An Improved Adaptive Scheduling Strategy Utilizing Simulated Annealing Genetic Algorithm for Data Center Networks

Wentao Wang[1], Lingxia Wang[1], Fang Zheng[1]
[1] College of Computer Science, South-Central University for Nationalities
Wuhan 430074, China
[E-mail: 30307095@qq.com; wanglingxiawlx@163.com; 294081300@qq.com]

Received March 12, 2017; revised June 26, 2017; accepted July 17, 2017; published November 30, 2017

Abstract

Data center networks provide critical bandwidth for the continuous growth of cloud computing, multimedia storage, data analysis and other businesses. The problem of low link bandwidth utilization in data center network is gradually addressed in more hot fields. However, the current scheduling strategies applied in data center network do not adapt to the real-time dynamic change of the traffic in the network. Thus, they fail to distribute resources due to the lack of intelligent management. In this paper, we present an improved adaptive traffic scheduling strategy utilizing the simulated annealing genetic algorithm (SAGA). Inspired by the idea of software defined network, when a flow arrives, our strategy changes the bandwidth demand dynamically to filter out the flow. Then, SAGA distributes the path for the flow by considering the scheduling of the different pods as well as the same pod . It is implemented through software defined network technology. Simulation results show that the bisection bandwidth of our strategy is higher than state-of-the-art mechanisms.

Keywords: scheduling; adaptive methods; simulated annealing genetic algorithm; software defined network; data center

The research presented in this paper is supported by the following projects "the National Committee for reform of the National People's Committee", ID:15013

https://doi.org/10.3837/tiis.2017.11.004 **ISSN : 1976-7277**

1. Introduction

In recent years, Internet online business (e.g., searching, transacting, contacting) has been growing rapidly. In order to meet the increasing transaction need, cloud computing has been widely concerned in industry and academy[1]. In the infrastructure of cloud computing, a data center network plays a significant role in the interconnect dedicated links and switches. However, as businesses continue to extend, how to allocate the proper bandwidth demand presents a special challenge.

The development of the business and the use of new technologies have brought new challenges to the data center network. Especially, flow scheduling is very significant in data centers, which aim at providing enough bisection bandwidth for popular applications.

There have been flow scheduling approaches in the last years. They are generally classified into centralized traffic scheduling and distributed traffic scheduling. Hedera[2], Ashman-BestFit, Ashman-ProFit[3], MicroTE[4], FreeWay[5] and a Coarse-grained Scheduling[6] are popular centralized traffic scheduling. They intend to rely on some controllers to monitor the path allocation as a passive way. On the other hand, VLB (Valiant Load Balancing)[7], DRAD[8] and DiFS[9] are popular distributed traffic scheduling. They do not need a controller to schedule some flows as an active way.

Though the existing flow scheduling approaches can promote more bisection bandwidth, they not only create path conflicts, but also are lack of intelligent management. Since the flow can not be scheduled according to the self-demand, they fail to take full advantage of the link resource to achieve high bisection bandwidth.

The rise of SDN provides a new idea of solving the problem of the data center network. SDN is an innovative network architecture and stems from Stanford University in the United States in 2006 clean slate research project. Especially, its core is the separation of the control plane and the traditional distributed network devices in order to realize the centralized control of them. The centralized control can obtained by some similar network operating systems. Naturally, the centralized control provides flexible developing and programming interfaces while the network device is only responsible for simple data forwarding[10-12]. When the controller starts, OpenFlow[13-15] switches try to open a secure channel to the controller. After some switches are connected to the controller, the controller can start to manage different modules. As a result, it enables us to perform query, insert and modification about some flow entries. OpenFlow protocol includes some special features that can be leveraged to realize our idea. It is worth noting that the controller has many functions using a variety of listener events to improve the performance of our experiments. Therefore, compared with some traditional scheduling ways, the genetic algorithm is a good choice by combining different algorithms.

The genetic algorithm is a kind of global search algorithms. It can use some operations to get different species. And an adaptive scheduling algorithm [16] is used to cope with the problem of multimedia traffic. We can deal with the problem of the flow scheduling by a dynamic scheduling mechanism with SAGA. In this paper, we improve the scheduling mechanism by updating the bandwidth demand and recovering the path bandwidth. And we propose SAGA (simulated annealing genetic algorithm) taking full advantage of the link sources. It considers the communication in the different pods as well as the same pod.

our contribution includes:

1. We improve a choosing flow strategy to filters out flows in need for scheduling based on the dynamic demand in real time t to lower the load of the controller.
2. SAGA considers the scheduling of the different pod as well as the same pod based on the changing demand to facilitate the bandwidth resources of the network link and improves the network bandwidth utilization.

This paper is organized as follows. We overview the related work in Section 2. Section 3 introduces the background about the scheduling strategy. We address the SAGA and adaptive scheduling strategy in Section 4. Section 5 describes the implementation and evaluation. Finally, we conclude this paper in Section 6.

2. Related work

There have been recent efforts for addressing various flow scheduling methods to improve the bisection bandwidth. In all, all research is classified into two categories: centralized traffic scheduling and distributed traffic scheduling.

2.1 Centralized Traffic Scheduling

The traditional way adopts ECMP (Equal Cost Multipath). It statically strips flows across available paths and then produces some path conflicts. Those conflicts will result in serious congestion. In order to deal with the above problem, hedera[2] is presented by designing GFF (Global First Fit) algorithm and SA (simulated annealing) algorithm.

GFF is a kind of greedy strategy, which can find the first path to satisfy the demand of flow bandwidth by searching each flow. Though GFF takes the remaining path bandwidth into account, it fails to consider the already changing demand due to the effect of the added flow on the previous flow. SA is a global heuristic algorithm. On one hand, the key insight of SA is to assign a single core switch for each destination host. This only considers the communication in different pods. However, the communication in the same pod also exists. On the other hand, the link is regarded as an idle state in each SA scheduling period, and then the global scheduling of all flows is performed, which makes frequent replacement of the path. Unfortunately, the scheduled path may be in the non-idle state which will cause the flow conflict. And the frequent path replacement will also increase the controller load.

Hedera remains more fragments in allocating bandwidths. In order to take advantage of fragments, Ashman-BestFit and Ashman-ProFit are raised gradually. Ashman-BestFit allocates the path to the flow with the closest to the bandwidth demand. But it is possible to cause many flows to be scheduled on the same link, which leads to the frequent usage of the partial links while the other idle. Ashman-ProFit sets up the allocation probability for each candidate path. If the path bandwidth is closer to the bandwidth demand, the more likely the path will be allocated to the flow.

A fine-grained flow engineering scheme based on MicroTE is proposed with monitoring module, a network controller and a routing module to reduce the workload of network equipment.

2.2 Distributed Traffic Scheduling

VLB (Valiant Load Balancing) [7] is adopted to balance the load. Each server randomly

selects each flow to a relay switch, and then selects a path to the destination host. It is a static way of flow scheduling. To deal with the scalability of centralized traffic scheduling, DRAD [8] allocates many multiple level IP for terminal hosts to forward data packets. Each switch on all paths is necessary to be monitored to get the state of links. But DRAD has the upgrading of the protocol and the complexity of monitoring. So DiFS[9] is proposed, which does not ask for centralized control and not allow to split a flow. Therefore, it will avoid the disorder of the packet. But there are the local and remote flow conflict in DiFS. The local flow is in the same pod while the remote flow in the different pod. DiFS needs to avoid these two types of conflicts in respect and weighs the traffic load. On the whole, the way of avoiding the local conflict is simple. Each switch uses PAA (Path Allocation Algorithm) to send evenly traffic to all the ports in order to avoid the local conflict. However, the way of avoiding the remote conflict is relatively complex. At first, IDA (Imbalance Detection Algorithm) is used to detect some links that are connected with the switch. After a collision is detected, the switch will send an EAR (Explicit Adaption Request) message to change the switch path. When the EAR message is received, EAA (Explicit Adaption Algorithm) is carried out to avoid remote flow conflict. Those above distributes traffic scheduling methods add more loads to the controller.

Even though the existing scheduling methods can deal with the shortage of multipath, they are lack of intelligent management and adaptive scheduling.

3. Background

In this section, we introduce the topology, ECMP, traffic monitoring, bandwidth demand estimation and bandwidth resource recovery. The topology is simulated with popular fat-tree in data centers, from which the controller can collect detailed information. ECMP is called to bring the basic state of the flow as the start of the system. The main purpose of the traffic monitoring is to filter out the elephant flow for scheduling. Bandwidth demand estimation is aimed at estimating the demand as a threshold. Bandwidth resource recovery saves resources to make full use of the bandwidth. Each is described in detail below.

3.1 Topology

The traditional topology is improved in switches and servers. Some popular topologies are Fat-Tree[17], ElasticTree[18], VL2[7], PortLand[19], Jellyfish[20], Helios[21], c-Through[22], OSA[23] with a large number of switches. And some popular topologies are DCell[24], BCube[25], FiConn[26], MDCube[27], PCube[28] with a large number of servers . In this paper, we adopt the fat-tree.

The traditional tree structure is a hierarchical network topology, which is composed of a core layer, an aggregation layer and an access layer. In general, the core layer and the aggregation layer adopt some devices with good performance. The switch in the access layer provides some ports with 1Gbps downlink and 10Gbps uplink. Those downlinks are usually connected with the data center server while uplink with the aggregation layer. The aggregation layer switch is connected with the core layer with 10Gbps downlink and uplink. The traditional tree has the advantages of a simple structure and easy operation. However, this topology is only suitable for a small data center network.

Now, due to multipath between a source host and a destination host, Fat-Tree is a good choice of simulating the topology of data center in a variety of researches. The Fat-Tree includes above three layers with some core switches and pods. A pod includes aggregation switches and edge switches. Given k-port, the number of core switches is $(k/2)^2$, and the

number of pod is *k*. *k*/2 edge switches and *k*/2 aggregation switches form a pod. It is worth noting that a core switch is determined given a destination host.

3.2 ECMP

In the traditional data center network, the switch runs ECMP protocol with five tuples to get the output port. It is a static scheduling way. However, the process of ECMP in the SDN controller is not different from the traditional network. When a flow arrives in a switch without the corresponding flow entry in the flow table, the controller is consequently informed via the packet_in message. Furthermore, the controller gets the five tuples of the flow by processing the packet_in message, and then uses the cyclic redundancy check(CRC32) hash function to get the hash value. Eventually, a path is entirely selected from multiple equivalent paths in the light of the corresponding hash value. Instantly, the flow table information is sent to the switch.

3.3 Traffic Monitoring

As we know, collecting traffic is of great significance for flow scheduling. Fortunately, OpenFlow protocol has many messages of getting relevant information such as Read-State message. In a data center network, we just collect the traffic passing the edge switch. When the collecting traffic timer is triggered, the controller sends the OFPT_STATS_REQUEST message to some edge switches. And then those switches send OFPT_STATS_REQUEST_RECEIED message to the controller in response. Then, the controller analyzes the statistical information to obtain the byte size and duration time of a flow from FlowStatsReceived event. The true transmission rate is calculated according to the Formula(1)[13]. Byte_count means the total byte number of a flow. The numerator is defined as the alive time of a flow in OpenFlow.

$$rate = \frac{(8 \times byte\_count)/(1 << 20)}{(duration\_\sec \times 10^9 + duration\_n\sec)/10^9} \tag{1}$$

The rate tends to filter out the elephant flow (bandwidth greater than 10% link capacity) in the network to determine whether the flow is a newly detected flow or not. Because some elephant flows are more likely to cause the problem of path conflict. In view of the efficiency of traffic monitoring , it has been used in a variety of applications.

3.4 Bandwidth Demand Estimation

Estimating the natural bandwidth demand of the current network is an essential part of the system architecture. We use bandwidth demand estimation [2]. A TCP flow's current sending rate is not related to its natural bandwidth demand in an ideal non-blocking network. Thus, in order to make an intelligent flow placement decision, we need to know the max-min fair bandwidth allocation as if they are limited by the sender or receiver NIC. When a network limited, a sender will try to distribute its available bandwidth fairly among all its outgoing flows. Below, there is an example to explain the detailed calculation process in **Fig. 1** .

Suppose there are four hosts in the current network (H0, H1, H2, and H3). H0 sends a flow to H1, H2 and H3 respectively . H1 sends two flows to H0 and a flow to H2; H2 sends a flow to H1 and H3 respectively. H3 sends two flows to H1. According to the Algorithm 1, at first it uses EST-SRC(h) to enlarge the demand with the same source node named the first demand matrix, then EST-DST(h) traverses all flows to reduce the same destination node bandwidth

demand named the second demand matrix. Thus, the bandwidth demand of all flows are not changed until the demand matrix are convergent.

Algorithm 1: bandwidth demand estimation

```
Estimate-Demands()
1   for all src in M
2     i, j ← 0
3   do
4     for all host h in H do EST-SRC(h)
5     for all host h in H do EST-DST(h)
6   while some M i, j .demand changed
7   return M
EST-DST(h.dst)
1   dT， dS， nR ← 0
2   for all flow f in F do
3     f.rl ← true
4     dT ←dT + f.demand
5     nR ←nR + 1
6   if dT ≤ 1.0 then
7     return
8   eS ← 1.0/nR
9   do
10    nR ← 0
11    for all flow f in F do and f.rl do
12      if f.demand < eS then
13        dS ←dS + f.demand
14        f.rl ← false
15      else
16        nR ←nR + 1
17    eS ← (1.0−dS)/nR
18  while some f.rl was set to false
19  for all flow f in F do
20    Mf.src,f.dst.demand ←eS
21    Mf.src,f.dst.converged ← true
```

$$\begin{bmatrix} & (1/3)_1 & (1/3)_1 & (1/3)_1 \\ (1/3)_2 & & (1/3)_1 & 0_0 \\ (1/2)_1 & 0_0 & & (1/2)_1 \\ 0_0 & (1/2)_2 & 0_0 & \end{bmatrix} \Rightarrow \begin{bmatrix} & [1/3]_1 & (1/3)_1 & (1/3)_1 \\ [1/3]_2 & & (1/3)_1 & 0_0 \\ [1/3]_1 & 0_0 & & (1/2)_1 \\ 0_0 & [1/3]_2 & 0_0 & \end{bmatrix} \Rightarrow \begin{bmatrix} & [1/3]_1 & (1/3)_1 & (1/3)_1 \\ [1/3]_2 & & (1/3)_1 & 0_0 \\ [1/3]_1 & 0_0 & & (2/3)_1 \\ 0_0 & [1/3]_2 & 0_0 & \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} & [1/3]_1 & (1/3)_1 & [1/3]_1 \\ [1/3]_2 & & (1/3)_1 & 0_0 \\ [1/3]_1 & 0_0 & & [2/3]_1 \\ 0_0 & [1/3]_2 & 0_0 & \end{bmatrix}$$

Fig. 1. An example of bandwidth estimation demand

3.5 Bandwidth resource recovery

According to the residual bandwidth information of each link, the scheduling algorithm allocates a path for a flow in demand. When a flow completes, there exists the problem of cyclic bandwidth usage. In this paper, we try to leverage the idle_timeout mechanism of OpenFlow protocol and flow-removed message to recover the bandwidth on the link. When the duration time of a flow is beyond the idle_timeout, the flow entry is accordingly deleted in the flow table. Besides, the value of the idle_timeout can be set by the controller when the information is sent to the table. When the flow table is deleted, the switch sends the FlowRemoved message to the controller. At this point, we need to set the flag field for 1` in the flow table. After receiving the FlowRemoved message, the controller can obtain the assigned path and bandwidth of the flow. As a result, the bandwidth of all the links on the path will be recovered.

4. Methodology

We present our on-demand adaptive traffic scheduling mechanism in data center networks. The network adopts fat-tree by interconnecting the OpenFlow Switch and SDN controller with ECMP. The on-demand adaptive traffic scheduling and link bandwidth recovery can achieve network traffic and resource management, as shown in **Fig. 2.** We address SAGA and the on-demand traffic adaptive scheduling strategy in detail in this section.

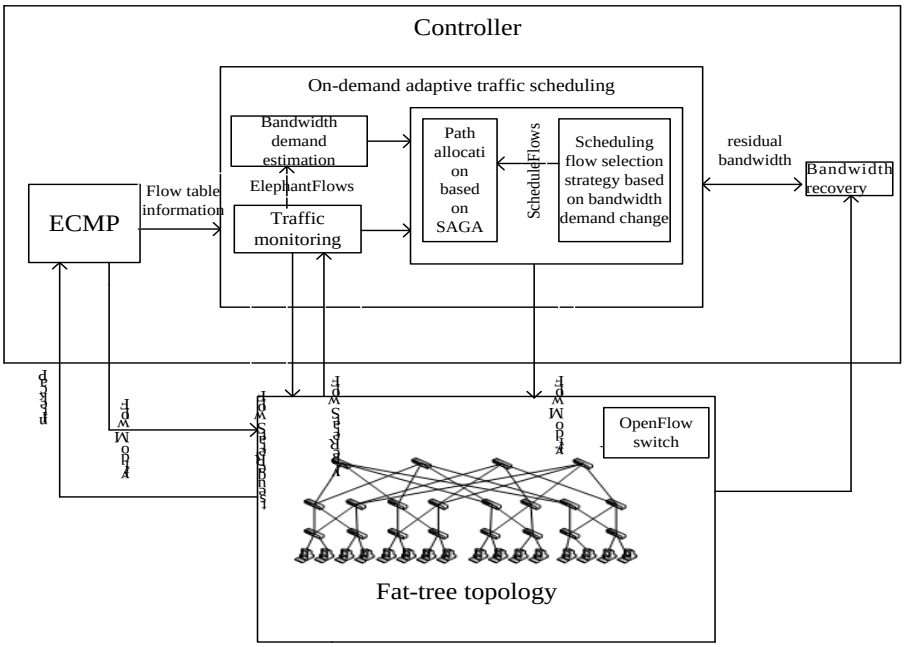

Fig. 2. System Architecture

4.1 SAGA

SAGA is a combination of the genetic algorithm and the simulated annealing algorithm to enhance the ability of searching. As we know, the genetic algorithm has strong global search ability, but it is prone to premature convergence. While the simulated annealing algorithm uses the metropolis criterion to accept the inferior solution with a certain probability, it

effectively overcomes the premature convergence of genetic algorithm. In this paper, with the purpose of maximizing link bandwidth resource utilization, SAGA is used to search the global scheduling path in the network. It adopts the mapping between the host and the highest level switch. In a general network, the path searching without exceeding any link capacity is a multi-commodity flow problem, which is a NP hard problem. A scheduling algorithm based on SAGA is proposed to obtain the approximate optimal solution of the flow path. The model and some steps of SAGA will be described in the following.

4.1.1 SAGA Model

The SAGA model is described as follows: the data center network topology is defined as $G(S,L)$,where S presents a set of switches, $S=\{s_1,s_2,\cdots s_n\}$; L is a set of links, described as $L=\{l_1,l_2,\cdots,l_l\}$. Then we set $C=\{c_{l_1},c_{l_2},\cdots,c_{l_l}\}$ to represent the capacity of the link and γ($\gamma=\{\gamma_{l_1},\gamma_{l_2},\cdots\gamma_{l_l}\}$) is a set of residual bandwidths in each link.

Suppose that $F=\{(f_1,d_1),(f_2,d_2),\cdots,(f_k,d_k)\}$ is a set of flows needing to be scheduled, where d_i is the bandwidth demand of $f_{i.;}$ $P=\{(p_1,bw_1),(p_2,bw_2),\cdots,(p_k,bw_k)\}$ is a set of each flow allocation path. If $f_{i.}$ is allocated to p_i($p_i=\{l_{i1},l_{i2},\cdots,l_{ip}\}$), bw_i is the residual bandwidth of $p_{i\,,}$ which is present as Formula (2)[2].

$$bw_i=\begin{cases} d_i & if\ \alpha_{p_i}\geq d_i \\ \alpha_{p_i} & if\ \alpha_{p_i}<d_i \end{cases} \tag{2}$$

where a_{pi} is the minimal bandwidth of $p_{i..}$
The total link bandwidth utilization of all flows is described as Formula (3):

$$U_F=\frac{\sum_{p_i=p_1}^{p_k}\sum_{l_{ij}=l_{i1}}^{l_{ip}}bw_i}{\sum_{l_q=l_1}^{l_m}c_{l_i}} \tag{3}$$

The numerator represents the sum of the reserved bandwidth of each link on all paths. The denominator represents the sum of the capacity of each link on all flow scheduling paths. K represents the number of flows that the network needs to schedule. P is the number of links on the flow to be allocated path. M represents the sum of all links.

The purpose of the algorithm model is to maximize the bandwidth utilization, so it is defined as Formula(4). But the reserved bandwidth in the scheduling path does not exceed the natural bandwidth demand of the flow. The sum of the reserved bandwidth in each link does not exceed the capacity of the link. The sum of reserved bandwidth in all scheduling paths does not exceed the capacity of all links.

$$\begin{aligned} &maximize \quad U_F \\ &s.t., \quad bw_i\leq d_i\ ; \\ &\sum_{l_i}bw\leq c_{l_i}\ ; \\ &\sum_{p_i=p_1}^{p_k}\sum_{l_{ij}=l_{i1}}^{l_{ip}}bw_i\leq\sum_{l_q=l_1}^{l_m}c_{l_i} \end{aligned} \tag{4}$$

The ultimate goal of flow scheduling is to utilize the reasonable network resources and improve the bandwidth utilization of the network. Therefore, we use U_F as the fitness function in SAGA.

4.1.2 Chromosome encoding

Fat-Tree provides an equivalent path for a pair of source and destination hosts in a network. The flow between two hosts connecting the same edge switch directly forwards from the edge switches. At the meantime, a flow in the same pod without connecting the same edge switch forwards from the corresponding aggregation switch. Other flows in the different pods need to be forwarded by the core switch. The same flow chooses different aggregation switches or core switches, which results in the corresponding different paths. Therefore, the genetic algorithm will take the mapping of a host and an aggregation switch or core switch to locate the downlink of the flow. Since the flow with the host reaching the same destination is assigned to the corresponding aggregation switches or core switches, the genetic algorithm search space is greatly reduced.

$$\begin{cases} CoreSwitch\_ID = DstHost\_ID \bmod \left((k/2)^2 \right) \\ AggSwitch\_ID = DstHost\_ID \bmod (k/2) \end{cases} \tag{5}$$

For a Fat-Tree level K topology with a total of $(k/2)^2$ core switches, a core switch id (*CoreSwitch_ID*) is limited in between 1 and k/2 as well as the aggregation switch id (*AggSwitch_ID*) in {1,2,... k/2}. The size of the chromosome is equal to the total number of the destination host, which is determined by the host number of the current network flow. The value in chromosome gene is the id of the destination host(*DstHost_ID*). The destination host is mapped to the highest level switch of the scheduling path (the core switch or the aggregation switch). So we adopt the mapping according to Formula(5). For instance, when *k* is 4, the chromosome {3,5,4,3,2} indicates the total number of hosts is 5. The flow with destination host (ID=3) in an inter-pod is assigned to the core switch (ID=3), whereas it is assigned to the aggregation switches (ID=1) in an intra-pod.

4.1.3 Genetic Operations

Genetic operations consist of a selection operation, a mutation operation and a crossover operation. The selection operation is selected from the original group according to a given rule. In this paper, it adopts the biased roulette wheel to select. According to the fitness value, it selects the parent individual. The selection probability is shown in the Formula (5) . f_i is defined as the fitness value of an individual.

$$p_i = \frac{f_i}{\sum_{i=1}^{M} f_i} \tag{5}$$

The crossover operation may produce some new chromosomes by exchanging and recombining. As a result, it will come into being new individuals. In this paper, a single point crossover is used to select an individual in their neighbor at random. Then, the mapping between a destination host and a highest switch is naturally changed. In other words, the corresponding path of a flow is automatically changed.

The mutation operation selects two chromosomes at random. Then the genes of the chromosome are exchanged in some probability.

4.1.4 Simulated Annealing

The metropolis criteria of SA are used to determine whether to accept a new individual or not. Metropolis criteria are shown in the Formula (6).

$$P(f_n, f, T) = \begin{cases} 1 & if\ f_n \geq f \\ \exp\left(\frac{f - f_n}{KT}\right) & if\ f_n < f \end{cases} \quad (6)$$

Formula (6) represents the acceptable probability of an individual. f means the fitness value of an old individual while f_n a new individual. If f_n is greater than or equal to f, the old individual will be replaced with a new individual where the acceptable probability is 1. Otherwise, the new individual is generally accepted in some probability.

SAGA includes:

① Initialize control parameters (the species size, initial temperature, crossover probability and mutation probability) .

② Encode the chromosomes according to the destination host.

③ Get the fitness value of each individual according to the initial path.

④ Carry out the genetic operation and get new species.

⑤ Select an individual according to the metropolis criteria .

⑥ Update the corresponding path and the demand.

It is depicted in **Fig. 3** in detail. Assuming that the number of iteration is n and the size of species is f , its time complexity is O(n*f).

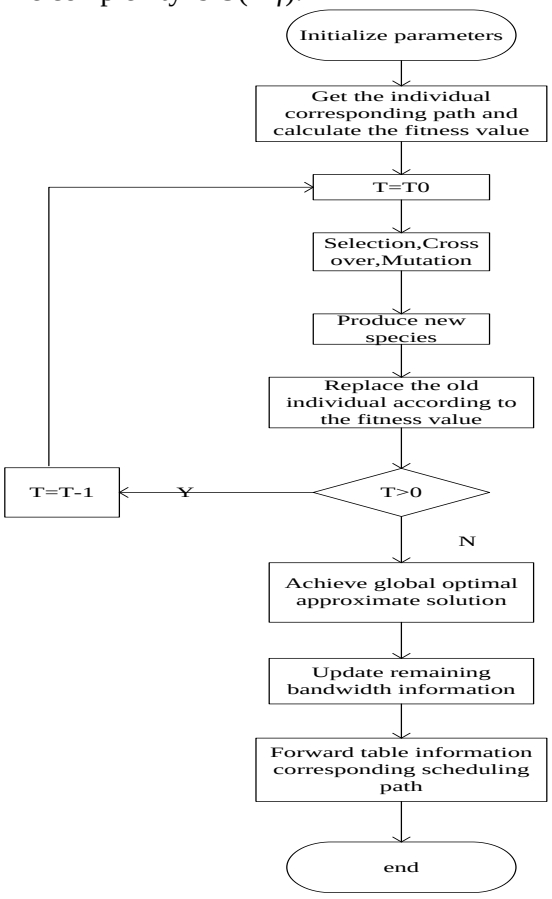

Fig. 3. SAGA steps in detail

When the adaptive scheduling strategy filters out some flows, SAGA schedules them to allocate available paths. In the following, we will introduce the adaptive strategy.

4.2 On-demand Traffic Adaptive Scheduling Strategy

In a multi-rooted tree topology, there are several possible equal-cost paths between any pair of source and destination hosts. When a new large flow is detected, (e.g.10% of the host's link capacity),the scheduler linearly searches all possible paths to find one whose link components can all accommodate that flow. If a path is found, then that flow is placed on that path. Choosing some elephant flows is a prerequisite. On-demand traffic adaptive scheduling strategy considers not only the bandwidth of network resource usage, but also the effect of the demand. The change of the flow bandwidth demands in the current network has an effect on the estimation bandwidth demand after adding the new flow. And according to the natural flow bandwidth demand, the current network bandwidth resources should be dispatched reasonably.

When the flow in the network reaches the OpenFlow switch, the data packet is forwarded by the default ECMP, meanwhile, the controller stores the ECMP information. In addition, the adaptive flow controlling module filters out the elephant flow set (ElephantFlows) in the network according to the flow statistics. At the same time, it is important to choose the flow set (ScheduleFlows) that needs to be scheduled judging from the adaptive scheduling strategy. Because some flows satisfying their bandwidth demands can be scheduled on previous paths, it can reduce the load of the controller. The chosen strategy algorithm is described in Algorithm 2. If the total number of flows that need to be scheduled is F, the time complexity will be O(F). Eventually, it may calculate a new path by using SAGA in the light of the status of the resource usage in the current network, and update the OpenFlow switch information. The above process is shown in **Fig. 4**.

Algorithm 2: the chosen strategy based on bandwidth demand changing

```
for all flow f in Elephantflows do
 if f.assigned = False then
  ScheduleFlows ←ScheduleFlows ∪ {f}
  DstHostList ←DstHostList ∪ {f.dstIP}
 else
  if f.new_demand < f.old_demand
   for all links in f.path do
    FreeBW[l] ← FreeBW[l] + f.old_demand - f.new_demand
  else if f.new_demand > f.old_demand then
   IncreaseFlows ←IncreaseFlows ∪ {f}
  end if
 end if
end for
for all flow f in IncreaseFlows do
 if All links in f.path satisfy FreeBW > f.new_demand - f.old_demand then
  for all link l in f.path do
   FreeBW[l] ← FreeBW[l] + f.old_demand - f.new_demand
 else
  for all link l in f.path do
```

```
20        FreeBW[l] ← FreeBW[l] + f.old_demand
21        ScheduleFlows ← ScheduleFlows ∪ {f}
22        DstHostList ←DstHostList ∪ {f.dstIP}
23     end if
24  end for
```

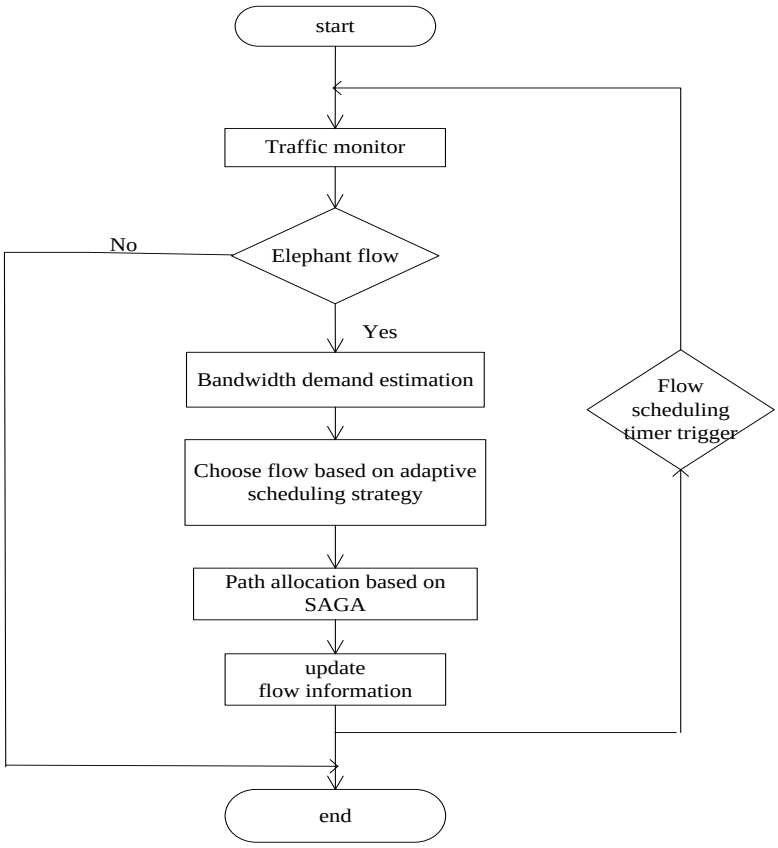

Fig. 4. On-demand adaptive traffic scheduling

The adaptive scheduling strategy chooses the flow satisfying the demand to schedule on the basis of the Algorithm 2. Thus, SAGA does not necessarily schedule all flows in all paths.

5. mplementation and evaluation

We have implemented our flow scheduling algorithm which is based on the POX [28] controller and Mininet [29]. Providing a good definition of API is the purpose of POX. Mininet should be a lightweight virtualization technology to simulate the network environment with multiple hosts, switches and links. The goal of our test is to determine the aggregate bisection bandwidth in various traffic patterns.

In the Fat-Tree topology, the traffic can be divided into the three kinds by switch types. They include:

(1) Intra-rack traffic: a host sends to a host in the same edge switch.

(2) Intra-pod traffic: a host sends to a host in the same pod.

(3) Inter-pod traffic: a host sends to a host in the different pods.

In the following, four communication models[2] are used in the experimental simulation.

(1) Stride(i): a host with x communicates with host with index(x+i) mod 16.

(2) Staggered Prob (Edge_P，Pod_P)：a host communicates with another host in the same edge switch by Edge_P, and to its same pod by Pod_P, and to the rest of the network by 1-Edge_P-Pod_P.

(3) Random: a host sends to any other host in the network with uniform probability.

(4) Randombij: a host communicates with the other hosts in some mapping.

5.1 Analysis of bisection bandwidth

We test the bisection bandwidth with different scheduling algorithms in the different communication models: stride, staggered, random and randombij, which are shown in **Fig. 5, 6, 7, 8,** respectively. Non-blocking is an ideal state without any limits. In the stride communication pattern, we choose stride(1),stride(2), stride(4) and stride(8). Three traffic types are distributed as Table 1. with the increase of the parameter i, the average bandwidth gradually decreases. The reason is that the inter-pod flow rate of the communication pattern will increase. The adding inter-pod flow will lead to the extra link load of the core layer and the congestion of some links. In each stride, SAGA is almost equal to SA except for stride(2). In stride(2), SAGA performs better than SA with about 5.86 Mb/s. Because the traffic has two kinds of intra-pod and inter-pod in that communication patter. SAGA considers just the two kinds patters. SAGA is better than GFF with 20 Mb/s and outperforms ECMP with 30 Mb/s in all stride case. Because SAGA can schedule flows to different core and aggregation switches, which reduces the load in the network.

Table 1. Rate of three traffic types

Stride(i)	Intra-rack	Intra-pod	Inter-pod
Stride(1)	50%	25%	25%
Stride(2)	0%	50%	50%
Stride(4)	0%	0%	100%
Stride(8)	0%	0%	100%

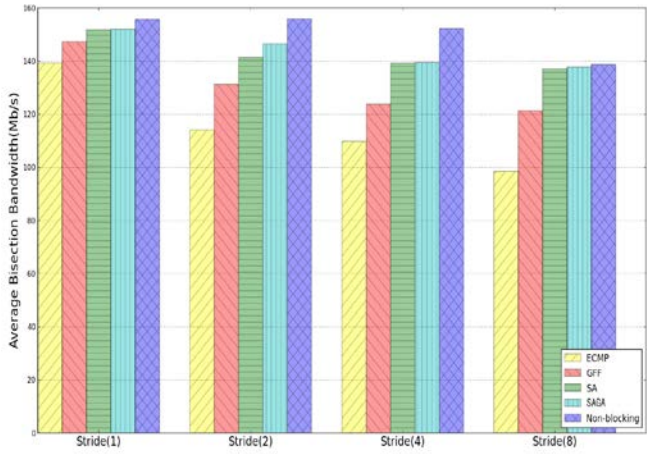

Fig. 5. Average bisection bandwidth in Stride(i)

In the staggered communication pattern, we test in staggered(0.2,0.3) , staggered(0.2,0.5) and staggered(0.5,0.3). Each pattern has three groups. The bisection bandwidth of SAGA is about 145 Mb/s in all communication patters. As a whole, SAGA outperforms ECMP, GFF and SA. Especially, SA is 12.3% lower than SAGA in stag2(0.2,0.3) . Because 50% inter-pod traffic and 30% intra-pod traffic exist in this patter. SA only considers the different pod instead of the same pod, nevertheless SAGA considers the two kinds of flow scheduling. GFF is 30% lower than SAGA in stag0(0.2,0.3). Due to the dynamic scheduling mechanism, SAGA can schedule the flow according to the dynamic demand. The bisection bandwidth in stag(0.2,0.3) is lower than stag(0.5,0.3) for all algorithms, because the traffic passing a core switch is added in stag(0.2,0.3) , which causes more load in the core switch with a low link utilization. However, no matter what, SAGA outperforms other algorithms.

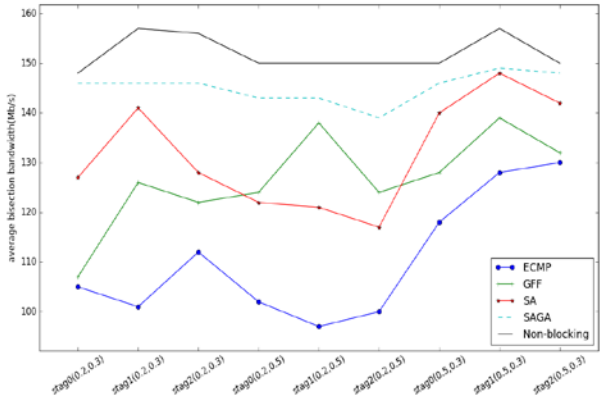

Fig. 6. Average bisection bandwidth in StaggeredProb

In random communication pattern, we run experiment for three groups. In the best case, SAGA outperforms SA with 8.12 Mb/s. Since the host within the pod randomly sends traffic to the destination host, the random mapping method of between the destination host and the top switch in SAGA has greater advantages over searching current scheduling path of the approximate optimal solution. The average bisection bandwidth is about 20 Mb/s higher than others in the case.

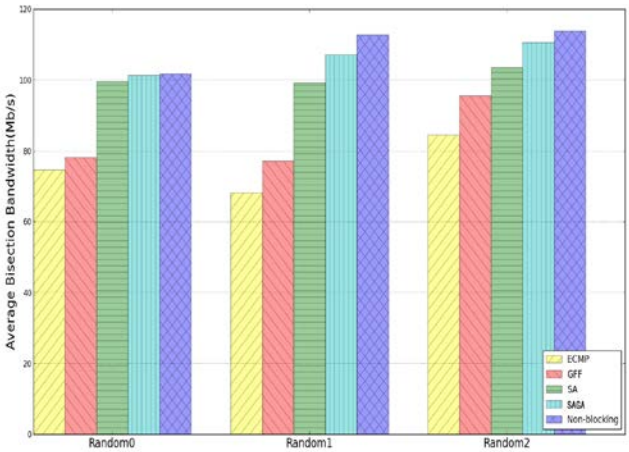

Fig. 7. Average bisection bandwidth in Random

In randombij, we run experiment for three groups. Since the mapping between the host the highest switch, which contributes to adding the scale of searching spaces, SAGA is about 15.23 Mb/s lower than SA at the worst case. Given the dynamic scheduling mechanism, SAGA still outperforms GFF and ECMP.

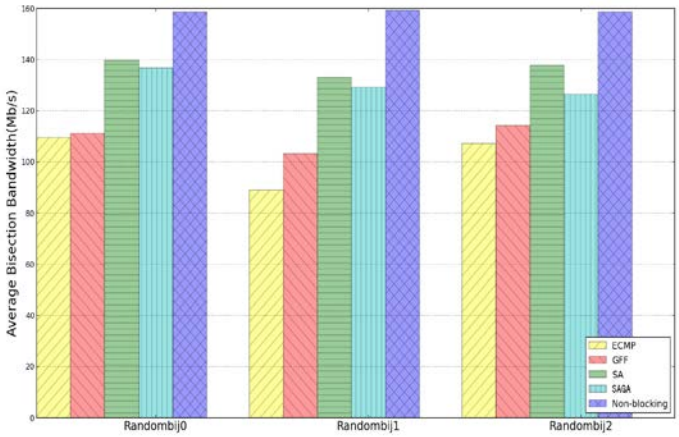

Fig. 8. Average bisection bandwidth in Randombij

In order to verify the scalability of the algorithm, we compare SAGA with Ashman-BestFit. Ashman-ProFit. Ashman-BestFit and Ashman-ProFi are proposed to reduce the fragment. It is described in **Fig. 9, 10, 11, 12**. In stride, SAGA is higher than Ashman-BestFit with 15.76 Mb/s and Ashman-ProFi with 7.68 Mb/s. However, SAGA is slightly lower than Ashman-ProFi in stride(4) and Ashman-BestFit in stride(8). Because there is only inter-pod traffic in stride(4) and stride(8). The mapping of SAGA may add the range of searching the best path. And in randombij, SAGA is lower than Ashman-BestFit with 4.13 Mb/s. In staggered and random, the average bisection bandwidth of SAGA is higher than Ashman-BestFit and Ashman-ProFit.

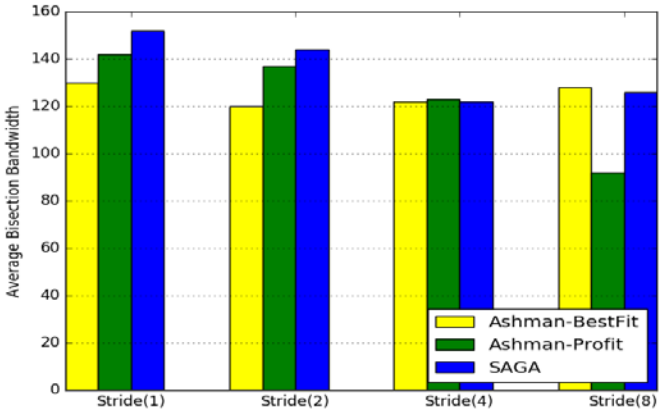

Fig. 9. Average bisection bandwidth in Stride(i)

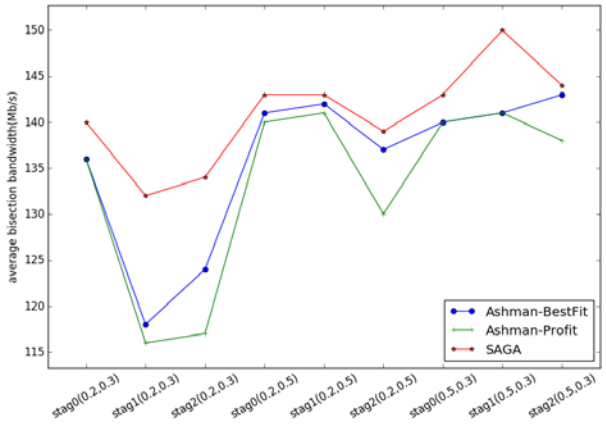

Fig. 10. Average bisection bandwidth in StaggeredProb

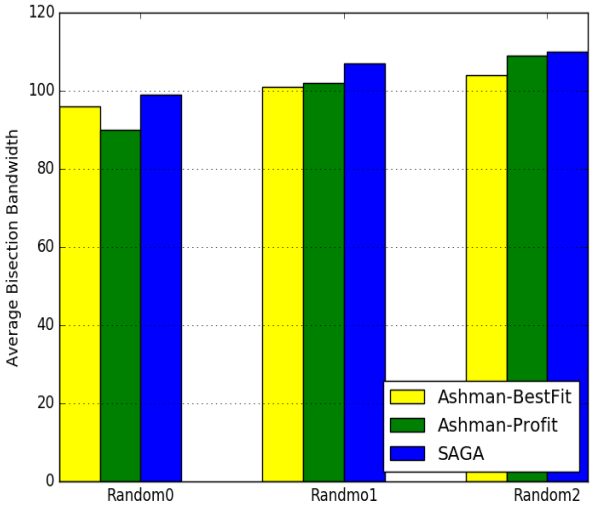

Fig. 11. Average bisection bandwidth in Random

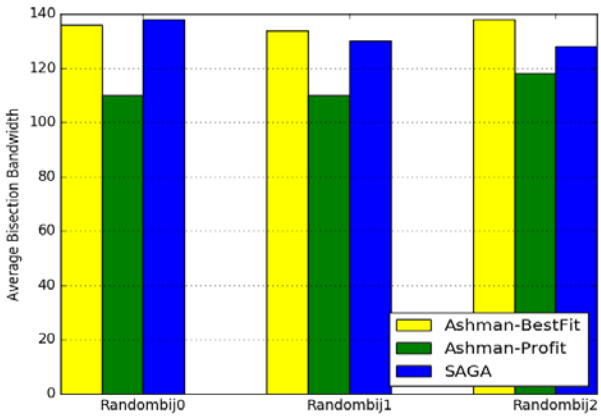

Fig. 12. Average bisection bandwidth in Randombij

5.2 Analysis of convergence time

In order to explore the convergence time in SAGA, we tested it in above communication patters. We show in **Table 2** the average convergence time. In Staggered and random communication mode, the average convergence time of SA is less than SAGA. Because SAGA takes into account the traffic scheduling between the same pod, the range of searching increases. In the above two modes, the bisection bandwidth obtained by SAGA is higher than SA, thus verifying that SAG increases the search time to achieve high bandwidth. However, in Stride4 and Stride8 communication modes, the average convergence time of SA is greater than SAGA. On the one hand, because the traffic between the scheduling only involves traffic scheduling between two different pods, SAGA reduces the search scope. On the other hand, SAGA selects some urgent scheduled flows according to the adaptive scheduling mechanism, so the number of flow needed to be scheduling may be less than SA. In the randombij model, the average convergence time of SA is greater than SAGA. Since the traffic in this mode is mapped by one to one, the mapping between the host and the core switches used by SA can obtain the optimal solution. Due to the small range of search in SAGA , so it may not get the best solution and verify that the SA two bandwidth higher than SAGA at the same time.

Table 2. Average convergence time (s)

Communication Patterns	SA	SAGA
Stag_0_2_3	0.4573	0.8914
Stag_1_2_3	0.4321	0.7833
Stag_2_2_3	0.4993	0.7220
Stag_0_5_3	0.2740	0.6653
Stag_1_5_3	0.3200	0.7380
Stag_2_5_3	0.3386	0.7533
Stride1	0.2440	0.6113
Stride2	0.4160	0.7653
Stride4	1.1526	0.7160
Stride8	0.8513	0.7913
Random0	0.6626	0.6840
Random1	0.5426	0.5873
Random2	0.5493	0.7693
Random0_bij	0.5426	0.5300
Random1_bij	0.7486	0.7340
Random2_bij	0.6613	0.6413

5.3 Scalability

To evaluate the scalability of our scheduling algorithm, we tested the algorithm in k=6. The parameter is set as **Table 3**. Because of the difference of the two kinds of topological total bisection bandwidth, the average bisection bandwidth rate of many algorithms was compared and analyzed, as shown in **Fig. 13**. We can see from the figure, with the increase of the network size, the average bandwidth of each traffic scheduling algorithm is decreased.

Because the scale is bigger, the searching space is wider. In summary, the result show that the SAGA based on adaptive demand scheduling strategy has an advantage over other existing scheduling approaches for data centers.

Table 3. The total bisection bandwidth in different network scale

Scale	Core switch	Aggregation switch	Edge switch	Host	Total bisection bandwidth
k=4	4	8	8	16	160Mbps
k=6	9	18	18	54	270Mbps

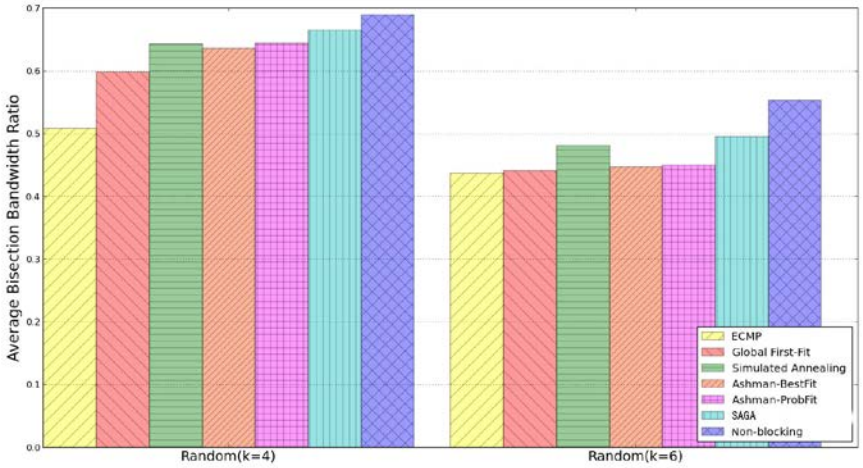

Fig. 13. Average bisection bandwidth in random

6. Conclusion

Since the problem of low link bandwidth utilization in data center network is gradually addressed in more hot fields, and previous approaches fail to schedule according to their demands in real time, this paper presents the SAGA strategy based on the adaptive traffic scheduling mechanism through software defined network technology. Due to the good features in fat-tree with the high degree of available path diversity, SAGA based on adaptive traffic scheduling mechanism considers not only the communication in the different pods, but also the communication in the same pod. In addition, recovering resources is included in our module. Experimental results show that SAGA based on traffic adaptive scheduling mechanism achieves higher bisection bandwidth than those of ECMP, GFF, SA, Ashman-BestFit and Ashman-ProFit. We believe that our scheduling mechanism can better facilitate bandwidth resources in links and improve network utilization. However, the threshold for filtering out flows is set by hand. And our way shows the shortage in randombij. In the future, we will focus on getting the threshold dynamically and narrowing the searching space.

References

[1] D Li，GH Chen，FY Ren，CL Jiang，MW Xu, "Data Center Network Research progress and trend," *Chinese Journal of Computer*, vol. 37, no. 2, pp.259-274 , 2014. Article (CrossRef Link)

[2] Al-Fares M, Radhakrishnan S, Raghavan B, "Hedera: Dynamic Flow Scheduling for Data Center Networks," in *Proc. of 7th USENIX conference on Network Systems Design and Implementation (NSDI)* , pp. 19-19, April 28 – 30, 2010.

[3] Yuchen Liu, "Research on bandwidth fragment in network load balancing of data centers," shanghai: Shanghai Jiao Tong University, 2014.

[4] Benson T, Anand A, Akella, "MicroTE: Fine Grained Traffic Engineering for Data Centers," in *Proc. of 7th conference on emerging Networking Experiments and Technologies*, December 06 – 09, 2010. Article (CrossRef Link)

[5] Wang W, Sun Y, Zheng K, "Freeway: Adaptively Isolating the Elephant and Mice Flows on Different Transmission Paths," in *Proc. of 22th International Conference on Network Protocols*, pp. 362-367, October 21-24, 2014. Article (CrossRef Link)

[6] M Rifai, D Lopez-Pacheco, G Urvoy-Keller, "Coarse-grained Scheduling with Software-Defined Networking Switches," in *Proc. of ACM Conference on Special Interest Group on Data Communication*, pp. 95-96, August 17 -21,2015. Article (CrossRef Link)

[7] Greenberg A, Hamilton JR, Jain N , "VL2: A scalable and flexible data center network," in *Proc. of the ACM SIGCOMM 2009 conference on Data communication*, pp. 51-62, August 16-21, 2009. Article (CrossRef Link)

[8] Wu X, Yang X , "DARD: Distributed Adaptive Routing for Datacenter Networks," in *Proc. of 33th IEEE Conf. on Distributed Computing Systems*, pp. 32-41, June 18-21,2012. Article (CrossRef Link)

[9] Cui W, Qian C, "DiFS: distributed flow scheduling for adaptive routing in hierarchical data center networks," in *Proc. of 10th ACM/IEEE symposium on Architectures for networking and communications systems (ANCS)*, pp. 53-64, October 21-22, 2014. Article (CrossRef Link)

[10] Zuo Q Y, Chen M, Zhao G S, "Research on SDN technology based on OpenFlow Technologies," *Journal of software*, vol. 24, no. 5, pp. 1078-1097, 2013. Article (CrossRef Link)

[11] Kreutz D, Ramos FMV, Rothenberg C, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 10-13, 2014. Article (CrossRef Link)

[12] Nunes BA. A, Mendonca M, Nguyen XN, "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617-1634, 2014. Article (CrossRef Link)

[13] OpenFlow Consortium. OpenFlow website[EB/OL]. [Online]. Available: http://archive. OpenFlow. org/.

[14] Erickson D, "The beacon openflow controller," in *Proc. of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pp. 13-18, August 16-16, 2013. Article (CrossRef Link)

[15] Mckeown N, Anderson T, Balakrishnan H, "OpenFlow: enabling innovation in campus networks, " *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69-74, 2008. Article (CrossRef Link)

[16] Florin Pop, Ciprian Dobre, Dragos Comaneci , Joanna Kolodziej, "Adaptive scheduling algorithm for media-optimized traffic management in software defined networks," *Computing*, vol. 98, pp. 147–168, 2016. Article (CrossRef Link)

[17] Al-Fares M, Loukissas A, Vahdat A , "A scalable commodity data center network architecture," in *Proc. of the ACM SIGCOMM 2008 conference on Data communication*, pp. 63-74, August 17-22, 2008. Article (CrossRef Link)

[18] Heller B, Seetharaman S, Mahadevan P, "ElasticTree: Saving energy in data center networks," in *Proc. of 7th USENIX conference on Networked Systems Design and Implementation (NSDI)*, pp. 17-17, April 28-30, 2010. Article (CrossRef Link)

[19] Mysore RN, Pamboris A, Farrington N, "PortLand: A scalable fault-tolerant layer 2 data center network fabric," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4 , pp. 39-50, 2009. Article (CrossRef Link)
[20] Singla A, Hong CY, Popa L, "Jellyfish: Networking Data Centers Randomly," *In Proc. of 9th USENIX conference on Networked Systems Design and Implementation*, pp. 17-17, April 25-27, 2011.
[21] Farrington N, Porter G, Radhakrishnan S , "Helios: A hybrid electrical/optical switch architecture for modular data centers," in *Proc. of the ACM SIGCOMM 2010 conference*, pp. 339-350, August 30- September 03 , 2010. Article (CrossRef Link)
[22] Wang G, Andersen DG, Kaminsky M , "c-Through: Part-time optics in data centers," in *Proc. of the ACM SIGCOMM 2010 conference*, pp. 327- 338, August 30 - September 03, 2010. Article (CrossRef Link)
[23] Chen K, Singlay A, Singhz A, "OSA: An optical switching architecture for data center networks with unprecedented flexibility," *IEEE/ACM Transactions on Networking*, vo1.22, no. 2, pp. 498-511, 2014. Article (CrossRef Link)
[24] Guo C, Wu H, Tan K, "Dcell: A scalable and fault-tolerant network structure for data centers," *Acm Sigcomm Computer Communication Review*, vol. 38, no. 4, pp. 75-86, 2008. Article (CrossRef Link)
[25] Guo C, Lu G, Li D, "BCube: A high performance, server-centric network architecture for modular data centers," *Acm Sigcomm Computer Communication Review*, vol. 39, no. 4, pp. 63-74, 2009. Article (CrossRef Link)
[26] Li D, Guo C, Wu H , "FiConn: Using backup port for server interconnection in data centers," in *Proc. of INFOCOM* , pp. 2276-2285, April 19-25, 2009. Article (CrossRef Link)
[27] Wu H, Lu G, Li D, "MDCube: A high performance network structure for modular data center interconnection," in *Proc. of 5th international Conference on emerging Networking Experiments and Technologies (CoNEXT)*, pp. 25-36, December 01- 04, 2009. Article (CrossRef Link)
[28] Huang L, Jia Q, Wang X , "PCube: Improving power efficiency in data center networks," in *Proc. of the 2011 IEEE International Conference on Cloud Computing (CLOUD)*, pp. 65-72, July 04 -09, 2011. Article (CrossRef Link)
[29] POX: https://openflow.stanford.edu/display/ONL/POX+Wiki.
[30] Mininet: https://mininet.org.

Wentao Wang

He received M.S. degrees in Computer Application Technology from South Central University for Nationalities, Wuhan, China, in 2001, and received the Ph.D. degree in Control science and Engineering from Huazhong University of Science and Technology in 2010. Also, from 2002 to 2003, he worked as a visiting scholar at department of Computer Engineering, Chonbuk National University, South Korea. His current research interests include mobile and sensor networks, image processing, and computational intelligence.

Lingxia Wang

She received bachelor degree in information engineering from Hubei University of Chinese Medicine in 2015 and now studying for her master Degree of computer application technology in South-Central University for Nationalities. Her research interesting includes computer networks.

Fang Zheng

She received bachelor degree in electronic and information engineering from Huazhong University of Science and Technology Wuchang Branch in 2012 and master Degree in computer application technology from South-Central University for Nationalities in 2016. Her research interesting includes computer networks.