Comput Syst Sci & Eng (2018) 4: 251–258
© 2018 CRL Publishing Ltd

International Journal of
Computer Systems
Science & Engineering

OpenFlow Based Dynamic Flow Scheduling with Multipath for Data Center Networks

Haisheng Yu[1], Heng Qi[1], Keqiu Li[1], Jianhui Zhang[1],Peng Xiao[2], Xun Wang[1]

[1]*School of Computer Science and Technology, Dalian University of Technology, Dalian, China*
[2]*School of Information Science & Engineering, Dalian Polytech University, Dalian, China*

The routing mechanism in Data Center networks can affect network performance and latency significantly. Hash-based method, such as ECMP (Equal-Cost Multi-Path), has been widely used in Data Center networks to fulfill the requirement of load balance. However, ECMP statically maps one flow to a path by a hash method, which results in some paths overloaded while others remain underutilized. Some dynamic flow scheduling schemes choose the most underutilized link as the next hop to better utilize the network bandwidth, while these schemes lacks of utilizing the global state of the network. To achieve high bandwidth utilization and low latency, we present a dynamic flow scheduling mechanism based on OpenFlow protocol which enables monitoring the global network information by a centralized controller. Depending on the network statistics obtained by the OpenFlow controller, the routing algorithm chooses the best path for the flow. Because there are two kinds of flows in a Data Center, short-lived flows and long-lived flows, we proposed two different algorithms for them. The implementation uses pox as OpenFlow controller and mininet as the network emulator. The evaluation results demonstrate that our dynamic flow scheduling algorithm is effective and can achieve high link utilization

Keywords: Data Center; Software-defined Networking; ECMP; DLB; FC-DLB

1. INTRODUCTION

Today many private enterprises and universities have deployed their Data Centers to run a variety of interactive cloud-based services and applications with different requirements, such as search engine, multimedia content delivery, social networking, e-commerce, data analysis, scientific computing, and etc.[1-4]. To satisfy these requirements, the Data Centers must provide high performance network to connect tens of thousands of hosts. Many novel topologies, including Fat-Tree [4,5] and VL2 [6], have been proposed to enhance network performance by providing multiple routing paths. These topologies are built up in certain manners with hierarchical and scalable multi-layer structures, and widely adopt many multipath routing algorithms, such as ECMP (Equal-Cost Multi-Path) [7] and VLB (Valiant Load Balancing) [8]. Both ECMP and VLB are static flow scheduling techniques and focus on scheduling the flows in order to improve the overall network bandwidth utilization. In ECMP, it statically maps one flow to one of these multiple paths by hash. VLB is known as randomly balancing to choose the next-hop. Both of them cause the congestion through the oversubscribed links. For example, ECMP may lead to hash collision when mapping a flow to a path, which will cause some links highly overloaded while other links underutilized. These static flow scheduling mechanisms do not take the current network condition into consideration when scheduling the flows.

To the best of our knowledge, the existing advanced protocols for addressing the problem of multiple routing paths: (1)Transport layer solutions such as MPTCP; and (2) ECMP, such as Hedera [9] and Planck [10]; and (3) some dynamic flow scheduling algorithms. In what follows, we will present and analyze these schemes, respectively.

Multipath TCP (MPTCP) [11, 12] is a major modification to TCP that allows multiple paths to be used simultaneously by a single transport connection. Changing TCP to use multiple paths is not a new idea; it was originally proposed more than 15 years ago. MPTCP draws on the experience gathered in previous work, and goes further to solve issues of fairness when competing with regular TCP and deployment issues as a result of middle-boxes in today's Internet. MPTCP can react faster but require widespread adoption and are difficult to enforce in multitenant datacenters where customers often deploy customized VMs [13].

A variety of load balancing schemes aim to address the problems of ECMP. Centralized schemes, such as Hedera and Planck, collect network state and reroute elephant flows when collisions occur. These approaches are fundamentally reactive to congestion and are very coarse-grained due to the large time constraints of their control loops [9] or require extra network infrastructure [10]. In-network reactive distributed load balancing schemes, e.g., CONGA [14] and Juniper VCF [15], can be effective but require specialized networking hardware.

Many algorithms that are designed to satisfy the requirements of high performance and large-scale computing systems [16]. Recently, some dynamic flow scheduling algorithms are proposed to balance the network load when the flows go through the multiple paths on the multi-rooted tree topologies, especially on the Fat-Tree topology[17,18]. However, the flows are routed by the partial network information in these schemes, without considering the global network information. As given in [19] , the path of a flow is an arbitrary choice that it considers the links upward to the core switch rather than the link conditions from the core switch to the destination host.

OpenFlow protocol [20-23] is an innovative networking technology that can efficiently obtain the network statistics from the network devices. It offers a global view of the network, relying on which can dynamically develop a forwarding policy for a flow in the Data Center. That means it is available to dynamically schedule the flows by using the global state of the network and the OpenFlow protocol.

Based on the OpenFlow framework, we present a dynamic flow scheduling mechanism for the flow transmission in the Data Center network to maximize aggregate network utilization—bisection bandwidth—and to do so with minimal scheduler overhead or impact on active flows. In our scheme, the centralized controller will choose an appropriate path for each flow, depending on the flow size and the available bandwidth of the links in a path. Some challenges must be addressed in deploying our mechanism. We collects flow information from OpenFlow Controller, computes non-conflicting paths for flows, and instructs switches to re-route traffic accordingly. First, we must collect the statistics of the network efficiently, because collecting the information will bring additional overhead. Second, when computing the path for a flow, we must consider the different flow sizes. As presented in [6], more than 80% of the flows in a Data Center are short-lived flows with less than 100KB per size while less than 20% long-lived flows take up more than half of the bandwidth, which causes link congestion and starves the short-lived flows. So it is necessary to schedule the long-lived flows and the short-lived flows at the same time.

To address these problem, we implement different dynamic scheduling mechanisms for the short-lived flows and long-lived flows. When a flow comes, the centralized controller calculates a route path for the flow with a default dynamic scheduling algorithm. When a long-lived flow is detected, the centralized controller recalculates its route path by another algorithm designed for the long-lived flows.

Our routing mechanism combines the DLB algorithm for short-lived flows and our proposed algorithm for the long-lived flows. In our routing mechanism, the flows are classified according to their flow size. We call our routing mechanism Flow Classified DLB (FC-DLB). In our routing mechanism, we choose the DLB algorithm as the default routing algorithm, because the DLB algorithm is the same simple as ECMP but better than ECMP. More than that, the DLB algorithm is a load balancing method which is based on the link situation and can reduce the probability of occurrence of link congestion.

The rest of this paper is organized as followed. Section 2 presents the background and related work on multipath routing mechanism in Data Center. Section 3 presents our network architecture and multipath routing algorithm. Section 4 shows the implementation and our evaluation to discusses the results. We conclude the paper in Section 5.

2. BACKGROUD & RELATED WORK

Recently, many researches are focused on Data Center architecture, such as Fat-Tree [3], VL2 [6], Bcube [24], Dcell [25] and etc. All of them are densely interconnected network topologies that take the form of a multi-rooted tree with high speed links. Fat-Tree is built on commodity switches that can increase the aggregate bandwidth. The advantage of the Fat-Tree is that all devices are identical, enabling us to leverage cheap commodity products in the architecture.

Most of the current Data Center use Spanning Tree or ECMP for routing. In Spanning Tree, it is optimized to select a single path for each source-destination pair, like that all traffics traverse in a single tree which results in many unused links. This algorithm does not work well with multi-rooted tree Data Centers. To take better advantage of these topologies characteristics, and make full use of the multiple paths, ECMP is adopted wildly. ECMP randomly routes flows through the available equal cost paths based on the hash value of certain fields of a packet. This kind of traffic distribution in ECMP attempts to balance the load in terms of the number of flows on different paths rather than on the bit rate. As there are different sizes of flows in the Data Center network, ECMP, this uniform distribution in accordance with the number of flows, may lead to congestion on some links, while others remain underutilized.

Hedera [9] is a dynamic flow scheduling mechanism for Data Center Networks which uses multi-rooted tree topology. Hedera routes short-lived flows with ECMP while long-lived flows with a dynamic approach. When detecting a long-lived flow, Hedera estimates the natural demand of the flow, and calculates a suitable path for the flow. The dynamic and central scheduling scheme is only for long-lived flow. Hedera can achieve the goal of maximizing aggregate network utilization by dynamically scheduling the long-lived flows but ignoring the short-lived flows' demand. For example, a short-lived flow may be scheduled on a fully loaded link, and is thus brought excessive delay.

This centralized scheduling is dependent on the state of the global information of the network. In order to obtain global information more efficiently, Hedera implements its scheme on an OpenFlow testbed in which the switches all run OpenFlow protocol. OpenFlow defines a programmable protocol designed to manage and perform per-flow routing through a centralized controller. The controller can add and delete forwarding entries on the switches at fine-gained time scales. Not only to determine how flows traverse the network, but also the controller is able to gather global network view by sending appropriate commands to the switches.

Some other researches have also been presented on flow scheduling by making use of the OpenFlow framework. In [19], an OpenFlow based dynamic load balancing algorithm(DLB) is proposed to split the flows cross the multiple paths in a Fat-Tree topology instead of the traditional ECMP technology. But the routing algorithm only considers the condition of the path upward to the core switch, and ignoring the condition of the path downward to the destination host.

The design of the routing scheme in [26] focuses on re-routing the large flows to other path of the candidates when congestion or link failure happens. The scheme monitors the network and collects statistics from all OpenFlow switches. Based on these statistics, the centralized controller computes all the possible shortest path, and chooses the best path for the flow. The controller also inspects congestion and the link failure periodically in the links. Once the congestion or link failure happens, the controller will re-route the flow to an available path. This is a generally applicable method that is designed for no special Data Center topology, so it does not utilizes the features of multi-root tree topology, such as Fat-Tree.

3. DYNAMIC ROUTING ARCHITECTURE

In this section, we propose the architecture for our routing framework and describe its functional modules. The architecture of our routing framework consists of three components: the monitoring component, a module that monitors the network and obtain the global information of the network; the routing component, a module that calculates the routes for flows depending on the statistics of network collected by the monitoring component; the centralized controller module that aggregates the statistics from the monitoring module and installs the routes onto the network switches. All the functional modules are integrated in the centralized controller.

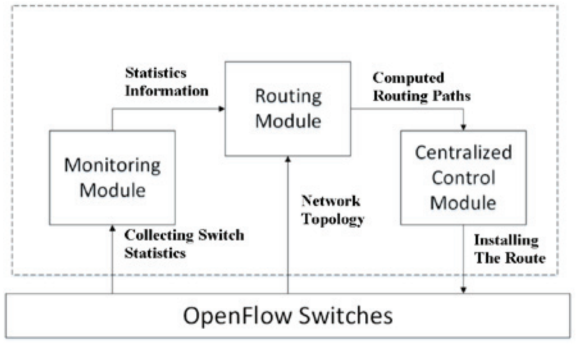

Figure 1 OpenFlow-based dynamic routing control framework.

3.1 Monitoring Module

The monitoring module is designed to collect statistics from all OpenFlow switches, including ports statistics, flows statistics. To obtain the statistics of the ports and specific flows, the monitoring module periodically inquires switches through the OpenFlow API. The switches will respond the monitoring module with bytes counts for the ports and the flows. The monitoring component will store the statistics, and conclude some meaningful results (e.g., the link utilization and the transmission rate of every flow) for routing module using.

To make our routing mechanism feasible, the monitoring module needs to collect two types of statistics: all the physical ports' statistics on every switch and the all the flows' statistics in the network. The physical port statistics include received bytes and transmitted bytes. We use the port statistics to compute the link utilization (a link connects two ports from two switches). A flow's statistics include the flow's transmitted bytes, and according to this, we can decide whether the flow is a long-lived flow. If a flow grows beyond a threshold, 10% of the bandwidth in our implementation, we consider it a long-lived flow.

Both the ports' statistics and the flows' statistics are updated periodically. The monitoring cycle must be decided by the size of the network. If the monitoring cycle is too small, it will bring us too much overhead; if the monitoring cycle is too large, it will not reflect the situation of the network in time.

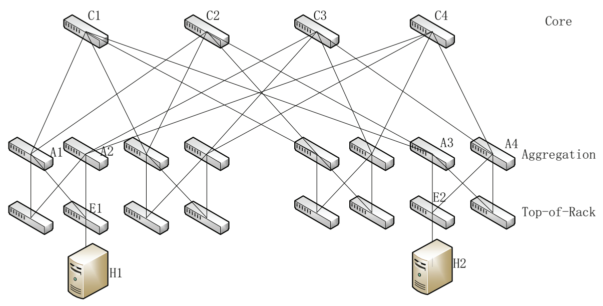

Figure 2 A k=4 Fat-Tree network.

3.2 Routing Module

The routing module is responsible for computing routing paths using the statistics provided by the monitoring module. We design our routing algorithms mainly according to two aspects: the Data Center traffic characteristics and the feature of the Fat-Tree topology. As we all know, Fat-Tree is a 3-layer topology that consisted of many pods. Thus, there will be three different types of flows according to the source and the destination host attached to the switches. Fig 2 shows us a 4-ary Fat-Tree network.

- The source and the destination hosts are connected to the same switch in a pod
- The source and the destination hosts are connected to the different edge switches but in the same pod.
- The source and the destination hosts are connected to the different edge switches in the different pods.

The most important difference among the three types of flows is that when they are transmitted, they will reach different layers that they need access to.

Another characteristic of routing in Fat-Tree topology is that all flows in the network from the source hosts will be transmitted upward until reaching the highest layers and then the path to the destination hosts is deterministic. In other word, when we route a flow, we need to choose a path among all available paths, and it is the same as that we select a switch among all available switches that in the highest layer the flow need to access.

The other aspect considered in the design of our routing mechanisms is the traffic characteristics. There are various sizes of flows in the Data Center traffic. Most of them are short-lived flows and the others are long-lived flows. Different sizes of flows may have different requirements to achieve different goals.

Our routing mechanism is based on the global view of the network and the real-time network statistics, which will bring extra overheads for collecting these statistics. So when we schedule the short-lived flows, we do not expect too many overheads and we can reduce the flows' completion times. We hope that these large numbers of short-lived flows can be scheduled immediately so we implement a local optimal algorithm for these short-lived flows. When scheduling the other long-lived flows, we expect to achieve the goal of avoiding link congestion, so that before we select a path for a long-lived flows, we must know the utilization ratio of every link in that path. Then we choose the most suitable path for that long-lived flow on the basis of the utilization ratio of every link in the paths. But there arises another question that the flow size is unpredictable, how to determine a flow is short-lived flow or a long-lived flow. Until now, the flow sizes are in fact not known a priori. Our solution is that at the beginning, we treat all the flows as the short-lived flows, because of the large number of them. As the flows are transmitted in the network, we monitor the flows' transmitted bytes. Once a flow's transmitted bytes exceed a threshold, the flow will be regarded as a long-lived flow.

When a host sends a flow to another one in the Data Center, the source host will forward the packets of this flow to the edge switch connected to it. When the first packet of this flow arrives at the edge switch, the switch will forward the packet to the centralized controller because of no match with its installed flow entries. After receiving the packet, the centralized controller will let the routing module to handle the packet.

Firstly, the routing module will get the source address and the destination address by analyzing the header field of the packet. Then the routing module will know the highest layer network topology. Secondly, the routing module route the flow using the default algorithm(at the beginning all the flows are routed as short-lived flows). Once a long-lived flow is detected, the other algorithm is triggered, and long-lived flow will be re-routed.

3.3 Centralized Control Module

The centralized control module is designed to connect all the switches in the network. The centralized control is the basic module in the architecture. It is charged with the task of aggregating the information from the monitoring module. It receives the route information from the routing module and install the route on the switches that are needed.

The control module requires two types of information of the network, all the links' utilization and the long-lived flows' statistics. It aggregates all the links' utilization from the monitoring module. The port statistics request is send by the centralized control module to the OpenFlow switches. Depending on these ports statistics, the control module calculates every link's utilization. When the centralized control module receives a request for the links' utilization from the routing module, it will reply to the routing module with the results. The long-lived flows' statistics are also aggregated by the control module. When a long-lived flow is detected by the monitoring module, the control module will store the long-lived flow information and send a re-routing request to the routing module. Then the routing module will re-compute the route for the long-lived flow to find a better path for this flow.

4. DESIGN DECISIONS AND CHALLENGES

In this section, we propose design decisions and challenges of FC-DLB.

4.1 Per-packet Multipathing vs Per-flow Multipathing

Multipathing sets up multiple forwarding paths between source and destination. It allows a given packet to pick a path from the set of available multiple paths. However, this will introduces the problem of determining how to use these multiple paths for forwarding the packets between source-destination pairs efficiently.

Using per-packet multipathing, each packet in a given flow can potentially be sent on a different path independently. Per-packet multipathing is generally more expensive and harder to implement in connection-oriented networks due to packet reordering overhead that can occur at the destination due to differences in the path latencies. On the other hand, when using per-flow multipathing, each flow can potentially be sent on a different path, but all packets from a given flow remain on a single path to preserve packet ordering in the flow. Per-flow multipathing is usually a preferred multipathing scheme, particularly for datacenter networks, because 99% of traffic inside a Data Center uses Transport Control Protocol (TCP), a connection-oriented protocol. ECMP is used mainly on the flow level while VLB can be applied on both the packet and flow levels.

4.2 Reactive vs Proactive Load Balancing

In Reactive Load Balancing, the first packet of flow is sent to controller for processing. Controller evaluates packet and sends flow message to switch insert a flow table entry for the packet. Subsequent packets (in flow) match the entry until idle or hard timeout. Reactive Load Balancing has the problem of source of DOS on controller (packet-in event). Busty behavior can create transient congestion that must be reacted to before switch buffers overflow to prevent loss. This requirement renders most of the centralized reactive schemes ineffective as they are often too slow to react to any but the largest network events, e.g., link failures. Furthermore, centralized schemes can hurt performance when rerouting flows using stale information.

Distributed reactive schemes like MPTCP[11] and CONGA [14] can respond to congestion at faster timescales, but have a high barrier to deployment. Furthermore, distributed reactive schemes must take great care to avoid oscillations. Presto takes a proactive, correct-by-design approach to congestion management. That is, if small, near-uniform portions of traffic are equally balanced over a symmetric network topology, then the load-balancing can remain agnostic to congestion and leave congestion control to the higher layers of the networking stack.

In Proactive Load Balancing, Controllers pre-populate flow elements in the switch before packets arrive, so Proactive flows eliminates any latency induced by consulting a controller on every flow. There are zero set-up time for new flows. Loss of connection between switch and controller does not disrupt network traffic. Proactive Load Balancing Requires smarter approaches than just reacting to network events (global knowledge, discovery, updates and etc.)

To avoid the shortcomings of Proactive Load Balancing and Proactive Load Balancing, we use two different algorithms to route short-lived flows and long-lived flows respectively. We choose the DLB algorithm as the default routing algorithm to route the short-lived flows, and propose a routing algorithm for the long-lived flows.

The DLB algorithm does not consider the link condition of the downward path when routing for a flow, so the path may be not the best one for the long-lived flow. Our routing algorithm will consider all the links' utilization of all the available paths and choose a better path.

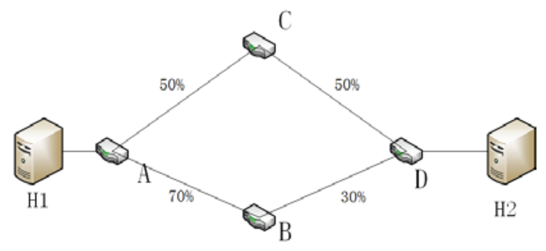

Figure 3 An example to show the drawback of DLB.

Fig 3 shows us an example topology with four switches and two hosts, and a flow is from H1 to H2. The percentages in the Fig 3 are the unutilized bandwidth ratio of every link. According to DLB algorithm, the next hop of A will be B, so the path chosen by DLB is A-B-D. But the unutilized ratio of link B-D is just 30%. The path A-B-D is worse than path A-C-D because path A-C-D can provide a higher bandwidth on the whole path. So the DLB algorithm could miss a better path, sometimes DLB may even find a terrible path, for example, when the link B-D is congested. The main reason is that the DLB algorithm does not consider the whole path's situation.

A path is composed of several links. There must be a link which has the minimal available bandwidth among all the links of a path. We call the link the bottleneck link of the path. The bottleneck link determines the maximum available bandwidth of this path.

When a long-lived flow is detected, the routing module calculates all the paths between the source and the destination host. Then the routing module will compare the available bandwidth of every path's bottleneck link and find the maximum. If the maximum bottleneck link's available bandwidth is greater than the flow's share of the link's bandwidth, the best path for the long-lived flow is found. The routing module updates the flow tables of all the switches in the new path and completes the re-routing of the long-lived flow.

For a long-lived flow f in a k-ary Fat-Tree network, $P = \{p_1, p_2, \cdots, p_k\}$ presents the set of equal cost paths for f. For every path p_i in P, p_i consist of several links, $p_i = \{l_{i1}, l_{i2}, \cdots, l_{in}\}$, $n = 2$ or 4. When $n = 4$, the source host and the destination host of the flow are from different pods; if n=2, the flow's source and destination hosts are in the same pod but different edge switches. The unutilized bandwidth of link l_{ij} is presented as u_{ij}, j=1,2,$\cdots$, n. The unutilized bandwidth of the bottleneck link in p_i is defined as

$$B(p_i) = \min u_{ij},\ j = 1, 2, \cdots, n \qquad (1)$$

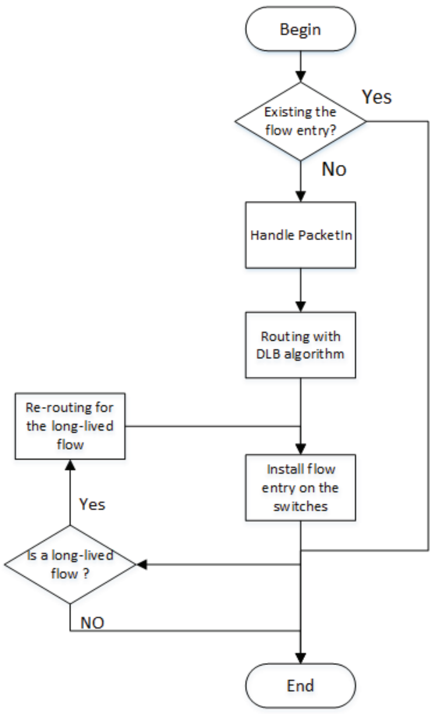
Begin
Existing the flow entry?
Yes
No
Handle PacketIn
Routing with DLB algorithm
Re-routing for the long-lived flow
Install flow entry on the switches
Yes
Is a long-lived flow ?
NO
End

Figure 4 Flow chart of the routing mechanism in the routing module.

B(pi) presents the transmission capacity that path pi could provide. We find the maximum B(pi) among all the equal cost paths in P. If the maximum B(pi) is greater than the current bandwidth the flow f has occupied, the route of flow f will be changed. The flow f will be moved to the path with the maximum B(pi). If the maximum B(pi) is not greater than the current bandwidth and the flow f has occupied, the route of flow f will remain the same.

In our routing mechanism, we choose the DLB algorithm as the default routing algorithm, because the DLB algorithm is as simple as ECMP but better than ECMP. More than that, the DLB algorithm is a load balancing method which is based on the link situation and can reduce the probability of occurrence of link congestion. Our routing mechanism combines the DLB algorithm for short-lived flows and our proposed algorithm for the long-lived flows. In our routing mechanism, the flows are classified according to their flow size. We call our routing mechanism Flow Classified DLB (FC-DLB).

5. IMPLEMENTATION

As shown in Fig 5, We implement our scheduling scheme on mininet, a high-fidelity network emulation platform based on Linux container. The Mininet is running on a single machine with dual quad-core Intel Xeon E5-2603 1.80GHz processors and 32-GB of RAM. We use mininet to model Data Center network behavior by building up a 4-ary Fat-Tree topology. The 4-ary Fat-Tree topology consists of 16 hosts connected by 20 switches. The 4-ary Fat-Tree network is controlled by the Pox OpenFlow controller with the RiplPox extension installed to support ECMP. Our routing algorithm is running at the Pox controller. We also implement ECMP routing algorithm and DLB algorithm in the Pox controller for the purpose of performance comparison with ours.

Current data packet forwarding requires switches in a data network to have matching rules to specify what direction to send an incoming packet, e.g., determining at each switch which port of the switch to transmit the packet through. Software Defined Networking architectures allows these matching rules to be computed and installed from a logically centralized controller. Under OpenFlow protocol, the matching rules are based on 12-tuples (OpenFlow allows matching rules to be installed based on 12 header fields). Matching of field in OpenFlow can be either an explicit match or a wildcard match. A wildcard match means the switch does not care what the value is in some specified field. An explicit match is a binary match, i.e., it matches or it does not. Even though OpenFlow protocol supports installing rules with very fined-grained matching of the packet header fields, the overall flexibility is limited by the available storage in switches.

We use OpenFlow protocol switches to build a FAT-TREE topology Data Center. With OpenFlow protocol, network link information and flow statistics can be collected from OpenFlow controller. We schedule two jobs to collect statistics at fixed period. The first job get traffic information from each port of the switch at every T1 time interval. The traffic information is defined as {swD, portNum, x}, and x means the bytes been transmitted in T1 time, and the bandwidth utilization is calculated by x. The second job get network statistics at every T2 time interval and extract statistical information based on the number of bytes which has been transmitted, so as to determine the size of the up-coming flow.

To reduce short-lived flows' transmission time, our FC-DLB policy can reduce the probability of colliding on multiple equal-cost paths. Following are the steps:

a. When a flow access the TOR switch, the controllers get the source and destination of the flow, and then calculate which layer this flow need to go.

b. Select the appropriate link for the flow from bottom to top. According to the worst-case adaptation algorithm, the link with the lowest bandwidth utilization will be selected as the path for the flow to upwards.

c. Once the data flow reaches the highest level, according to the characteristics of FAT-TREE topology, the path for the flow from the top downing to the destination has been uniquely determined .

d. Add the selected path to the switch's flow table.

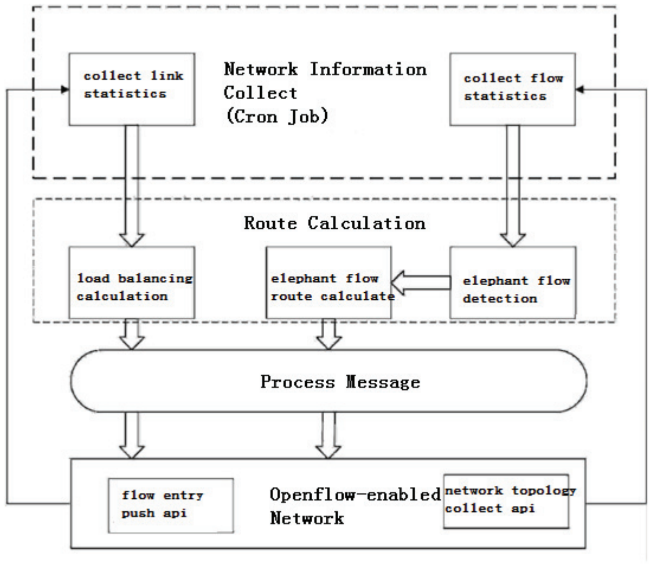

Figure 5 Implementation of FC-DLB.

To reduce long-lived flows' transmission time, we select globally optimal path to reduce the flows on the link to avoid congestion. Following are the steps:

a. When the crontab job notice that the transmitted data of a flow has reached a certain threshold, the flow is defined as a long-lived flow;

b. When a flow is determined to be a long-lived flow, FC-DLB acquires the source and destination of the flow, and then calculate all the paths from the source to the destination;

c. All of the alternate paths are composed by an equal number of links, and each path has a most bandwidth utilized link which is the bottleneck of the path. Based on the principle of max-min fairness, FC-DLB select the lowest bandwidth utilization path to forward the long-lived flow.

d. Delete the original flow table entries of the long-lived flow in the switch, and add the calculated path to the appropriate switches.

In a k=4 fat-tree Data Center network, all switches are OpenFlow-enabled and each link bandwidth is 1000MBPS. As shown in FIG. 2, host H1 send a flow f1 to host H2, then the controller have to calculate the forwarding rule for f1 if there is no corresponding rule to forward f1 in Top-Of-Rack switch E1 which connected to H1. After parsing the address of H1 and H2, the controllers knows that f1 has to arrive the core layer and then go to H2. When f1 arrived E1, the link with most available bandwidth will be selected to transmit f1 until it arrived a core switch.

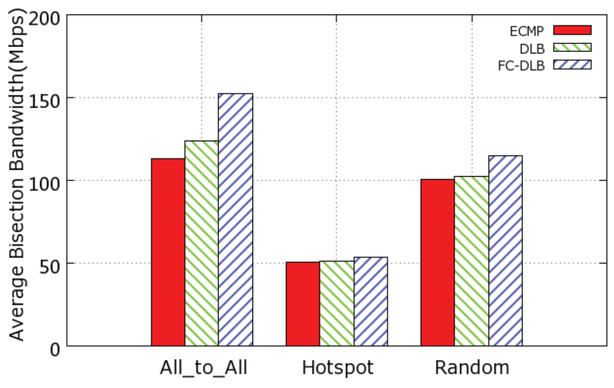

Figure 6 Comparison of ECMP, DLB and FC-DLB for average bisection bandwidth.

We use Iperf to generate TCP flows between the end hosts. To build up the testing environment similar to real ones, we used three types of traffic patterns according to the following styles:

(1) All to all: Every host sends data to every other hosts in the network, so there will be data flows between every two hosts in the network.

(2) Hotspot: A small number of hosts can receive data with high probability compared to other hosts in the network.

(3) Random: A host sends data to other hosts in the network with uniform probability.

Every traffic pattern in our implement is a mix of long-lived and short-lived flows. The portion of the long-lived flows is about 10%–30%.

We use bwm-ng to monitor the all the physical ports on all the OpenFlow switches every second, to observe all the ports' forwarding situation.

6. EVALUATION

In this section, we evaluate the performance of our routing strategy. We have two other algorithms as comparison. One is the ECMP algorithm, the other is the DLB algorithm. Fig 6. shows us the average bisection bandwidth for three communication patterns under three routing algorithms. As we can see, our routing mechanism outperforms ECMP and DLB, especially in the All_to_All pattern. Our FC-DLB keeps the highest average bisection bandwidth among all three communication patterns. DLB perform better than ECMP, but in Hotspot and Random patterns their performance is approximated. Even if in the Hotspot pattern, our FC-DLB also performs better than the other two algorithms. The result indicates that our mechanism performs best in the communication pattern with uniform flow distribution and large number of flows.

Figure 7 shows us the average link utilization ratio for All_to_All communication pattern. Our routing mechanism keeps the highest average link utilization of all three routing algorithms. At the beginning, the DLB and the FC-DLB both can reach 0.6, whereas the ECMP just reaches 0.4, because the DLB and the FC-DLB use the same algorithm before long-lived flows are detected. The average link utilization ratio decreases during the transmitting period because the number of completed flows is increasing. ECMP performs the worst, and the DLB is between the other two algorithms.

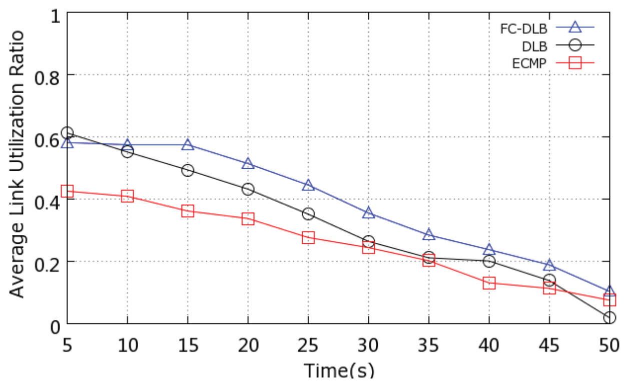

Figure 7 Comparison of ECMP, DLB and FC_DLB for average link utilization.

Figure 8 shows us the traffic load distribution on different pods in a 4-ary Fat-Tree topology network in the All_to_All pattern. FC-DLB and DLB significantly outperform ECMP algorithm. On every pod, our FC-DLB achieve the highest traffic load, and the ECMP achieve the lowest, and DLB is between them.

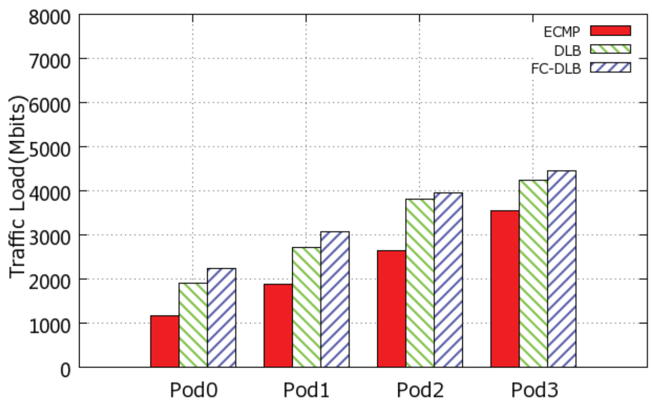

Figure 8 Comparison of three algorithm in term of traffic load on every pod in a k=4 fat-tree network.

7. CONCLUSION

In this paper, we propose an OpenFlow based dynamic flow scheduling mechanism FC-DLB for Data Center network with Fat-Tree topology. With FC-DLB, flows in a Data Center can be dynamically controlled and the forwarding strategies of the streams can be dynamically adjusted according to the size of the flow.

The routing scheme is made depending on the global view of the network obtained by a centralized OpenFlow controller. The scheduling mechanism uses two different algorithms to route short-lived flows and long-lived flows respectively. We choose the DLB algorithm as the default routing algorithm to route the short-lived flows, and propose a routing algorithm for the long-lived flows. We dynamically change the long-lived flows' route, to move long-lived flows from highly-utilized path to less utilized path. Our routing mechanism can improve the link utilization and make more use of the bandwidth than ECMP and the DLB algorithm.

To validate FC-DLB, we implement a prototype which has three modules, such as monitoring module (include two sub modules, network information acquisition module, and long-lived flow detecting module), routing module and centralized control module. Extensive simulation experiments have been conducted to evaluate the efficiency of the proposed FC-DLB scheme. The results manifest that this scheme considerably outperforms the state-of-the-art scheme in terms of bisection bandwidth and traffic load, average link utilization.

ACKNOWLEDGMENT

This work is supported by the State Key Program of National Natural Science of China(Grant No. 61432002), NSFC Grant Nos. 61772112 and 61672379, and the Dalian High-level Talent Innovation Program (No. 2015R049).

REFERENCES

1. M Wang, H Zhou, J Chen. Achieving a scalable and secure software defined network by identifiers separating and mapping[J]. Computer Systems Science And Engineering, 2017, 32(2): 159–169.
2. J Hajlaoui, M Omri, D Benslimane. A QoS-aware approach for discovering and selecting configurable IaaS Cloud services[J]. Computer Systems Science And Engineering, 2017, 32(4).
3. M. Al-Fares, A. Loukissas, and A. Vahdat. A Scalable, Commodity DataCenter Network Architecture. ACM SIGCOMM, 2008.
4. T. Benson, A. Anand, A. Akella, and M. Zhang. Understanding Datacenter Traffic Characteristics. SIGCOMM WREN workshop, 2009.
5. C. E. Leiserson. Fat-trees: Universal networks for hardware-efficient supercomputing. IEEE Transactions on Computers, 1985.
6. A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A.Maltz, P. Patel, and S. Sengupta. VL2: a scalable and flexible Data Center network. In SIGCOMM, 2009.
7. HOPPS, C. Analysis of an Equal-Cost Multi-Path Algorithm. RFC 2992, IETF, 2000.
8. W. J. Dally and B. Towles. Principles and Practices of Interconnection Networks. Morgan Kaufmann Publisher, 2004.
9. M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic flow scheduling for Data Center networks. In NSDI 2010.
10. J. Rasley, B. Stephens, C. Dixon, E. Rozner, W. Felter, K. Agarwal, J. Carter, and R. Fonseca. Planck: Millisecond-scale Monitoring and Control for Commodity Networks. In SIGCOMM, 2014.
11. A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses": http://tools.ietf.org/html/ draft-ietf-mptcp-multiaddressed-09.
12. C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O. Bonaventure, and M. Handley, "How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP," USENIX Symposium of Networked Systems Design and Implementation (NSDI'12), San Jose (CA), 2012.
13. D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley. Design, Implementation and Evaluation of Congestion Control for Multipath TCP. In NSDI, 2011.
14. M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, F. Matus, R. Pan, N. Yadav, G. Varghese, et al. CONGA: Distributed Congestion-aware Load Balancing for Datacenters. In SIGCOMM, 2014.
15. D. R. Hanks. Juniper QFX5100 Series: A Comprehensive Guide to Building Next-Generation Networks. "O'Reilly Media, Inc.", 2014.
16. J Wu. Energy efficient dual execution mode scheduling for real-time tasks with shared resources[J]. Computer Systems Science And Engineering, 2016, 31(3): 239–253.
17. R.Wang, D.Butnariu, J.Rexford. OpenFlow-Based Server Load Balancing Gone Wild. Hot ICE, 2011.
18. He K, Rozner E, Agarwal K, et al. Presto: Edge-based load balancing for fast datacenter networks[J]. ACM SIGCOMM Computer Communication Review, 2015, 45(4): 465–478.
19. Y.Li, P.Deng. OpenFlow based load balancing for Fat-Tree networks with multipath support. Proc. 12th IEEE International Conference on Communications, 2013.
20. N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J.Rexford, S. Shenker, and J. Turner. OpenFlow: Enabling Innovation in Campus Networks. ACM SIGCOMM CCR, 2008.
21. Pox OpenFlow Controller. http://www.noxrepo.org/pox/documentation/.
22. B. Lantz, B. Heller, and N. McKeown. A Network in a Laptop: Rapid Prototyping for Software-Definded Networks. ACM SIGCOMM, 2010.
23. OpenFlow Switch Specification, Version 1.0.0. http://www.OpenFlow.org/documents/OpenFlow-spec-v1.0.0.pdf.
24. C Guo, G Lu, D Li, et al. BCube: a high performance, server-centric network architecture for modular Data Centers[J]. ACM SIGCOMM Computer Communication Review, 2009, 39(4): 63–74.
25. C. Guo, H. Wu, K. Tan, L. Shi, et al. Dcell: a scalable and fault-tolerant network structure for Data Centers. SIGCOMM, 2008.
26. R.Kanagavelu, et al. OpenFlow based control for re-routing with differentiated flows in Data Center Networks. Networks (ICON), 2012 18th IEEE International Conference on. IEEE, 2012.