Liu *et al. Journal of Cloud Computing: Advances, Systems and Applications* (2021) 10:13
https://doi.org/10.1186/s13677-021-00226-w

Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH Open Access

A port-based forwarding load-balancing scheduling approach for cloud datacenter networks

Zhiyu Liu, Aqun Zhao and Mangui Liang*

Abstract

Today's datacenter networks (DCNs) scale is rapidly increasing because of the wide deployment of cloud services and the rapid rise of edge computing. The bandwidth consumption and cost of a DCN are growing sharply with the extensions of network size. Thus, how to keep the traffic balanced is a key and challenging issue. However, the traditional load balancing algorithms such as Equal-Cost Multi-Path routing (ECMP) are not suitable for high dynamic traffic in cloud DCNs. In this paper, we propose a port-based forwarding load balancing scheduling (PFLBS) approach for Fat-tree based DCNs with some new features which can overcome the disadvantages of the existing load balancing methods in the following aspects. Firstly, we define a port-based source-routing addressing scheme, which decreases the switch complexity and makes the table-lookup operation unnecessary. Secondly, based on this addressing scheme, we proposed an effective routing mechanism which can obtain multiple available paths for flow scheduling based in Fat-tree. All the path information is saved in servers and each server only needs to maintain its own path information. Thirdly, we propose an efficient algorithm to implement large flows scheduling dynamically in terms of current link utilization ratio. This method is suitable for cloud DCNs and edge computing, which can reduce the complexity of the switches and the power consumption of the whole network. The experiment results indicate that the PFLBS approach has better performance compared with the ECMP, Hedera and MPTCP approaches, which decreases the flow completion time and improves the average throughput significantly. PFLBS is simple and can be implemented with a few signaling overheads.

Keywords: Load-balancing, Link utilization ratio, Addressing scheme, DCNs, Edge computing

Introduction

Recently datacenter networks (DCNs) have became the most important infrastructure and attracted more attention in industry. A large number of DCNs have been deployed worldwidely with massive layered switches [1] and thousands of servers to interchange a great quantity of data. The DCN architectures (Leaf-Spine, Fat-tree, etc.,) with multi-tier switching layers need to provide the required network efficiency and flexibility to interconnect thousands of top of the racks (ToRs), each with tens-Tb/s aggregated traffic [2, 3]. The DCNs have to scale up to accommodate the tremendous increase in quantity of traffic [4, 5] and assign resources to tasks reasonably for the quality of service [6–8]. How to reduce the end-to-end delay, increase the throughput and keep the traffic load-balanced is a key issue. In most DCN architectures, multiple paths are commonly available between a pair of servers. Therefore, the data flows can be scheduled dynamically to evenly distribute traffic load on these paths. However, the traffic patterns in DCNs are quite different from those in the traditional Internet, so the flow scheduling is a very desirable but extremely challenging task.

*Correspondence: mgliang@bjtu.edu.cn
Institute of Information Science, Beijing Jiaotong University, Beijing, China

In order to find an effective load balancing method suitable for cloud DCN to improve network utilization, we have studied the current theoretical and practical solutions in this field [9, 10] and summarized the shortcomings of these solutions. The problem of routing flows through a capacitated network simultaneously is the multi-commodity flow(MCF) problem [11] which has been extensively studied from both theoretical and practical aspects. The main solution deployed in DCNs today is Equal-Cost Multi-Path (ECMP) [12]. ECMP employs a static hashing mechanism and hashes flows to the equal-cost paths. However, it is easy to cause congestion because of the large-flow collisions in a DCN, and far from optimal. Therefore, a variety of load-balancing approaches were proposed to address the problems of ECMP, which can be classified into the centralized solutions (e.g., Hedera [13], Mahout [14], Fastpass [15]), the switch-local solutions (e.g., FLARE [16], Presto [17], DRILL [18]), and the end-host solutions (e.g., MPTCP [19], FlowBender [20]). But all of them have some critical drawbacks. The centralized solutions are too slow for the traffic volatility and face severe scalability problems in today's DCNs. The switch-local solutions have good scalability and do not need to calculate per-path statistics, but lack the global view of a DCN and cannot deal with asymmetry very well. The host-based solutions such as MPTCP offer greater parallelism but are difficult to be deployed and make an already complex transport layer even more complicated due to the requirements of low latency and burst tolerance. Most of the existing solutions do not split flows onto multiple paths by making good use of the characteristic of DCNs.

Motivated by the above observations, we define a novel addressing and routing architecture for Fat-tree (also applicable to other regular topologies, such as VL2 [21], Portland [22], BCube [23], DCell [24]). Then, we propose a host-based dynamical load-balanced scheduling approach to maximize the network throughput through balancing the flows in DCNs. In this paper, aiming at the Fat-tree topology, a port-based load-balancing scheduling approach is proposed. It can solve the existing problems of the current methods. For example, it can provide faster response time for congestion comparing with the centralized methods, lower scheduling overhead comparing with the end-host methods and a global view comparing with the switch-local methods. Our main contributions can be summarized as follows.

Firstly, we propose a new addressing scheme which applies a port-based source-routing address (PA) as the forwarding address. We use a shim layer to implement the function below TCP/IP stack and the existing applications will not be affected. This addressing scheme renders the table-lookup operation unnecessary and reduces the complexity of the switches.

Secondly, we design an effective routing mechanism to obtain multiple available paths which is implemented in servers for load balancing. Each server has a global perspective of the whole network topology and only needs to maintain routing information of its own flows.

Thirdly, we present a port forwarding load-balanced scheduling (PFLBS) algorithm for Fat-tree based DCNs. It can select multiple paths for a flow and update them periodically.Meanwhile, PFLBS can schedule the flow to a new well-chosen path timely when the links along the old path become congested.

we conduct experiments to verify the efficiency of the load-balanced scheduling algorithm. The results show that PFLBS approach has better performance compared with the ECMP, Hedera and MPTCP approaches, which decreases the flow completion time and improve the average throughput significantly.

The remainder of the paper is structured as follows. The related work is given in "Related work" section. "Addressing scheme and load-balancing algorithm" section is dedicated to describe the port-based addressing scheme, routing mechanism, shim layer design and load-balancing algorithm in Fat-tree. "Performance evaluation" section is the performance evaluation. "Conclusion" section is the conclusion.

Related work

We have proposed a forwarding address which applies the port-based source routing [25]. In this paper, we aim to find an effective load balancing method to improve network utilization. Load balancing means that the resources in DCNs are shared by all tasks equally. It can be described mathematically by means of a performance criterion. In general, the purpose of load balancing is to optimize resource utilization, minimize transmission delay, maximize throughput and avoid overload of any single resource. Network load balancing aims at evenly scheduling traffic among multiple links by using simple routing protocols. The common method to balance load in DCNs today is ECMP. ECMP can statically strip flows across available paths using flow hashing and performs well for most of small flows. However, the static mapping results in congestion and network utilization degradtion because it can cause flow hash collisions easily and does not take current network utilization and flow size difference into account.

Therefore, many researchers have proposed some novel traffic scheduling mechanisms to balance finer-grained units of traffic. These methods can be classified into three categories: centralized solutions, switch-local solutions and end-host solutions.

The centralized solutions typically run a scheduling algorithm at a single server. In order to evenly balance the flows according to traffic patterns and link utiliza-

tion, it is essential for a load balancing approach to obtain global network information. Combining with the centralized network architecture such as SDN [26], some mechanisms can use controller to collect global network information and assign flows to proper paths through the Openflow protocols [27]. However, these solutions lack scalability because the overhead of collecting information, computing paths and deploying paths makes them impractical to respond timely in large-scale networks. Meanwhile coordinating decisions in the face of unpredictability and traffic burstiness is also a serious problem. For example, although Hedera's scheduler runs every 5 seconds and has the potential to run at subsecond intervals, recent studies [28, 29] have shown that the size and workloads of today's datacenters require parallel route to be setup on the order of milliseconds. It makes a centralized solution infeasible even in small DCNs [30].

The switch-local solutions select paths for flows at local switches without a global view of the network. They do not need to collect any global congestion information and make much hardware or protocol modifications. So the switch-local solutions always achieve high scalability. But these scheduling algorithms do not consider the realtime states of links for other switches so that they cannot adapt to changing data flows. For example, Presto proactively splits each flow into equal small sizes and then distributes them evenly to the network using ECMP. Compared with ECMP, Presto achieves higher throughput and lower flow latency under different workloads. However, although flows are assigned evenly at each soft edge switch respectively, the flows arriving at different soft edge switches cannot ensure uniform. For another example, inspired by the "power of two choices" paradigm which is used in the supermarket queue model, DRILL implements a random packet allocation scheme using the switch local information. When a packet arrives at a switch in DRILL, the switch will randomly pick two available ports and compare their queue length with a recorded port, and then the packet will be sent to the port with the lowest buffer among the three. But in Fat-tree topology, DRILL can only obtain the optimal port which has the minimum queue length in the upstream switches. The collisions may occur in the downstream switches and random packet allocation scheme does not work because the downstream path are deterministic.

The end-host solutions offer more parallelism and give more provable guarantees. Clove [31], TeXCP [32] and DARD [33] dynamically balance traffic through multiple paths between pairs of ingress-egress routers established by an underlying routing architecture and only require modifications to end-host software. However, they have very limited path condition information and thus can only predict whether a path is congested or not based on the common signals such as explicit congestion notification (ECN) or round trip-time (RTT). These solutions rely on the virtualization technology, and the corresponding algorithms are implemented entirely in the virtual switches of hypervisors. They are not really end-host-based approaches. MPTCP splits a flow into multiple subflows, leverages the multi-paths between end-hosts and setups multiple sub-connections to make full use of the link bandwidth. However, MPTCP needs to be deployed on the multi-homed servers to transmit subflows separately and it cannot control the path of subflows, so subflows are usually handled by ECMP according to the source and destination addresses.

In a word, although each of the three solutions has its own advantages, there are also obvious drawbacks. The problems of the centralized solutions are high cost and slow response to congestion. The switch-local and end-host solutions are based on distributed scheduling and lack of global perspective.

In this paper, we propose a port-based source-routing addressing and routing algorithm for Fat-tree based DCNs. It advances the software-defined network concept by pushing the control functionalities to servers and reducing the computing and storage of switches. Then, we design a simple shim layer to implement the functionality for probing "good" paths and encapsulating packets. In this way, servers can monitor large flows, split them into subflows and transmit them through multiple "good" paths. Comparing to switch-local solutions, PFLBS obtains more congestion information of all paths from a source to a destination and makes better routing decisions. Comparing to the centralized solutions, PFLBS schedules paths for flows independently in their respective servers and avoids generating huge signaling overhead in a centralized controller.

Addressing scheme and load-balancing algorithm

PFLBS is a flow splitting and routing algorithm designed for Fat-tree network. In this section, we describe the port-based address, introduce the simple routing mechanism, design the shim layer protocol and present the PFLBS approach.

Port-based addressing and forwarding

A general Fat-tree model is a k-port n-tree topology. We use a special instance with n = 3, which is usually discussed in the DCN literature. Figure 1 shows the Fat-tree topology with 4 ports. The labeling mechanism in [34] is adopted to identify the locations of switches and servers in this topology. In networking, we consider that the identification address is used in the control plane to identify a node and the forwarding address is used in the data plane to determine the output port to which a packet will be sent. They should be separate addresses. Now-

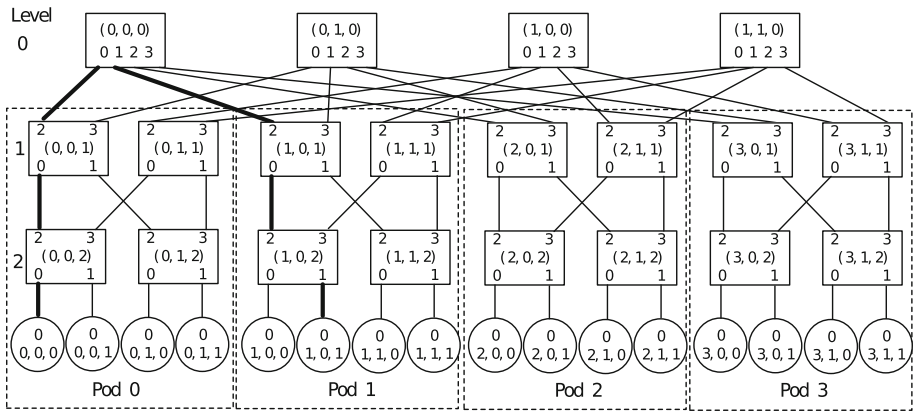


Fig. 1 The 4-port Fat-tree topology

days, IP address and MPLS are widely used in networks and represent two kinds of typical forwarding addresses. IP is a node-based addressing which is coded by numbering nodes in a network, while MPLS is a label addressing which is coded by numbering virtual switching labels in a network. Both of them use lots of memory space to save routing table or label mapping table on a switch.

To reduce the complexity of the switches, we define a port-based source-routing address which is coded by numbering the ports of each node along the path from the source to the destination. As is show in Fig. 1, if the source server (0,0,0) sends data to the destination server (1,0,1) along the path with the bold lines, the PA can be expressed as {0,2,2,1,0,1}, where the digits 0, 2, 2, 1, 0 and 1 are the output ports of nodes (0,0,0), (0,0,2), (0,0,1), (0,0,0), (1,0,1) and (1,0,2), respectively. Each output port in the PA is called as a PA element.

When a source sends a packet to a destination, it will encapsulate the PA in the packet. When receiving a packet, a switch only needs to execute the following actions as shown in Fig. 2: (1) extracting the first PA element x from the packet and removing it; (2) forwarding the packet to the output port indicated by x. As we can see, this addressing and forwarding scheme makes the table-lookup operation unnecessary and the switch can be simple and the scheme also can be used in other DCN topologies. So far, we achieve a simple data plane for DCNs.

A novel routing mechanism

Fat-tree is a regular DCN topology. By leveraging the regularity and the characteristics of the PA, a simple and efficient routing mechanism becomes feasible. Therefore, we push most of the routing function to the servers almost without switch involvement. It offers many obvious advantages in DCN networks. Firstly, this routing mechanism places most of the control functionalities in servers and keeps switches much simpler. Secondly, each server only needs to maintain its own routing paths for local flows and runs this mechanism independently. Thirdly, because of the regular characteristic for the Fat-tree topology, all the servers completely know the topology and do not need to store other information except for the parameter k. The routing mechanism will enumerate all of the shortest paths according to the source and the destination locations in the first step, then choose one or more suitable paths among all the available ones. This routing mechanism mainly focuses on the expression of the paths rather than the routing mechanism. We will introduce the detailed path selection method in the next subsection.

The routing algorithm is shown in Algorithm 1, which lists all the paths from the source to the destination in different conditions. If two servers are connected to the same level-2 switch, the unique shortest PA is $\{0, d_2\}$. For example, the shortest PA from (0,1,0) to (0,1,1) is 0,1 in Fig. 1. If two servers are connected to the same pod but not same level-2 switch, the shortest PAs are 4-hop and the total number of shortest paths is k/2. For example, one of the shortest PAs from server (0,0,0) to (0,1,1) is {0,2,1,1} in Fig. 1. If two servers are connected to different pods, the shortest PAs are 6-hop and the total number of shortest paths is $\frac{k^2}{4}$. For example, one of the shortest PAs from (0,0,0) to (1,1,1) is {0,2,3,1,1,1} in Fig. 1. In the rest of the paper, we only consider the complex situation of servers in different pods and focus on the 6-hops PAs. As is shown in Algorithm 1, PAs have some characteristics that the first PA element is always 0 and the last three elements depend on the destination locator. The second element x and the third element y are variable and values range from k/2 to k-1 respectively. Therefore, the time and space complexities are both $O\left(k^2\right)$.

The next step is to choose the most suitable paths. We propose two different PA selection strategies: the non-switch-assisted method and switch-assisted method. The

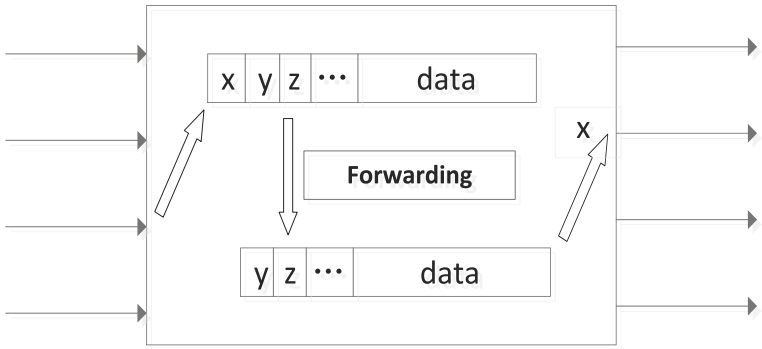


Fig. 2 Port-based forwarding in switch

Algorithm 1 PA enumeration algorithm in Fat-tree

Input: Input the source (s_0, s_1, s_2) and the destination (d_0, d_1, d_2).

Output: Output all the PAs from source to the destination

if $s_0 = d_0$ **then**
  if $s_1 = d_1$ **then**
    PA = $\{0, d_2\}$
  else
    PA = $\{0, x, d_1, d_2\}, x \in [k/2, k-1]$
  end if
else
  PA = $\{0, x, y, d_0, d_1, d_2\}, x, y \in [k/2, k-1]$
end if

non-switch-assisted method chooses a fixed path according to some specific traffic patterns. It does not need to encapsulate the signaling packets in servers and add control functionality to switches. For example, a server will choose a path with PA $\left\{0, d_2 + \frac{k}{2}, d_1 + \frac{k}{2}, d_0, d_1, d_2\right\}$ which is called destination-based policy under one-to-all traffic pattern. Here the second and the third PA elements are chosen depending on the destination locator, so it can be named as the destination-based policy. This policy can make sure that the selected paths distributed evenly in the DCN under one-to-all traffic pattern. Similarly, there is also source-based policy that can make sure that selected paths distributed evenly in the DCN under all-to-one traffic pattern. Although each server can change its PA-selection policy in terms of different traffic patterns and contribute to balancing the traffic load, it cannot ensure the selected paths to be distributed evenly in the network under all traffic patterns. The adaptive PA-selection method takes advantage of the fact that each PA-selection policy has its own superiorities under some specific traffic patterns. For instance, when an arbitrary server in pod p_1 sends packets to all the servers in pod p_2, the selected paths will be evenly distributed over all the upward links of the level-2 switch connected with the source server, all the level-1 switches and their upward links in the source pod p_1, all the level-0 switches and their downward links to the destination pod p_2, and all the switches and links in pod p_2. Similarly, the source-based policy can ensure the selected paths distributed evenly in the DCN under all-to-one traffic pattern. Therefore, in the adaptive PA-selection method, each server can independently adjust its PA-selection policy in terms of different traffic patterns. The adaptive PA-selection method is conducive to balancing the traffic load, yet it cannot cope with bursty traffic and dynamically adjust the routing policy. The switch-assisted method achieves bandwidth utilization of the available paths by sending signaling messages periodically. It adds some control signaling overheads to servers and a little control functionality to switches, but improves the performance of load balancing significantly. We focus on the implement of the approach in the following subsection.

Shim layer for multipath detection

In order to implement the port-based addressing and switch-assisted approach, we design a shim layer below TCP/IP including the shim headers and address tables, which is shown in Fig. 3. There are two types of shim headers: data header ("Type"=0) and signaling header ("Type"=1). Data header is used for packet transmission if the PA has existed in the address table. Signaling header is used for probing available PAs and obtaining port information of passing switches when a server connects to a new destination. Therefore, each switch needs to add a little control functionality to parse the signaling message and obtain port information by which the signaling message is forwarded. Meanwhile, the shim layer maintains an address table (AT) and an alternative address table (AAT) indexed by the destinations (through the "Des" field). The AT stores the primary PA which is currently in use for

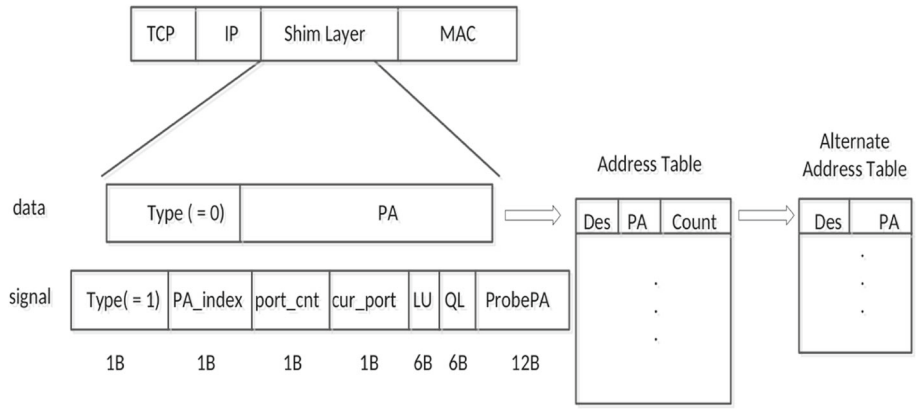


Fig. 3 The design of the shim layer

the destination and the AAT stores other alternative PAs, which are used for switching path if the primary PA fails to meet the condition.

When the source sends data to the destination, the shim layer will firstly search the AT for the primary PA according to the destination. If this entry exists, a data header will be created. The "Type" field of the shim header is set to 0 and the "PA" field is loaded with the primary PA, which means that this packet is a data packet and is sent out along the path indicated by the primary PA. If the entry does not exist, the shim layer will generate a signaling message with the "Type" field be 1 to probe port information of passing switches along each available shortest path. All the shortest PAs can be calculated by Algorithm 1. Then, we explain the specific meaning of each filed. The "PA_index" field represents the index of PA. When a signaling message returns to the source, this field is used to identiy which PA has been detected. The fields "port_cnt" and "cur_port" represent that how many ports of information need to be collected in total and how many have been collected, respectively. The fields "LU" and "QL" are used to record the link utilization ratio and queue length of each forwarding port that the signaling has passed through. The "ProbePA" field is used to store address for the signaling message. ProbePA is designed as a loopback PA which enables a server sends a signaling message to itself without other servers involved. In Algorithm 1, PA is expressed as $\{0, x, y, d_0, d_1, d_2\}$ (line 8) when a source and a destination are in different pods. ProbePA is expressed as $\{0, x, y, d_0, d_1, x, y, s_0, s_1, s_2\}$ which sends a message along the PA and let it return along the reversed path. Note that ProbePA does not use the d_2 port to send a message to the destination at the edgeswitch of downstream. As a result, each server can implement paths detection by itself.

Load balancing measure metric

In this section, we will present how to measure the load imbalance degree of a path and the scheduling trigger threshold. When a signaling message returns, the link utilization ratio and queue length of each forwarding port can be obtained through the field "LU" and "QL". The link utilization ratio is used to depict the load state of a link, which is defined as

$$\gamma_{u,v}(t) = \frac{b_{u,v}^{used}(t)}{B_{u,v}}, \tag{1}$$

where $\gamma_{u,v}(t)$ is the link utilization ratio of the link $l_{u,v}$ between switches u and v. $b_{u,v}^{used}(t)$ refers to the occupied bandwidth of the link $l_{u,v}$, while $B_{u,v}$ is the capacity of the link $l_{u,v}$. Each switch periodically computes the link utilization ratio of all its links. The queue length $\lambda_{u,v}(t)$ is used to describe the delay state of a link. The queuing delay of packets will increase with the growing of queue length. In DCNs, flows can be classified into small flows and large flows according to their size. Small flows are sensitive to delay which require the queue length is as short as possible and large flows are sensitive to throughput which require the link utilization ratio is as low as possible.

We define three metrics to depict the state of a path: the path bandwidth utilization ratio $\gamma_{PA}(t)$, the variance of utilization ratio $\delta_{PA}(t)$ and the average queue length $\overline{\lambda_{PA}(t)}$. The path bandwidth utilization ratio $\gamma_{PA}(t)$ is the maximum link utilization ratio of all the links along the path represented by PA, which is defined as

$$\gamma_{PA}(t) = \max_{\forall l_{u,v} \in PA} \{\gamma_{u,v}(t)\}. \tag{2}$$

The average bandwidth utilization ratio of a path is defined as

$$\overline{\gamma(t)} = \frac{\sum_{l_{u,v} \in PA} \gamma_{u,v}(t)}{N}. \tag{3}$$

where N is the number of links along PA. Next, the variance of utilization ratio $\delta_{PA}(t)$ is used to evaluate the load fluctuation along the path, which is defined as

$$\delta_{PA}(t) = \frac{\sum_{l_{u,v} \in PA} \left[\gamma(t) - \gamma_{u,v}(t)\right]^2}{N}. \quad (4)$$

Finally, $\overline{\lambda_{PA}(t)}$ is used to evaluate the queuing delay for the path, which is defined as

$$\overline{\lambda_{PA}(t)} = \frac{\sum_{l_{u,v} \in PA} \lambda_{u,v}(t)}{N}. \quad (5)$$

The three metrics are important parameters in our work to evaluate the "good" paths. $\gamma_{PA}(t)$ and $\delta_{PA}(t)$ are used to estimate bandwidth capacity and stability of a path. Large flows depend on the metrics to choose the suitable PAs. $\overline{\lambda_{PA}(t)}$ is used to choose the optimal PA with the minimum delay for a small flow. When the path selection is finished, each server will detect the unbalance degree of the paths periodically. In order to determine the time to switch or reselect PAs, we define the thresholds to trigger the dynamical flow scheduling. γ^* and δ^* are thresholds for $\gamma_{PA}(t)$ and $\delta_{PA}(t)$ respectively, which directly represent the scheduling frequency. The lower γ^* and δ^* are, the more evenly flows are distributed but the more frequently flows will be scheduled once the threshold conditions are not satisfied. There is no threshold for $\overline{\lambda_{PA}(t)}$ because small flows are short-lived and do not need to schedule frequently. Like the small flows in ECMP, once the optimal path is selected by $\overline{\lambda_{PA}(t)}$, it will be used until the flow finishes.

Port forwarding load-Balancing scheduling algorithm

The important purpose of our PFLBS algorithm is to ensure that the traffic load of each server be evenly distributed among the available links. It uses a local scheduling method in servers to implement global load balance in the whole network. The algorithm consists of two steps: multi-path selection and flow scheduling. In the step of multi-path selection, when a new flow needs to be transmitted, the shim layer will generate signaling messages to probe all the available paths. After all the messages return, the utilization ratio and queue length of all the links along each path have been collected in the fields "LU" and "QL". Then, the values of $\overline{\lambda_{PA}(t)}$, $\gamma_{PA}(t)$ and $\delta_{PA}(t)$ for each path can be calculated easily by "LU" and "QL". Next we propose a multipath routing selection algorithm to select the "good" paths according to the metrics. Firstly, we sort all the PAs based on the average queue length $\overline{\lambda_{PA}(t)}$ and choose the PA with the minimal $\overline{\lambda_{PA}(t)}$ as the optimal path. The PA will be stored in the Address Table(AT) and used for a new flow transmission. This is because that more than 80% of flows are small flows in DCNs. Therefore, when a new flow appears in the network, it should be regarded as a small flow with latency-sensitive characteristic and uses the above PA to transmit data. The field "count" in AT is used to record the number of packets that a flow has sent. We define L^* as a threshold to identify whether it is a small flow or not. If the "count" does not exceed the threshold, it is a small flow. Otherwise, it is treated as a large flow. Secondly, we select some alternate paths according to $\gamma_{PA}(t)$ and $\delta_{PA}(t)$ and store them in the Alternate Address Table(AAT). The remaining PAs are sorted in the ascending order of $\gamma_{PA}(t)$ first. If there are multi PAs with the same $\gamma_{PA}(t)$, we sort these PAs in the ascending order of $\delta_{PA}(t)$. Then we choose the first "n" PAs from the sorted results and store them in the AAT. The selection of value "n" will vary with the parameter k-port in Fat-tree. In this paper, the value of "n" is set to 2. The $\gamma_{PA}(t)$ represents whether a path has enough remaining bandwidth to accommodate a large flow, and the $\delta_{PA}(t)$ reflects whether there are potential hot-spot links in a path or not. The alternate paths are mainly used to migrate a flow to the optimal path once the flow is classified as a large flow. The lower the values of $\gamma_{PA}(t)$ and $\delta_{PA}(t)$ are, the better a path is. So far, one best path and two alternate paths are obtained through the multi-path routing selection algorithm as is shown in Algorithm 2.

Algorithm 2 Multi-path Routing selection algorithm in Fat-tree

Input: Input a source (s_0, s_1, s_2) and a destination (d_0, d_1, d_2).

Output: optimal PA in the AT and alternative PAs in the AAT

1: the source enumerate all the available PAs using Alg.1 and calulate ProbePAs
2: **for** $ProbePA_i \in ProbePAs$ **do**
3: generate a signaling message and send it out to collect the link utilization ratio and queue length
4: calculate $\gamma_{PA}(t)$, $\delta_{PA}(t)$ and $\overline{\lambda_{PA}(t)}$ for the path
5: **end for**
6: sort all the PAs in the ascending order of $\overline{\lambda_{PA}(t)}$
7: choose the path with the minimal $\overline{\lambda_{PA}(t)}$ as the optimal path and store it in the AT
8: sort the remaining paths in the ascending order of $\gamma_{PA}(t)$
9: **if** there are multiple paths with the same $\gamma_{PA}(t)$ **then**
10: sort these paths in the ascending order of $\delta_{PA}(t)$ based on the previous sort result
11: **end if**
12: choose the first two paths as the alternate paths and store them in the AAT

In the next stage, the server will monitor the PAs in the AT and AAT periodically to keep load balanced in the network during local data flow transmission. When

a detection cycle comes, our PFLBS algorithm will first check the "count" of the PA in AT. If the value is less than the threshold L^*, the flow is regarded as a small flow and the current path will continue to be used. Otherwise, it is identified as a large flow which needs to be scheduled. We firstly update $\gamma_{PA}(t)$ and $\delta_{PA}(t)$ of the PAs in AT and AAT by sending signaling messages. If the current PA in AT is satisfied with $\delta_{PA}(t) < \delta^*$ and $\gamma_{PA}(t) < \gamma^*$, the large flow can continue using this path to transmit data. Otherwise we choose an optimal PA in AAT to replace the previous PA in AT. Then, the paths which do not meet the metrics will be updated based on $\gamma_{PA}(t)$ and $\delta_{PA}(t)$ using the multipath routing selection algorithm. The detailed process of the algorithm is shown in Algorithm3. Note that each server uses the scheduling algorithm to monitor only the PAs in AT and AAT by itself, and does not need to update the PAs unless they cannot meet the threshold conditions. It reduces the overhead of signaling messages significantly, and does not affect the network performance. Although we do not choose the optimal PA all the time, the network load can be distributed evenly among all the links.

Algorithm 3 Port-based forwarding load-balancing scheduling algorithm

Input: Input the paths in AT and AAT, δ^* and γ^*
Output: Output load-balanced scheduling

1: periodically update $\delta_{PA_i}(t)$ and $\gamma_{PA_i}(t)$ of the paths in the AT and AAT by sending signaling messages
2: **if** if the field "count" $< L^*$ **then**
3: **for** PA in ATT **do**
4: **if** $\delta_{PA_i}(t) \geq \delta^*$ or $\gamma_{PA_i}(t) \geq \gamma^*$ **then**
5: select the optimal path to replace it using Alg.2
6: **end if**
7: **end for**
8: **else if** $\delta_{PA_i}(t) \geq \delta^*$ or $\gamma_{PA_i}(t) \geq \gamma^*$ in AT **then**
9: select the optimal path in AAT to replace the current path
10: use line 3-7 to update the path in AAT
11: **else**
12: use line 3-7 to update the path in AAT
13: **end if**

Handing asymmetry

Until now, the design of load-balanced scheduling algorithm has assumed that it is symmetric in Fat-tree. But the symmetric network may experience failure that causes asymmetry if a link between two switches fails (we do not consider link failure between a server and an edge switch in this paper). Although our scheduling algorithm can choose an alternate path from AAT quickly when the current path fails, the failed link cannot be localized. Therefore, a fault tolerance mechanism is needed to deal with this issue. An end-to-end approach is proposed with the same design philosophy by taking advantage of PA and the regularity of the Fat-tree topology. It should be noted that because a node failure can be considered as multiple link failures for all its links, and a link congestion has the same consequence as a link failure, we only describe how to deal with the link failures in the rest of this paper.

The motivation of the end-to-end approach is to continue placing all the fault-tolerance functionalities in servers, and keep switches simple. The process to deal with link failures are as follows. Generally, the Fat-tree topology has some regularity and does not need to be stored, but it is broken by the failures. Thus the servers need to store the dynamic topology information. We store the information with the form of unavailable prefix tables rather than the entire topology. In this way the storage space will be decreased, because the ratio of the number of faulty links and the number of normal ones is usually small. In addition, taking advantage of the characteristic of PA, the unavailable prefix table entry is in the form of $\{p_0, p_1, ..., p_i\}$, which is helpful for the servers to simplify the operations when dealing with link failures. The design includes probing the path(s) with link failures, locating the faulty link along the path, converting the failure information to the form of the unavailable prefix table entry, and distributing the information to the other servers. The detailed procedure to handle the link failures is designed as follows.

When a signaling message is sent to collect the links information for a flow, the server will set a timeout interval for the message. If timeout event and retransmission happen three times in a row, the fault tolerance mechanism will start to search link failures. The server will sends 3 locating packets along the path to the relevant nodes and let them return to locate the link failure (line2-16).Similar to a signaling message, these packets does not involve the destination nodes and are controlled completely by the source server using loop PAs. It should be noted that the algorithm only can detect a single link failure along a path. For multiple link failures on a path, more probing messages must be sent to traverse the network more than once. When a server locates a link failure, it will translates the link failure to an unreachable prefix table entry and store it in the server according to Algorithm 4. The information stored in a server depends on not only the location of the link failure, but also the location of the server. The server locating the failure will distribute the failure information to all the other servers in the DCN. Next, they will update their unavailable prefix tables. To decrease the overhead, we design a recursive distribution policy, leveraging the property of Fat-tree. The information is firstly sent to one server in each pod, and then these

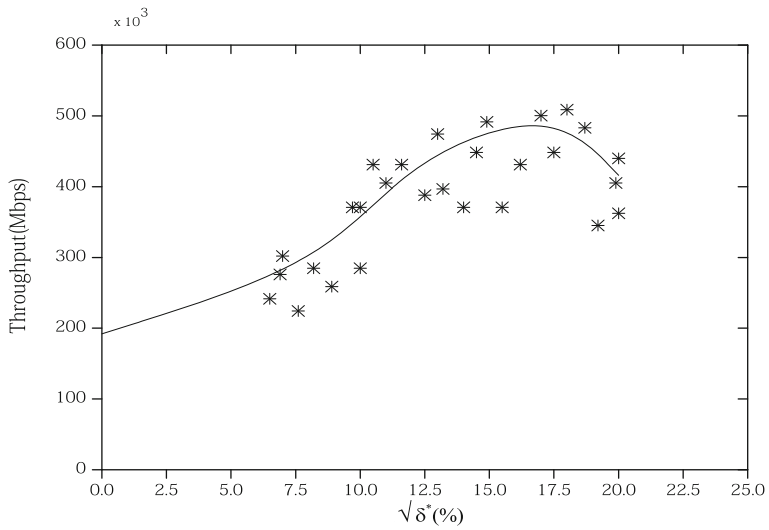


Fig. 4 Relationship between throughput and threshold $\sqrt{\delta*}$

servers send the information to other servers in their own pods using the same recursive policy. If a server wants to send a packet, it will choose a path which does not involve any link failures by selecting a ProbePA not matching the unavailable prefix entries in the table. Therefore, a server only needs to execute simple address matching operations to avoid using faulty links when sending messages.

Performance evaluation

In this section, we evaluate the performance of PFLBS through simulation system. We compare PFLBS with ECMP, Hedera and MPTCP [35], the state-of-the-art multipath transport protocol. ECMP can uniformly randomize the outgoing flows across a set of ports and work well for small flows. Hedera uses 500ms as the scheduling period. Note that 500ms is an optimistic interval for Hedera from the afforded overhead point of view, because it uses periodic polling to pull the per-flow statistics from all the edge switches for large flow detection. It takes a lot of time to assign paths for the large flows and install paths to the related switches. For MPTCP, we use 8 sub-flows for each TCP connection as [35] recommendation. The results show that PFLBS achieves higher throughput and lower mean FCT comparing with other scheduling approaches under heavy loads in Fat-tree. Details of the evaluations are as follows.

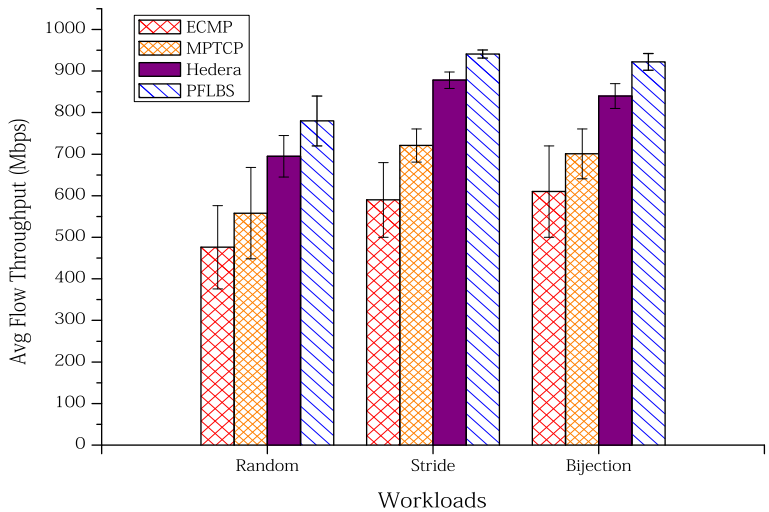


Fig. 5 Large flow throughput for ECMP, MPTCP, Hedera and PFLBS in random, stride and random bijection workloads

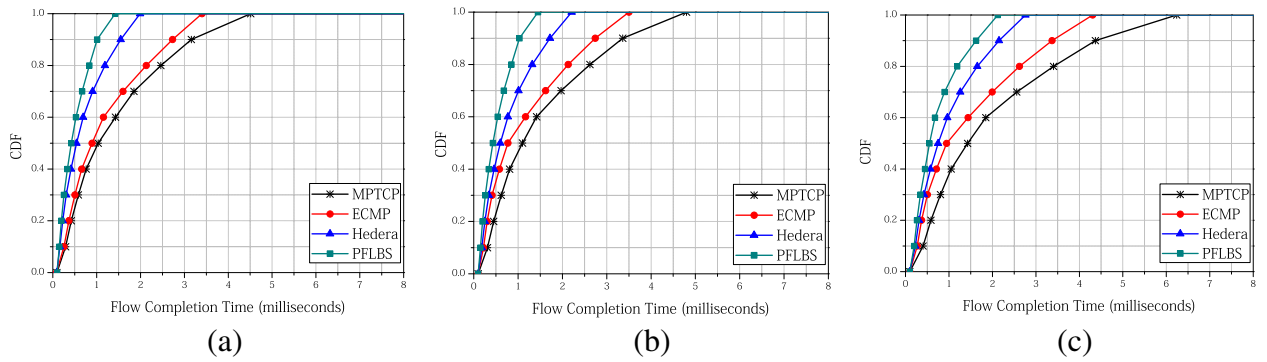


Fig. 6 Small flow FCT of MPTCP, ECMP, Hedera and PFLBS in **a** stride, **b** random bijection and **c** random workloads

Algorithm 4 Failure probing and prefix table-updating algorithm

Input: Input the source (s_0, s_1, s_2) and the destination (d_0, d_1, d_2).

Output: Output the unavailable prefix table entries in the server

the source sends signaling messages when a new flow need to be transmitted
if the signaling message with ProbePA=$\{0, x, y, d_0, d_1, x, y, s_0, s_1, s_2\}$ does not return in a pre-set time **then**
  sends a loopback message with PA=$\{0, x, s_1, s_2\}$
  if the message does not return **then**
    the link $(s_0, s_1, 2)$ to $(s_0, x - k/2, 1)$ fails
    add {0,x} to unavailable prefix table
  else
    sends a loopback message with PA=$\{0, x, y, s_0, s_1, s_2\}$
    if the message does not return **then**
      the link $(s_0, x-k/2, 1)$ to $(x-k/2, y-k/2, 0)$ fails
      add {0,x,y} to unavailable prefix table
    else
      sends a loopback message with PA=$\{0, x, y, d_0, y, s_0, s_1, s_2\}$
      if the message does not return **then**
        the link $(x - k/2, y - k/2, 0)$ to $(d_0, x - k/2, 1)$ fails
        add $\{0, x, y, d_0\}$ to unavailable prefix table
      else
        the link $(d_0, x-k/2, 1)$ to $(d_0, d_1, 2)$ fails
        add $\{0, x, w, d_0, d_1\}(k/2 \leq w \leq k-1)$ to unavailable prefix table
      end if
    end if
  end if
end if

System setting

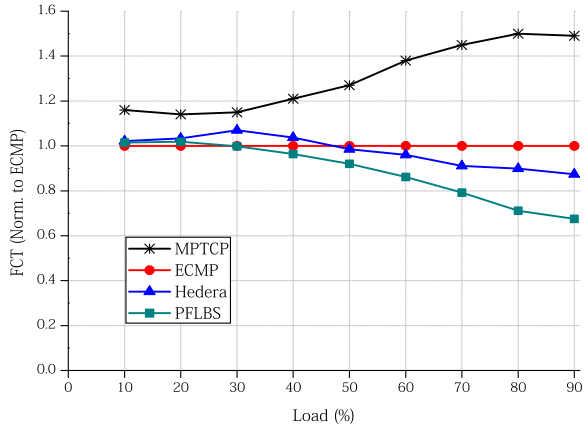
PFLBS is evaluated by using OMNET++ simulator in a 16-port Fat-tree topology with 1024 servers and 320 switches. Each server has 1 Gbps NIC and each link has 1 Gbps capacity. The propagation delay of each link is setted to be 0.05 μs, which means that the link is approximately 10 m. The fixed data size for a packet is 1024 Bytes. Each experiment is executed for 60 s over 10 runs. We measure flow completion times (FCT) and the average throughput as the performance metrics. FCT shows the average queuing time in the switches, while the average throughput indicates the network congestion situation which reflects the rationality of path allocation in the load balancing scheduling algorithm.

Overheads analysis

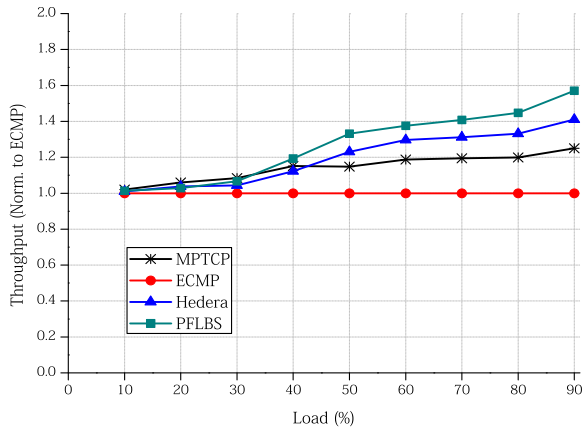
In this subsection, we analyze the expected signaling overhead and probing delay for PFLBS comparing with Hedera and ECMP. Note that ECMP belong to the congestion-unaware distributed mechanisms. Although ECMP does not generate extra signaling overhead, it is a static scheme which does not collect the congestion information from network and leverage these information to balance load. MPTCP is similar to ECMP because MPTCP relies on ECMP or other routing schemes to select paths for sub-flows. In Hedera, the controller pulls the per-flow statistics periodically from each switch. In order to obtain the true rate of a flow for large flow detection, the statistics of each flow must be collected in Fat-tree. Hedera uses OpenFlow to pull statistics for all flows and needs to set up a flow table entry for each flow. So each flow must be sent to the controller before it starts to transmit data. In the 16-port Fat-tree topology, the total number of edge switches is 128 and each edge switch connects to 8 servers. We assume that each server generates 20 new flows per second and the average duration of a flow is 60 seconds. Therefore, the number of flow table entries needed at each edge switch is 8*20*60 = 9600. It is hard for a general

switch with OpenFlow to support so many entries in the flow table because of the limited TCAM. The Hedera controller needs to handle 128*8*20*60 ≈ 1.2 million requests per second because it monitors flows at all edge switches. A single controller can only handle 20-30 K requests per second and multiple controllers are needed to handle the flow setup load. However, it does not seem to be a simple task to implement distribution. Then, we set each statistics packet to 24 Bytes according to [14] and the size of control messages which the controller needs to process is 24*128*8*20*60 ≈ 28.1 MB per second.

In our PFLBS, each server only needs to maintain the flows which are generated by itself and allocate three optimal paths for every flow. The number of flow table entries is merely 3600. The maximum length of a signaling message is set to be 28 Bytes when the path is 6-hop according to the design of shim layer and its per-hop delay is 0.3 μs, constituted by the propagation, transmission and processing delay. Then we calculate the size of signaling messages and delay for the best case and the worse case. When the PA of each flow does not satisfy with the threshold conditions, the server needs to enumerate all the available PAs and select the optimal PA for every flow. The maximum size of signaling messages is 28*64*20*60 ≈ 2.05 MB and the delay is about 16ms. When all the PAs in AT and ATT do not exceed the threshold conditions, the server only probes and updates the PAs in AT and ATT. The minimum size of signaling messages is 28*3*20*60 ≈ 98.4 KB and the delay is about 0.76ms. Therefore, the detection interval for each flow in our PFLBS is an order of magnitude lower comparing with Hedera.



(a) Small Flows ($\leq$ 100KB)



(b) Large Flows ($\geq$ 1MB)

Fig. 7 FCT and throughput statistics for the enterprise workload in Fat-tree (k=16) topology. Note that **a** and **b** are nornalized to the value achieved by ECMP

Workloads

In our simulations, we use synthetic traffic workloads and realistic workloads based on empirically observed traffic models in DCNs respectively to cover most transmission patterns. In the synthetic traffic workloads, the flow size is distributed from 1 KB to 100 MB. We define flows between 1 KB and 100 KB as small flows which are latency-sensitive and the rest as large flows which are throughput-sensitive. The proportion of the number for small flows is 80% but more than 80% of the data bytes belong to large flows. A group of traffic patterns similar to [36] are applied according to the following categories. (1) Stride: each server (s_0, s_1, s_2) sends data to the destination $((s_0 + 1)mod(k\_port), s_1, s_2)$; (2) Random: each server sends data to a random destination not in the same pod with uniform probability. Multiple senders can send to the same receiver; (3) Random bijection: each server sends data to an unique destination which is not in the same pod. Different from the random traffic pattern, each server only receives data from one sender. In the realistic workloads, we consider two types of workloads from production DCNs: an enterprise workload [37] and a data-mining workload [21]. Note that both of the distributions are heavy-tailed which means that a small fraction of the flows contribute most of the data.

Results and evaluation

We firstly discuss the value of the triggers γ^*, δ^* and L^* in the simulations. The statistics show that the link utilization in the DCNs is between 10% and 90%, and we set γ^*=50% as the scheduling threshold for a path. The value of δ^* is determined by the simulation experiments. We test the average network throughput under different values of δ^* for 50% traffic load. Figure 4 illustrates the relationship between the throughput and δ^*. We use the square root

$\sqrt{\delta^*}$ instead of δ^* for the convenience of data plotting. Under this simulation configuration, we can see that when $\sqrt{\delta^*}$ approximates to 16, the throughput reaches a maximal value. Therefore, we set δ^* to be 256. Lastly, we set L^*=100 K, which is the maximum size of the small flows in the experiments. The detection period for PFLBS is set to be 50ms in our experiments.

Figure 5 shows the average throughput of large flows in random, stride and random bijection workloads in synthetic workloads under heavy loads (the sending rate is 1Gbps). As expected, PFLBS achieves the maximum throughput for large flows under all workloads. PFLBS improves upon ECMP by 21-72%, improves upon MPTCP by 17-35% and improves upon Hedera by 12-18%. For the random workload, the throughput of all the schemes decreases because this pattern cannot ensure the traffic is evenly distributed in the network and hotspots may emerge. Figure 6 shows the CDF of the mice flow completion time (FCT) under each workload. The latency of PFLBS is near-optimal for the stride and random bijection workloads because these workloads are non-blocking. PFLBS always chooses a path with the shortest average queue length for a small flow and the 99.9th percentile FCT for PFLBS is within 1.5ms for these workloads. It results in 1.4x, 2.3x and 3.2x lower FCT compared to Hedera, ECMP and MPTCP, respectively. Hedera only schedules large flows when congestion occurs and cannot handle small flows well. MPTCP and ECMP suffer from congestion under heavy loads. MPTCP is worse than ECMP because MPTCP with multiple subflows experiences reordering and timeout in MPTCP. In the random workload, large flows may collide in the downstream links or on the last-hop output port. Therefore the FCT of small flows has a degradation for all the schemes.

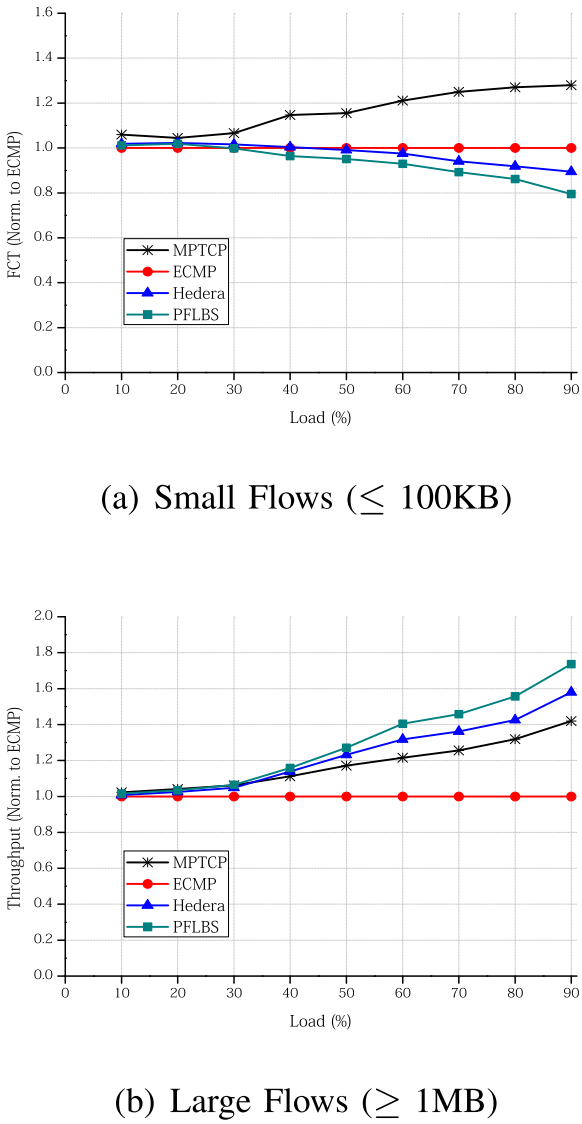


Fig. 8 FCT and throughput statistics for the data-mining workload in Fat-tree

We also evaluate PFLBS under the following two realistic workloads: an enterprise workload and a data-mining workload. Note that both distributions are heavy-tailed. A small fraction of the flows contribute most of the data in DCNs. Particularly, the data-mining workload has fewer small flows than the enterprise workload but a very heavy tail with 95% of all data belonging to 5% of flows that are larger than 10 MB. In the experiments, small flows are defined as flows that are less than 100 KB in size and large flows are defined as flows that are greater than 1 MB. The FCT of small flows and throughput of large flows are normalized to ECMP. Figures 7 and 8 show the results under the two workloads in Fat-tree. Part (a) of the two figures shows the FCT of each scheme for small flows and part (b) shows the throughput of each scheme for large flows.

Under the enterprise workload, we find that MPTCP is noticeably worse than the other schemes for the small flows FCT. It is up to 50% worse than ECMP. The reason is that MPTCP with 8 subflows per connection increases congestion at the edge links because the multiple subflows may cause more burstiness. Meanwhile, the small subflows easily suffer packet reordering and the small window sizes for subflows increases the chance of timeout. Hedera and PFLBS achieve up to 13% better and 32% better than ECMP respectively for small flows. Although Hedera only detects large flows and assigns them to the proper paths, it can balance traffic at high load level and improve FCT for small flows indirectly. PFLBS chooses paths with the shortest queue length for small flows and obtains the best performance. For the throughput of large flows, ECMP is the worst among all the schemes. MPTCP achieves up to 22% better than ECMP and has the best performance under light load due to multiple subflows. Hedera achieves up to 40% better than ECMP but the control overhead

and scheduling interval impact the throughput as the load increases. Moreover, it cannot guarantee that all large flows are accommodated when the load is heavy. PFLBS handles large flows well with limited signaling overhead and the throughput is up to 58% better than ECMP. In the data-mining workload, MPTCP still has the worst FCT for small flows comparing with other schemes. Hedera and PFLBS are 11% better and 20% better than ECMP respectively. For large flows, PFLBS is up to 71% better than ECMP and obtains the best performance. There are some difference between the enterprise and data-mining workloads. Under the enterprise workload, the load is less "heavy" because it has fewer large flows and more small flows. But in the data-mining work load, large flows greater than 35 MB contribute 95% of all bytes. Hence, the data-mining workload is more challenging to handle from a load balancing perspective.

Conclusion

In this paper, we define a port-based source-routing address and propose a port forwarding load-balanced scheduling algorithm for Fat-tree based DCNs. Leveraging the characteristics of PA and the regularity of the Fat-tree topology, an extremely simple routing mechanism is designed. With the multipath selection strategy and large flow scheduling approach, PFLBS can handle small and large flows effectively with low overhead and decrease the switch complexity. The simulation results show that PFLBS reduces the average flow completion time, improves the throughput of large flows and makes full use of available capacity in the network.

Acknowledgments

This work was supported in part by the Joint Project of the National Nature Science Foundation of China under Grant No. U1636109 and the National 863 Program(No.2007AA01Z203).

Authors' contributions

The research presented in this paper is part of the Ph.D. dissertation of the first author under the supervision of the second author. Both authors read and approved the final manuscript.

Authors' information

Zhiyu Liu is currently a PhD candidate of information and communication engineering at Beijing Jiaotong University, Beijing, China. He received his BE degree in optical information science and technology from Jilin University, Changchun, China in 2010 and MS degree in information and communication engineering from Beijing Jiaotong University, Beijing, China in 2014. His area of specialization is computer networks and communication systems, and his current research interests include datacenter networking, network addressing and routing, and complex networks.

Aqun Zhao is currently an associate professor of computer science at Beijing Jiaotong University, Beijing, China, and a visiting professor at Lancaster University, Lancaster, UK. He received his BE and PhD degrees in computer science from Southeast University, Nanjing, China, in 1997 and 2004, respectively. He was a visiting scholar at the University of Victoria, Victoria, Canada between 2016 and 2017. His area of specialization is computer networks and cloud computing, and his current research interests include datacenter networking, network architecture and protocols, and network addressing and routing.

Mangui Liang is currently a professor of information and communication engineering at Beijing Jiaotong University, Beijing, China. He received his BE degree in communication engineering from North China Electric Power University, Baoding, China in 1982. He received his MS and PhD degrees in information and communication engineering from Beijing Jiaotong University, Beijing, China, in 1984 and 1988, respectively. He was a senior visiting scholar at Columbia University, New York, USA in 2008. His area of specialization is computer networks and speech processing, and his current research interests include network architecture and protocols, datacenter networking, and speech processing.

Funding

The Joint Project of the National Nature Science Foundation of China under Grant No. U1636109 and the National 863 Program(No.2007AA01Z203).

Availability of data and materials

Not applicable.

Competing interests

The authors declare that they have no competing interests.

Received: 21 May 2020 Accepted: 6 January 2021
Published online: 08 February 2021

References

1. Cao Z, Kodialam M, Lakshman TV (2016) Joint Static and Dynamic Traffic Scheduling in Data Center Networks. In: IEEE/ACM Transactions on Networking, vol. 24, no. 3. pp 1908–1918. https://doi.org/10.1109/TNET.2015.2434879
2. Quttoum AN (2018) Interconnection Structures, Management and Routing Challenges in Cloud-Service Data Center Networks: A Survey. Int J Interact Mob Technol 12(1):36–60
3. Imran M, Haleem S (2018) Optical Interconnects for Cloud Computing Data Centers: Recent Advances and Future Challenges. In: International Symposium on Grids and Clouds (hold at Academia Sinica in Taipei, Taiwan from 16-23 March 2018)
4. Emara TZ, Huang J (2019) A distributed data management system to support large-scale data analysis. J Syst Softw 148:105–115
5. Bonawitz K, Eichner H, Grieskamp W, Huba D, Ingerman A, vanov V, Kiddon C, Konecny J, Mazzocchi S, McMahan H, Van Overveldt T (2019) Towards Federated Learning at Scale: System Design. arXiv preprint arXiv:01046
6. Ma X, Gao H, Xu H, Bian M (2019) An IoT-based task scheduling optimization scheme considering the deadline and cost-aware scientific workflow for cloud computing. EURASIP J Wirel Commun Netw 249:2019. https://doi.org/10.1186/s13638-019-1557-3
7. Zhu Y, Zhang W, Chen Y, Gao H (2019) A novel approach to workload prediction using attention-based LSTM encoder-decoder network in cloud environment. EURASIP J Wirel Commun Netw 247:2019. https://doi.org/10.1186/s13638-019-1605-z
8. Gao H, Zhang K, Yang J, Wu F, Liu H (2018) Applying improved particle swarm optimization for dynamic service composition focusing on quality of service evaluations under hybrid networks. Int J Distrib Sens Netw(IJDSN) 14(2):1–14
9. Deng S, Xiang Z, Zhao P, Taheri J, Gao H, Yin J, Zomaya A (2020) Dynamical resource allocation in edge for trustable iot systems: a reinforcement learning method. IEEE Trans Ind Inform:974875. https://doi.org/10.1109/TII.2020.2X00000
10. Kuang L, Gong T, OuYang S, Gao H, Deng S (2020) Offloading Decision Methods for Multiple Users with Structured Tasks in Edge Computing for Smart Cities. Futur Gener Comput Syst (FGCS). https://doi.org/10.1016/j.future.2019.12.039
11. Sen S, Shue D, Ihm S, Freedman MJ (2013) Scalable, Optimal Flow Routing in Datacenters via Local Link Balancing. In: Proc ACM CoNEXT. pp 151–162
12. Hopps C (2000) Analysis of an Equal-Cost Multi-Path Algorithm. RFC2992, Internet Engineering Task Force
13. Al-Fares M, Radhakrishnan S, Raghavan B, Huang N, Vahdat A (2010) Hedera: Dynamic How scheduling for data center networks. Symposium on Networked Systems Design and Implementation (hold at San Jose, U.S.A). USENIX
14. Curtis AR, Kim W, Yalagandula P (2011) Mahout: Low-overhead datacenter traffic management using end-host-based elephant

detection. In: 2011 Proceedings IEEE INFOCOM, Shanghai. pp 1629–1637. https://doi.org/10.1109/INFCOM.2011.5934956
15. Perry J, Ousterhout A, Balakrishnan H, Shah D, Fugal H (2014) Fastpass: A Centralized Zero-queue Datacenter Network, SIGCOMM '14 held in Chicago, Illinois. In: Proceedings of the 2014 ACM conference on SIGCOMM. pp 307–318. https://doi.org/10.1145/2619239.2626309
16. Kandula S, Katabi D, Sinha S, Berger A (2007) Dynamic load balancing without packet reordering. SIGCOMM Comp Comm Rev:37
17. He K, Rozner E, Agarwal K, Felter W, Carter J, Akella A (2015) Presto: Edge-based Load Balancing for Fast Datacenter Networks. In: ACM SIGCOMM Computer Communication Review. pp 465–478. https://doi.org/10.1145/2829988.2787507
18. Ghorbani S, Godfrey B, Ganjali Y, Firoozshahian A (2015) Micro Load Balancing in Data Centers with DRILL, HotNets-XIV (hold in Philadelphia, PA). In: Proceedings of the 14th ACM Workshop on Hot Topics in Networks, Article No.: 17. pp 1–7. https://doi.org/10.1145/2834050.2834107
19. Wischik D, Raiciu C, Greenhalgh A, Handley M (2011) Design, implementation and evaluation of congestion control for multipath TCP. Symposium on Networked Systems Design and Implementation (hold at Boston, MA, USA). USENIX
20. Kabbani A, Vamanan B, Hasan J, Duchene F (2014) Flowbender: Flow-level Adaptive Routing for Improved Latency and Throughput in Datacenter Networks. CoNEXT '14 (hold in University of Technology Sydney in Sydney, Australia). In: Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies. pp 149–160. https://doi.org/10.1145/2674005.2674985
21. Greenberg A, Hamilton JR, Jain N, Kandula S, Kim C, Lahiri P, Maltz DA, Patel P, Sengupta S (2009) VL2: a scalable and flexible data center network. ACM SIGCOMM CCR 39(4):51–62
22. Niranjan Mysore R, Pamboris A, Farrington N, Huang N, Miri P, Radhakrishnan S, Subramanya V, Vahdat A (2009) Portland: a scalable fault-tolerant layer 2 data center network fabric. ACM SIGCOMM CCR 39(4):39–50
23. Guo C, Lu G, Li D, Wu H, Zhang X, Shi Y, Tian C, Zhang Y, Lu S (2009) BCube: a high performance, server-centric network architecture for modular data centers. ACM SIGCOMM CCR 39(4):63–74
24. Guo C, Wu H, Tan K, Shi L, Zhang Y, Lu S (2008) DCell: a scalable and fault-tolerant network structure for data centers. ACM SIGCOMM CCR 38(4):75–86
25. Liang M (2006) A method for vector address coding. Chinese patent
26. McKeown N (2009) Software-defined Networking. IEEE INFOCOM Keynote Talk 17(2):30–32
27. Bianco A, Krishnamoorthi V, Li N, Giraudo L (2014) OpenFlow driven ethernet traffic analysis. In: 2014 IEEE International Conference on Communications (ICC), Sydney. pp 3001–3006. https://doi.org/10.1109/ICC.2014.6883781
28. Benson T, Akella A, Maltz DA (2010) Network traffic characteristics of data centers in the wild. In: IMC. Association for Computing Machinery New York NY United States (ACM), Melbourne
29. Kandula S, Sengupta S, Greenberg AG, Patel P, Chaiken R (2009) The nature of data center traffic: Measurements analysis. In: IMC. Association for Computing Machinery, New York, NY, United States (ACM), Chicago
30. Curtis AR, Mogul JC, Tourrilhes J, Yalagandula P, Sharma P, Banerjee S (2011) Devoflow: Scaling flow management for high-performance networks. In: SIGCOMM. Association for Computing Machinery, New York, NY, United States (ACM), Toronto
31. Katta N, Ghag A, Hira M, Keslassy I, Bergman A, Kim C, Rexford J (2017) Clove: Congestion-Aware Load Balancing at the Virtual Edge. In: CoNEXT. Association for Computing Machinery, New York, NY, United States (ACM), Incheon
32. Kandula S, Katabi D, Davie BS, Charny A (2005) Walking the Tightrope: Responsive yet stable traffic engineering. In: SIGCOMM. Association for Computing Machinery, New York, NY, United States (ACM), Philadelphia
33. Wu X, Yang X (2012) DARD: Distributed adaptive routing for datacenter networks. In: ICDCS. IEEE, Macau
34. Lin X-Y, Chung Y-C, Huang T-Y (2004) 18th International Parallel and Distributed Processing Symposium, 2004. Proceedings., Santa Fe. https://doi.org/10.1109/IPDPS.2004.1302913
35. Raiciu C, et al (2011) Improving datacenter performance and robustness with multipath tcp. In: SIGCOMM. Association for Computing Machinery, New York, NY, United States (ACM), Toronto
36. AL-FARES M, LOUKISSAS A, VAHDAT A (2008) A Scalable,Commodity Data Center Network Architecture. In: Proceedings of ACM SIGCOMM. Association for Computing Machinery, New York, NY, United States (ACM), Seattle
37. Alizadeh M, Greenberg A, Maltz DA, Padhye J, Patel P, Prabhakar B, Sengupta S, Sridharan M (2010) Data center tcp (dctcp). ACM SIGCOMM Comput Commun Rev 40(4):63–74

Publisher's Note