s \to d$, the path information is expressed as $P_{sd} \leftarrow \{B_a, path_1 : b, path_2 : b\}$, where B_a represents the available bandwidth of the heaviest load path in P_{sd}, while b is the available bandwidth of a given path. By using path information, we only compute the multipath only once for a given ToR switches pair.

The primal dual algorithm for our problem works as follows: The algorithm first computes a value δ that is the function of the desired accuracy level ε and the number of switch n, then get all uncompleted flows F_{sd} sorted according to MRTF in ascend order and computes all available paths P_{sd} for a given $s \to d$. In each primal phase $\sum w_f$ units of flows are routed along a path p until $\Delta b < 0$, where $\Delta b = b_p - B_a$ and B_a is the available bandwidth of the heaviest path in P_{sd}. This process of augmenting flow and updating the dual length is repeated until the problem is dual feasible. More details of the algorithm are given in algorithm 2.

Algorithm 2: Maximize_Throughput (F_c, R_c)

Input: F_c, R_c

Output: λ, F_c, R_c //updated flow and routing information

$\delta = (1+\varepsilon)/((1+\varepsilon)n)^{1/\theta}$ where $\theta = \varepsilon(1+\varepsilon)$
$D_L \leftarrow \delta$, $l(e) \leftarrow \delta \quad e \in E$, $R_{sd} \leftarrow \phi \quad \forall(s,d)$
for $s \to d \quad \forall s \in V, d \in V$ **do**
  while $D_L < 1$ **do**
    $F_{sd} \leftarrow \{f : w_f\}$ /* sort flows according to MRTF in ascend order*/
    $P_{sd} \leftarrow \{B_a, path_1 : [b, \Delta b], path_2 : [b, \Delta b]\}$, $W_{sd} \leftarrow \sum b_p, \forall p \in P_{sd}$
    for path p in P_{sd} **do**
      $\Delta b \leftarrow b - B_a$
      while $\Delta b \geq 0$ **do**
        Augment flows $\sum w_f$ along the path p until $\Delta b < 0$
        $B_{sd} \leftarrow B_{sd} + \sum w_f$
        for each link e in p **do**. $l(e) \leftarrow l(e)(1+\varepsilon \sum w_f / c(e))$
        for each link e in p **do**. $b(e) \leftarrow b(e) - \sum w_f$
        $D_L \leftarrow \sum b(e) l(e)$, $B_a \leftarrow b - \sum w_f$
      end while
    end for
  end while
end for
return R_u, $F_u \leftarrow F_{sd}$
output: $\lambda = \max(B_{sd})/W_{sd}$, F_u

This algorithm follows the same vein in [16], and the time complexity of the algorithm is $O\left(\varepsilon^{-2}mn\log m\right)$, where m is the number of links, and the final flow is a $(1-3\varepsilon)$ -approximation to the optimal flow. In each iteration, algorithm 2 computes near-optimal routing and the corresponding bandwidth provision for each flow to maximize the throughput of the whole network. Unlike the algorithm in [16], the computation is organized on ToR pair rather than server pair, and we use precise path information for these grouped flows instead of a random path set. More important, we distribute many flows on multipath at the same time complexity based on the path information, which is totally different from finding admissible pair algorithm in [16].

5.2 Scheduling and incremental update

Although algorithm 2 has calculated the optimal path and corresponding bandwidth provision for all uncompleted flow, it is hard to implement the new routing immediately because the controller cannot install flow tables for all flows at the same time. Installing flow table for MRTF flows firstly is a basic principle for decreasing FCT, which provides a guide for us to enforce routing update. However, this principle cannot help us to accelerate the process of (re)routing. Therefore, we introduce incremental update and flow aggregation mechanisms to decrease the (re)routing time, and these two mechanisms can decrease the number of flow tables which can strengthen the scalability of this scheme. More specifically, when we install flow tables for the flows in R_u, we firstly remove the flow from R_u for which their routings unchanged, i.e., we only update the incremental routing, and we call this mechanism as incremental update. Moreover, we aggregate some flows which have part of or entire overlap path into a new flow. We put incremental routing information into $\{\Delta R\}_o$ and $\{\Delta R\}_n$, where $\{\Delta R\}_0$ is the set of flows which are already in network, while $\{\Delta R\}_n$ represents the new flows. For routing information in $\{\Delta R\}_0$, we need to execute flow rerouting with SFMM, while we just need to install flow table in $\{\Delta R\}_n$. More details are shown in algorithm 3.

Algorithm 3: Scheduling and incremental update

Input: R_c, R_u
Output: R_u

1: $\{\Delta R\} \leftarrow R_u - R_c$ /* $\{\Delta R\}$: incremental update routing information*/
2: classify $\{\Delta R\}$ into $\{\Delta R\}_0$ and $\{\Delta R\}_n$ /* $\{\Delta R\}_0$ is the set of flows which are already in network, while $\{\Delta R\}_n$ represents the new flows*/
3: sort all the flows in $\{\Delta R\}_0$ according to their remaining FCT in descend order
4: **for** f in $\{\Delta R\}_0$ **do**
5: Rerouting_flow(*f*) /* implemented with seamless flow migration mechanism*/
6: **end for**
7: sort all the flows in $\{\Delta R\}_n$ according to their remaining FCT in ascend order
8: **for** f in $\{\Delta R\}_n$ **do**
9: Install path for *f*
10: **end for**
11: **return** $R_c \leftarrow R_u$

In algorithm 3, we prefer to reroute the flow with large remaining FCT in $\{\Delta R\}_0$ or with small remaining FCT in $\{\Delta R\}_n$ first to decrease AFCT.

5.3 Seamless Flow Migration

To guarantee that the flow transmission will not be disrupted and suffer any packet loss during flow rerouting, we introduce the seamless flow migration mechanism, which can realize transparent rerouting for end servers. In OpenFlow networks, the essence of flow rerouting is that the controller installs a new path for the flow. In OpenFlow specification v1.5.1 [17], the flow will choose the flow entry with higher priority when it simultaneously matches two flow table entries. Therefore, when we carry out flow migrating, we can build a new path by installing the flow entry with lower priority, and then delete the old flow entry to realize seamless flow migration. More details are shown in algorithm 4.

Algorithm 4: Seamless Flow Migration

```
for f in {ΔR}_0 do
  compute(new_path); new_path is lower than that of old_path*/
  find cutover_switch(new_path, old_path);
  find bottleneck link e
  while e do
      find(sw∈cutover_switch);/*sw is migration switch before bottleneck link*/
     delete flow table of old_path in sw;
  end while
end for
```

We illustrate the flow migration algorithm with an example. **Fig. 3** shows a VL2 datacenter network topology, and there is a flow transmitting from *s* to *d*. Now the flow is going to migrate to a new path <*s*,3,4,1,9,10,*d*> from the path <*s*,3,2,1,7,10,*d*>, and the cutover switch set will be {3,1,10}. When we install the flow entries in the cutover switch set for the new path, we must ensure that the priority of the new flow entries is lower than that of the old flow entries, which is to prevent the new flow table from disrupting the data transmission, because it is unable to install all flow entries simultaneously. The controller should wait till all flow table entries are installed, and then execute the flow migration algorithm.

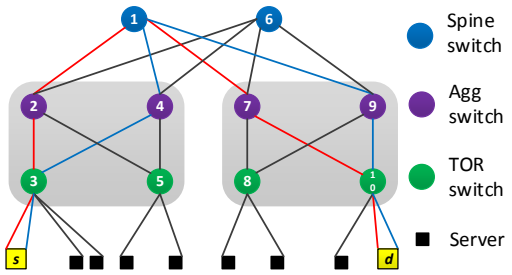


Fig. 3. VL2 datacenter network

During flow migration, the flow entries in the bottleneck link should be deleted firstly to get away from the congestion quickly. Regard to the other flow table entries, we adopt passive method, i.e., the flow table will be deleted automatically because of timeout. For example, if

the link between switch 1 and switch 7 is the bottleneck, we will preferentially remove the old flow entries from 1 to 7. In this migration mechanism, there always has at least one available path during the migration, and none of packets will be lost, so the flow migration is transparent for the end servers.

6. Performance Evaluation

We evaluate the performance of OFPT in Mininet [18] emulator. Mininet is a high-fidelity network emulation framework built on Linux container based virtualization, which creates a virtual network, switches and application code on a single machine. Its scale is smaller than production data center networks due to the single-machine CPU limitation (tens of Mbps link bandwidth compared to 1Gbps). Mininet has been shown to faithfully reproduce implementation results from [11], [8] with high fidelity [18], and has been used as a flexible testbed for networking experiments [3]. We realize OFPT on POX-carp [19] controller system. Moreover, both FatTree and VL2 are selected as the datacenter topology in our simulation, and we compare the performance of OFPT with TCP, ECMP and MPTCP under the same workloads which abide by the typical datacenter traffic patterns. In order to improve the accuracy of the experiment, each set of experiment runs for 10 runs lasting 200s, and we choose the average value as the experimental results.

6.1 Methodology

A. Topology

We set up the parameters of our test topologies according to [18], and the experimental scenarios are set as follows.

- k=8 FatTree. There are 128 servers and 80 switches, and each ToR switch connects with 4 servers, and the maximum parallel transmission path for each end-to-end connection in different pods is 8. We set the switch port buffer is 100 packets, and the link bandwidth between server and switch is 1Mbps and the link bandwidth between switches is 2Mbps.
- k=4 FatTree. There are 16 servers and 20 switches, and each ToR switch connects with 2 servers, and the maximum parallel transmission path for each end-to-end connection in different pods is 4. We set the switch port buffer is 100 packets, and the link bandwidth is 1Mbps.
- $d0$=2，$d1$=4，$d2$=4 VL2, where $d0$，$d1$ and $d2$ present the number of 10Gbps Ethernet ports in each ToR switch, aggregation switch and spine switch, respectively. There are 80 servers and 10 switches, and each ToR switch connects with 20 servers, and the maximum parallel transmission path for each end-to-end connection in different pods is 4. We set the switch port buffer is 100 packets, and the link bandwidth between server and switch is 1Mbps and the link bandwidth between switches is 10Mbps.

B. Benchmark workloads

Both [20] and [21] found that the traffic at the datacenter edge can be characterized by ON-OFF patterns, and the ON and OFF periods, and packet interval time draw from 3 different log normal processes, thus the traffic follows a heavy tailed distribution and bursty. We consider one flow size distribution from a cluster running web search [22], and the workloads exhibit heavy-tailed characteristics with a mix of small and long flows. In the web search workload, over 95% of the bytes are from 30% of flows larger than 1MB. In our

experiments, we use iperf traffic generation tool to generate ON-OFF flows on each server which abides by the traffic pattern above.

C. Communication model

In datacenters, the communication pattern between servers includes one-to-one, one-to-multi and all-to-all communication models [1], [3], [8], [19].

- One-to-one communication. One server only communicates with another server at the same time, and we use Stride(N/2) scheme as in [11]. In this scheme, a server with index x sends to the server with index (x+N/2) mod N, where N is the total number of servers.
- One-to-multi communication. One server communicates with multiple servers at the same time, and these servers are selected randomly.
- All-to-all communication. One server simultaneously communicates with all other servers, and the representative is Map-Reduce.

D. Schemes Compared

1) TCP：Standard TCP-New Reno is used as the baseline of our evaluation. The initial window is set to 12KB, and switches use DropTail queues with a buffer size of 100 packets. These are standard settings used in many studies [1].
2) ECMP：We use standard ECMP algorithm [1]. Other parameters are same as TCP.
3) MPTCP: We use standard MPTCP algorithm [8], and the number of subflow is determined by the number of available path. Other parameters are same as TCP.
4) OFPT：Our design is described in section 5，and we use TCP-New Reno in servers. Other parameters are same as TCP. We set the pooling time to be 5s.

Metrics: We use throughput and fairness to evaluate the performance of OFPT. The throughput is calculated by $\frac{B_{flow}}{Bisection\ \ Bandwidth}$, where B_{flow} is the throughput of the flow, and Bisection Bandwidth is the available bandwidth provided by network. Fairness is not an abstract concept for many distributed applications; for example, when a search application is distributed across many machines, the overall completion time is determined by the slowest machine. Hence worst-case performance matters significantly.

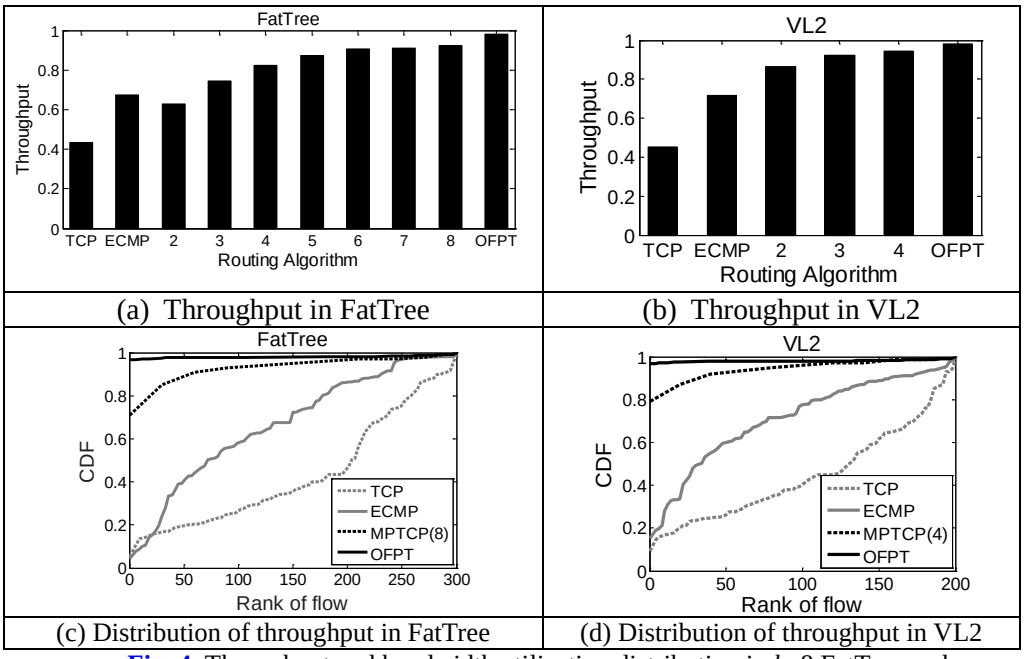
6.2 Experimental Results

6.2.1 Impact of Different Routing Mechanisms

We firstly test the performance of OFPT in one-to-one workload, and compare it with TCP, ECMP and MPTCP. **Fig. 4** presents the average throughput and CDF of single flow bisection bandwidth utilization in both k=8 FatTree and VL2 under different routing mechanisms.

Throughput **Fig. 4(a)** and **4(b)** show the throughput of TCP, ECMP, MPTCP (the number of subflow is from 2 to 8, and we use 2 to represent the MPTCP with 2 subflows) and OFPT in k=8 FatTree and VL2. We can learn that OFPT achieves the highest throughput in both FatTree and VL2. The performance of MPTCP depends on the number of subflow, and more subflows lead to higher throughput, which has been verified in [4] and [19]. Specifically, the throughput achieves 92.3% with 8 subflows, while 62.7% with 2 subflows in FatTree. In contrast, the throughput of OFPT is 98.1%, which is more than that of MPTCP with 8 subflows by 5.8%. Because only one path is available, the performance of TCP is rather poor, and just 43.5% of the bisection bandwidth can be utilized. ECMP also performs worse than OFPT due to its random hashing collisions, and its throughput is less than 67.5%. Due to only 4 paths existing

between each server pair in VL2, the number of subflow we select in MPTCP is less than 4. We get the similar conclusion that OFPT can efficiently improve the network throughput.



(a) Throughput in FatTree (b) Throughput in VL2

(c) Distribution of throughput in FatTree (d) Distribution of throughput in VL2

Fig. 4. Throughput and bandwidth utilization distribution in k=8 FatTree and VL2 under different routing mechanisms

Fairness **Fig. 4(c)** and **4(d)** show the bandwidth utilization of each flow with TCP, ECMP, MPTCP and OFPT in FatTree and VL2 respectively. Every flow's throughput is shown in ascend order. It is clear that both the bandwidth utilization and the fairness are improved in OFPT. In TCP, only few flows perform well, and the bisection bandwidth obtained by many flows is less than 0.5. ECMP performs better than TCP, but it still cannot make full use of all available bandwidth, and more than 23% flows' throughput are less than 0.5. In MPTCP with 8 subflows, more than 95.3% MPTCP flows utilize at least 0.8 of the bisection bandwidth, and none of them is less than 0.7. Compared with the other methods, OFPT achieves the best performance, and the throughput of each flow is higher than 0.9 in both FatTree and VL2.

In conclusion, OFPT can efficiently improve the throughput and fairness in both FatTree and VL2 under one-to-one communication model. We also find that each routing algorithm achieves better throughput in VL2 than in FatTree. It is VL2 that use a 10Mbps link among switches which coped with more bursty traffic, and this conclusion has also been shown in [4].

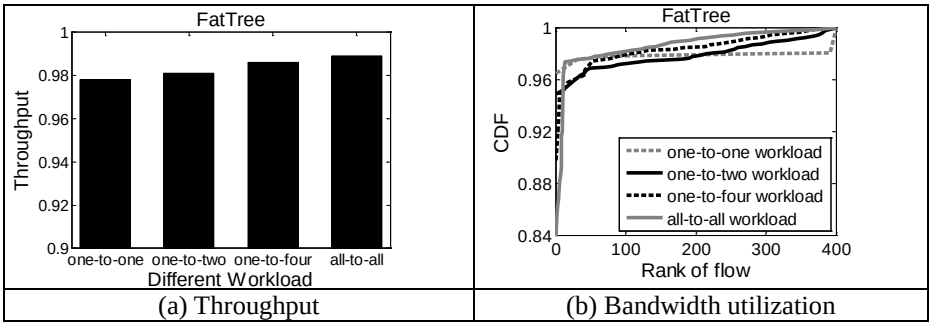
6.2.2 Impact of Different workloads

In order to verify the performance of OFPT in different communication patterns, we test the performance of OFPT under different workload in this section. Due to the CPU limitation, we test OFPT in k=4 FatTree, and we choose four communication workloads, i.e., one-to-one, one-to-two, one-to-four and all-to-all in our experiment. The experimental results are shown in **Fig. 5**.

Throughput **Fig. 5(a)** presents the throughput of OFPT under four different workloads, and we can learn that OFPT efficiently improve the network throughput in all workload models.

More specifically, OFPT achieves more than 0.97 throughput in every workload, and it even achieves 0.989 in all-to-all workload.

In contrast to one-to-one workload, one-to-*n* workload achieves better throughput, where *n* represents the number of servers that each source server communicates with. In one-to-*n* workload, there are *n* flows competing for the link bandwidth between the server and its ToR switch, so each flow can only get 1/*n* of the link bandwidth. Therefore, the transmitting bandwidth of each flow is smaller in one-to-*n* than in one-to-one workload. As OFPT adopt flow-level flow scheduling mechanism, smaller flows can achieve better performance on load balancing, and the throughput is improved as well. This conclusion is in accordance with the experimental results.



(a) Throughput (b) Bandwidth utilization

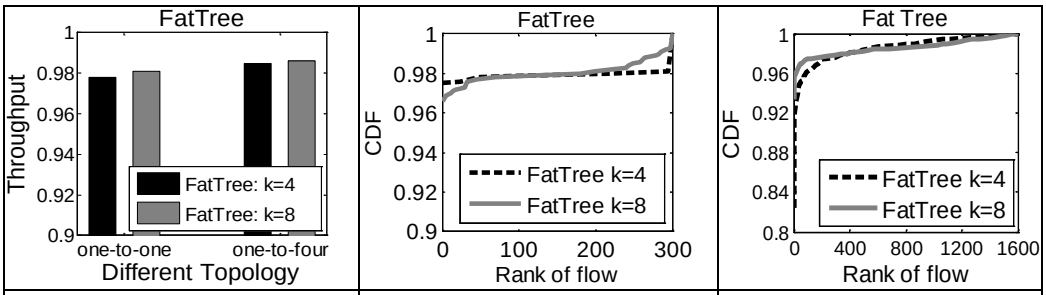
Fig. 5. Throughput and bandwidth utilization distribution of OFPT in *k*=4 FatTree under four different workloads

Fairness **Fig. 5(b)** presents the bandwidth utilization distribution of OFPT under four different workloads. In general, OFPT achieves good fairness in each workload, and the least throughput of single flow is still more than 0.838, while more than 98% flows reach 0.9 in our experimental results. However, in one-to-*n* workload, the fairness becomes worse with the increase of *n*. For example, the least bandwidth utilization is 0.965 in one-to-one workload, while the least bandwidth utilization is 0.838 in all-to-all workload. From the experimental results, we think that the number of flows in all-to-all workload is several times of that in one-to-one workload, and the number of flow assigns to each link is also increasing, which results in more congestion and thus some flows' throughput decreases. Although the fairness becomes slightly worse in all-to-all workload, the average throughput is still improved. Therefore, OFPT is still excellent in one-to-*n* workload.

6.2.3 Impact of Network Scale

In order to evaluate the scalability of OFPT algorithm, we test the performance in both *k*=4 and *k*=8 FatTree under one-to-one workload and one-to-four workload. **Fig. 6** shows the experimental results under different parameters.

Fig. 6(a) shows the throughput in different topologies and workloads. We can learn that OFPT achieves good throughput in both *k*=4 and *k*=8 FatTree, where their throughput is both more than 0.97. Furthermore, OFPT achieves better throughput in one-to-four workload than in one-to-one workload. **Fig. 6(b)** and **6(c)** shows the CDF of single flow's bandwidth utilization, and we can learn that the network scale has little impact on the fairness of flows in the same workload model. Therefore, we can conclude that OFPT has a good scalability.



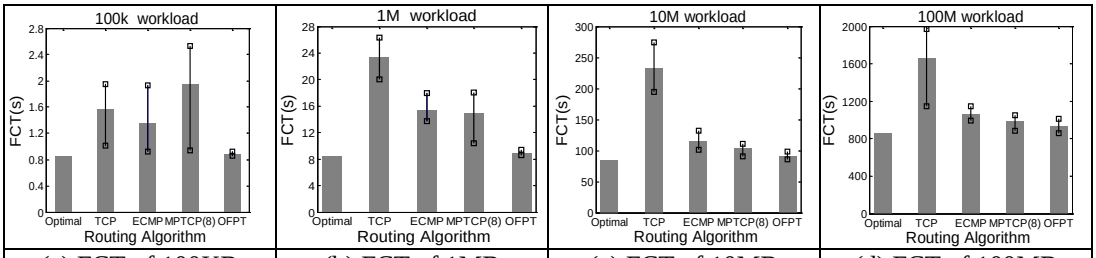
(a) Throughput (b) one-to-one workload (c) one-to-four workload

Fig. 6. Throughput and bandwidth utilization distribution in k=4 and k=8 FatTree under one-to-one workload and one-to-four workload

6.2.4 Flow Completion Time

While improving the throughput of datacenter network, OFPT also decreases flow completion times by using the MRTF scheduling and distribute bandwidth for short flow first. We test the FCT performance of OFPT in FatTree with k=8, and we set the link delay between server and ToR switches to be 5ms, and the link delay between switches to be 1ms as in [18]. **Fig. 7** shows the completion time of flows whose sizes are less than 100KB, 1MB, 10MB and 100MB under one-to-one workload, respectively. In our simulation scenario, the theoretical of bisection bandwidth is 1Mbps, so the optimal transmission time is equal to the value of average flow size dividing 1Mbps bandwidth.

No matter the flows are long or short, the AFCT in OFPT does not exceed the optimal value by 10%. Specifically, for short flows, the AFCT in OFPT is only more than the optimal by 4%, while TCP and ECMP increase by 84% and 60%, respectively. In addition, although MPTCP promotes the throughput to 95.3%, it extremely affects the AFCT of short flows. For example, the MPTCP with 8 subflows increases the AFCT of short flows (flow size is less than 100KB) by 24.6% and 129% compared to TCP and the optimal respectively, and this outcome is not unacceptable in datacenter. In summary, only OFPT can improve the throughput and decrease AFCT at the same time.



(a) FCT of 100KB (b) FCT of 1MB (c) FCT of 10MB (d) FCT of 100MB

Fig. 7. Flow completion time of TCP, ECMP, MPTCP and OFPT under one-to-one workload

6.2.5 Complexity of OFPT

To further verify the practicability of OFPT, we implement it in a testbed shown in **Fig. 8** and compare its performance with existing OpenFlow based scheme Hedera [11], which provides a deep discuss on the control overhead. Note that the Global First Fit (GFF) is only measured because Simulated Annealing is significantly more complex but does not provide much performance gain. In the testbed, there are nine Lenovo PCs with 2-core Intel Core i5-3470 3.2GHz CPUs, 4GB of RAM and one 1TB 7200RPM HDD, which run Ubuntu 12.04.5 LTS

operating system. One PC works as controller, and other eight PCs play the role of end servers. The OpenFlow switch is Pica8, and the bandwidth of each link is 1Gbps.We test the performance of OFPT and Hedera with different load under web search workload, and use average FCT, 95th percentile FCT and CPU utilization percent three metrics to present their performance. Specifically, we use CPU utilization percent of controller to evaluate the complexity of OFPT and Hedera, because their main functions are realized in the controller, and we can conclude that OFPT is practicable if the CPU utilization percent of OFPT is less than Hedera's. The experimental results are shown in **Fig. 9**.

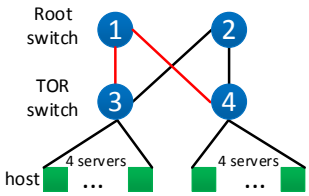
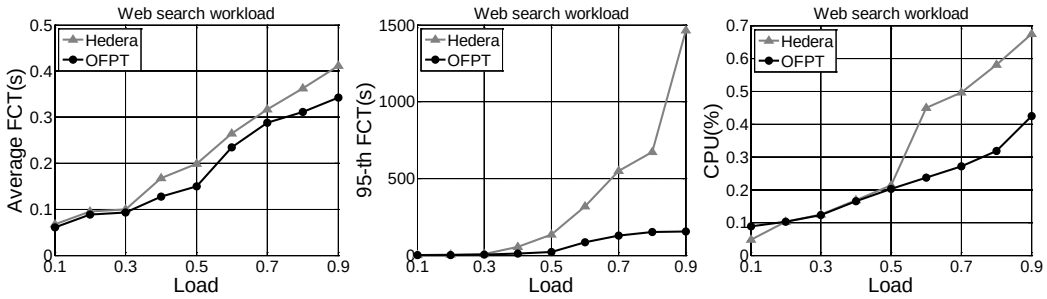


Fig. 8. Testbed topology

FCT. From **Fig. 9(a)** and **9(b)**, we can learn that OFPT performs better than Hedera in both average FCT and 95th percentile FCT. Specifically, OFPT provides 17% and 89.5% speedup than Hedera in AFCT and 95th percentile FCT respectively, which means that OFPT can realize better load balancing, i.e., each flow gets more transmitting bandwidth and thus suffers small FCT. With load increasing, OFPT gets obvious performance improvement due to more efficient bandwidth allocation algorithm than Hedera's GFF, because GFF cannot guarantee each flow getting the optimal transmitting path. Especially, OFPT obtains large performance improvement in 95th percentile FCT, which is due to bandwidth allocation and SFMM algorithm that can provide a better transmitting path for flows.



(a) Average FCT (b) 95th percentile FCT (c) CPU utilization percent

Fig. 9. The performance of OFPT and Hedera in web search workload

CPU. From **Fig. 9(c)** we can see OFPT produces smaller overhead on the SDN controller than Hedera overall, which means OFPT has better scalability and availability than Hedera. The good performance of Hedera has been verified by [11], which manifests that OFPT can be implemented in DCNs. When load=0.1, OFPT produces more overhead because OFPT maintains path information and computes more excellent path for flows. However, OFPT does not add more overhead on the controller because the CPU utilization of controller is less than 10%, thus OFPT is practicable. OFPT preinstalls flow tables for short flows, and it will decrease plenty of overheads on the controller for installing flow tables for short flows. In **Fig.**

9(c), we can see the CPU utilization of controller is less than 43% in OFPT when load is heavy. In all, OFPT achieves better performance as well as better scalability and availability.

6.2.5 Seamless flow migration mechanism

A. Experimental design

We employ iperf as our simulation tool and use a diamond topology shown in **Fig. 10**. Furthermore, we set the link bandwidth to be 1 Mbps and the size of switch buffer to be 100 packets. In order to simplify the experiment, we adopt static routing mechanism.

We choose UDP flows as workloads to avoid the impact of TCP congestion control. Simultaneously, we can easily derive the information about the packet loss and out-of-order issues.

In practical, the bandwidth occupied by UDP flow is 1Mbps and this flow lasts 10s. The flow delivers along the path <*h*1, *S*1, *S*3, *S*2, *h*2> in the first 5s, and then switches to the path <*h*1, *S*1, *S*4, *S*2, *h*2> in the later 5s. If the flow migration process has no packet loss around the 5th second, we can deem that SFMM realizes seamless flow migration.

The configuration process of static routing is as follows. When the switch connects to the controller, the controller installs two groups of flow entries in switch *S*1. The first group of flow entries is from S1 to S3, while the second group is from S1 to S4. We set the priority of the second group to be lower than the first group, so that the flow can transmit along the first path, and we simultaneously set the timeout of the first group to be 5s and the timeout of the second group to be 10s, so the flow will transmit along the second path due to timeout and the deletion of the flow entries.

B. Experimental results

The experimental results are shown in **Fig. 11**. When we send a 1Mbps UDP flow in *h*1, the throughput gets in *h*2 fluctuates between 964Kbps and 988Kbps, but there is no packet lost in the time interval of 4.5s-5.5s. If the sending rate of the UDP flow decreases to 0.8Mbps in *h*1, the throughput of *h*2 always is 800kbps and there is no packet lost in the experiment yet. Therefore, we think SFMM can realize the seamless flow migration.

We repeat this experiment for 10 times by loading UDP flows with different sending rate, and we find that SFMM can guarantee the flow migration without packet loss. However, there are 2 runs that the flow suffers slight out-of-order (2 packets) issue. From the experimental results, we find that the out-of-order issue is mainly caused by the buffered packets in the old path. When the flow migrates to the new path, the packets in the new path may reach *h*2 ahead of the packets buffered in the old path, and TCP can ideally deal with the slight disordered packets.

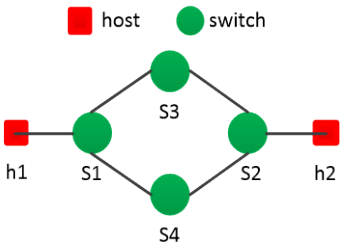


Fig. 10. Experimental Topology

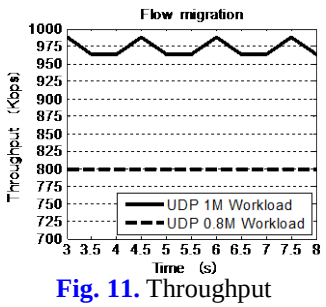


Fig. 11. Throughput

7. Conclusion

This paper mainly focuses on solving the issues of low throughput and poor load balancing existing in datacenters. We propose OFPT, which adopts the centralized control mechanism of OpenFlow and makes use of the parallel paths in dense interconnection datacenter networks, to allocate the traffic to all available paths. We also present a seamless flow migration mechanism, which can guarantee the flow does not suffer any packet loss during flow migration.

Both the theoretical analysis and experimental results show that OFPT can efficiently improve the throughput and load balancing in DCNs. The experimental results also show that OFPT increases the throughput up to 97.5% in both FatTree and VL2, which exceeds the TCP and ECMP by 74.7% and 36% respectively. Besides, OFPT achieves better load balancing and reduces AFCT. We also show that OFPT is scalable， both the network scale and network load models have little impact on its performance. Therefore, OFPT effectively addresses the datacenter TE issues. In the future work, we will devote to decrease the short flow completion time with OpenFlow technology in datacenter.

References

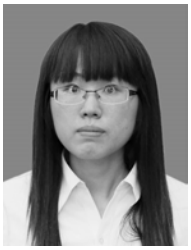
[1] M. Al-Fares, A. Loukissas and A. Vahdat, “A scalable, commodity data center network architecture,” in *Proc. of SIGCOMM*, pp. 63-74, August 17–22, 2008. Article (CrossRef Link).
[2] J. Mudigonda, P. Yalagandula, M. Al-Fares and J. C. Mogul, “Spain: Cots data-center ethernet for multipathing over arbitrary topologies,” in *Proc. of NSDI*, pp. 265-280, April 28-30, 2010.
[3] A. Greenberg et al, “VL2: a scalable and flexible data center network,” in *Proc. of SIGCOMM*, pp.51-62, August 17–21, 2009. Article (CrossRef Link).
[4] J. Cao, R. Xia, P. Yang, et al, “Per-packet load-balanced, low-latency routing for clos-based data center networks,” in *Proc. of CoNEXT*, pp. 49-60, December 9-12, 2013. Article (CrossRef Link).
[5] A. Kabbani, B. Vamanan, J. Hasan, et al, “FlowBender: Flow-level adaptive routing for improved latency and throughput in datacenter networks,” in *Proc. of CoNEXT*, pp.149-160, December 2–5, 2014. Article (CrossRef Link).
[6] A. Dixit, P. Prakash, Y. Hu and R. Kompella, “On the impact of packet spraying in data center networks,” in *Proc. of INFOCOM*, pp. 2130-2138, April 14-19, 2013. Article (CrossRef Link).
[7] D. Zats, T. Das, P. Mohan, D. Borthakur and R. Katz, “Detail: reducing the flow completion time tail in datacenter networks,” in *Proc. of SIGCOMM*, pp. 139-150, August 13–17, 2012. Article (CrossRef Link).
[8] C. Raiciu, S. Barr´e, C. Pluntke, et al, “Improving datacenter performance and robustness with multipath TCP,” in *Proc. of SIGCOMM*, pp.266-277, August 15–19, 2011. Article (CrossRef Link).
[9] K. He, E. Rozner, K. Agarwal, et al, “Presto: Edge-based load balancing for fast datacenter networks,” in *Proc. of SIGCOMM*, pp. 465-478, August 17–21, 2015. Article (CrossRef Link).
[10] M. Alizadeh, T. Edsall, S. Dharmapurikar, et al, “CONGA: Distributed congestion-aware load balancing for datacenters,” in *Proc. of SIGCOMM*, pp. 503-514, August 17–22, 2014. Article (CrossRef Link).
[11] M. Al-Fares, S. Radhakrishnan, B. Raghavan, et al, “Hedera: Dynamic flow scheduling for datacenter networks,” in *Proc. of NSDI*, pp. 19-33, April 28-30, 2010.
[12] T. Benson, A. Anand, A. Akella and M. Zhang, “MicroTE: Fine grained traffic engineering for datacenters,” in *Proc. of CoNEXT*, pp. 8-19, December 6–9, 2011. Article (CrossRef Link).
[13] N. McKeown, T. Anderson, H. Balakrishnan, et al, “OpenFlow: enabling innovation in campus networks,” *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69-74, March, 2008. Article (CrossRef Link).

[14] S. Jain et al, "B4: Experience with a globally-deployed software defined WAN," in *Proc. of SIGCOMM*, pp.3-14, August 12–16, 2013. Article (CrossRef Link).
[15] 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 Datacenters," in *Proc. of SIGCOMM*, pp. 63-74, August 17–21, 2009. Article (CrossRef Link).
[16] Madry A, "Faster approximation schemes for fractional multicommodity flow problems via dynamic graph algorithms," in *Proc. of the forty-second ACM symposium on Theory of computing*, pp. 121-130, May, 2010. Article (CrossRef Link).
[17] OpenFlow Switch Specification, Version 1.5.1. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.1.pdf, 2015.
[18] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz and N. McKeown, "Reproducible Network Experiments Using Container-Based Emulation," in *Proc. of CoNEXT*, pp. 253-264, December 10–13, 2012. Article (CrossRef Link).
[19] Pox-carp. [Online]. Available: https://github.com/noxrepo/pox/.
[20] T. Benson, A. Akella and D. Maltz, "Network Traffic Characteristics of Datacenters in the Wild," in *Proc. of IMC*, pp. 267-280, November 1–3, 2010. Article (CrossRef Link).
[21] Y. Chen, S. Jain, V. K. Adhikari, Z.-L. Zhang and K. Xu, "A First Look at Inter-Datacenter Traffic Characteristics via Yahoo! Datasets," in *Proc. of INFOCOM*, pp. 1620-1628, April 10-15, 2011. Article (CrossRef Link).
[22] M. Alizadeh, A. Greenberg, D. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta and M. Sridharan, "Datacenter TCP (DCTCP)," in *Proc. of SIGCOMM*, pp. 63-74, August 30–September 3, 2010. Article (CrossRef Link).

Bo Liu is currently a Ph.D. student at PLA University of Science and Technology. His research interests include network measurement and monitor, performance evaluation, software defined networking and datacenter network. E-mail: lbo.xidian@163.com

Bo Xu is currently an assistant-professor in the college of command information systems at PLA University of Science and Technology, Nanjing, China. His research interests include network measurement and software defined networking. Email: xubo820@163.com.

Chao Hu is currently an assistant-professor in the college of command information systems at PLA University of Science and Technology, Nanjing, China. His research interests include peer-to-peer multimedia communications, network modeling and software defined networking. E-mail: huchaonj@126.com



Hui Hu is currently a graduate student in the college of command information systems at PLA University of Science and Technology. Her research interests include network measurement and monitor, performance evaluation, software defined networking. E-mail: huhui_email@126.com.

Ming Chen received the Ph.D. degree from Nanjing Institute of Communication Engineering, China, in 1991. He is currently a professor in the college of command information systems at PLA University of Science and Technology, Nanjing, China. He held visiting position at Columbia University in 1999. He has published extensively in network architecture, network measurement and monitor, performance evaluation, distributed computing. E-mail: mingchennj@163.com