BSFS: A Bidirectional Search Algorithm for Flow Scheduling in Cloud Data Centers

Hasibeh Naseri
Department of Computer Engineering and IT, University of Kurdistan, Sanandaj, Iran
hasibehnaseri@gmail.com
Sadoon Azizi*
Department of Computer Engineering and IT, University of Kurdistan, Sanandaj, Iran
s.azizi@uok.ac.ir
Alireza Abdollahpouri
Department of Computer Engineering and IT, University of Kurdistan, Sanandaj, Iran
abdollahpouri@uok.ac.ir

Received: 04/Jul/2019 Revised: 24/Aug/2019 Accepted: 08/Dec/2019

Abstract

To support high bisection bandwidth for communication intensive applications in the cloud computing environment, data center networks usually offer a wide variety of paths. However, optimal utilization of this facility has always been a critical challenge in a data center design. Flow-based mechanisms usually suffer from collision between elephant flows; while, packet-based mechanisms encounter packet re-ordering phenomenon. Both of these challenges lead to severe performance degradation in a data center network. To address these problems, in this paper, we propose an efficient mechanism for the flow scheduling problem in cloud data center networks. The proposed mechanism, on one hand, makes decisions per flow, thus preventing the necessity for rearrangement of packets. On the other hand, thanks do SDN technology and utilizing bidirectional search algorithm, our proposed method is able to distribute elephant flows across the entire network smoothly and with a high speed. Simulation results confirm the outperformance of our proposed method with the comparison of state-of-the-art algorithms under different traffic patterns. In particular, compared to the second-best result, the proposed mechanism provides about 20% higher throughput for random traffic pattern. In addition, with regard to flow completion time, the percentage of improvement is 12% for random traffic pattern.

Keywords: Cloud Computing; Data Center Networks; Flow Scheduling; Routing Algorithm; Load Balancing; Bidirectional Search.

1- Introduction

Over the past few years, several companies and organizations have shifted their services such as large scale computing, web search, online gaming, and social networking to cloud computing environment [1]. Recently, with the emergence of IoT-based applications and massive data processing, the demand for cloud resources has increased dramatically. In order to meet these needs, various data center networks are deployed around the world, including hundreds of servers and large amounts of traffic are exchanged between them.

Today's data center networks often use multi-rooted tree topologies such as Fat-tree [2-4] and Clos [5, 6]. These topologies provide multiple paths at an equal cost between each pair of end hosts, and thus significantly increase bisection bandwidth. However, given the burstiness and unpredictable nature of the traffic matrix and the flow pattern generated by virtual machines on hosts, achieving load balancing in a data center network is not a trivial task.

Over the past few years, network researchers and traffic engineers have proposed various algorithms and mechanisms to provide load balancing in cloud data center networks [7–17]. Although these efforts are valuable steps towards improving the efficiency of data center networks, there exist still some challenges and issues in this regard. The mechanisms that use *per-packet* approach to manage network traffic, although provide good load balancing across the network, but they are faced with the phenomenon of packet re-ordering. Packet re-ordering not only affects TCP throughput but also imposes significant computational overhead on hosts [12]. On the other hand, *flow-based* mechanisms usually suffer from the

* Corresponding Author

phenomenon of collision between the elephant flows, which leads to network performance reduction. Therefore, the issue of load balancing in data center networks is still challenging and needs further research efforts [1].

In this paper, we aim to design an efficient flow-based mechanism to achieve load balancing in data center networks. Given the fact that most flows in data centers are only a few kilobytes in size (i.e., mice flows) and a very small percentage of them are large-sized flows (i.e., elephant flows), we take advantage of a hybrid mechanism for flow scheduling in a data center network. For this purpose, we use the distributed ECMP algorithm for mice flows, while a central controller is used for elephant flows. When an elephant flow is detected by a host, it sends the flow to the controller for routing the first packet. In the controller, based on the defined cost matrix, an optimal bidirectional search is performed on the network to find and select the best route for that flow.

Our proposed mechanism has three major advantages. First, it prevents the packet re-ordering phenomenon; because it performs per flow. Second, since the controller is used only for elephant flows, it does not become a bottleneck. Third, because the central controller provides a macroscopic view of the network traffic, our algorithm is able to distribute the elephant flows smoothly across the network. We have compared our approach with various mechanisms such as Static [2], ECMP [18] and DiFS [19]. The results of the experiments clearly show the superiority of our algorithm in terms of *delay*, *throughput* and *flow completion time* in comparison with other mentioned approaches.

The rest of the paper is organized as follows. In Section 2, related works are reviewed. Background and problem definition are described in Section 3. The proposed mechanism is presented in Section 4. In Section 5, we describe the simulation and evaluate the performance of the proposed method. Finally, Section 6 concludes the paper.

2- Related Works

In general, flow scheduling algorithms are divided into two main categories [1]: distributed and centralized. On the other hand, in terms of how the flows are handled, they can be classified into three categories [1]: packet-based, flow-based and flowlet-based. Below, we review some of the most important works performed on flow scheduling in cloud data center networks.

ECMP [18] is the most common routing algorithm for flow scheduling in data center networks. It is a distributed and flow-based algorithm. When a flow enters a switch for the first time, the ECMP performs the routing operation by applying the hash function to the header of the packet. Although the implementation of this algorithm is very simple, it does not differentiate between mice and elephant flows; and therefore, collisions between elephant flows is inevitable.

DARD [4] is another flow-based distributed algorithm. In this algorithm, end-hosts are responsible for monitoring the status of network traffic. Based on the network feedback received from the probe packets, each host moves the flows from high-traffic routes to low-traffic routes. However, injecting a large number of probe packets into the network puts considerable overhead on it. In addition, since this algorithm is host-based, all hosts need to be upgraded, which imposes a lot of administrative costs.

Cui et al. [14, 19] have recently proposed an adaptive distributed mechanism, called DiFS, for flow scheduling in data center networks with Fat-tree topology. They use ECMP to forward mice flows, while for scheduling the elephant flows each switch greedily distributes them to the output ports. In order to prevent over-utilized links, DiFS may change the path of some flows based on the collaboration between switches. Simulation results show that DiFS performs far better than ECMP. However, since this algorithm has no macroscopic view of the network, it has to transmit a large number of messages between the switches to provide load balancing. This, on the one hand, leads to overhead on the network, and on the other hand, as some flows change their direction, packets may need to be re-ordered.

DRILL [20] is another distributed mechanism which is inspired by the idea of "the power of two random choices". In each switch and for each packet, this approach decides which output port to send the packet to, based on local information about the queue length. The port selection mechanism in DRILL is simple and easy to implement. However, due to the fact that DRILL operates on a per-packet basis, packet out-of-ordering is inevitable.

Some other works [2, 21, 22] use Static or deterministic routing to forward packets over the network. In Static routing, the path between each host pair is determined in advance and remains unchanged. Although in practice the implementation of such mechanism is very simple, it cannot well take advantage of the multi-path benefit provided by the topology of data center networks, and usually provides very low performance.

Hedera [7] is a dynamic flow scheduling system in which mice flows are separated from elephant flows using a specified threshold value. By default, Hedera uses ECMP to transmit flows on the network. However, when a

large flow is detected, the system generates a demand matrix for active flows. Therefore, it proposes two schedulers "Global First Fit" and "Simulated Annealing" to send the flows. The results show that Hedera achieves a significant performance improvement compared to ECMP for the moderate cost. On the one hand, the demand estimation matrix in Hedera is only run once per scheduling period, which takes about 200 milliseconds for a data center network with 27,648 hosts and 250,000 large flows. On the other hand, the execution time of the scheduler is in order of tens of milliseconds. Putting these two points together, it can be seen that it takes several hundred milliseconds to find a suitable approximate path for guiding large flows in a data center network, which is a considerable time. In addition, given the drastic changes in traffic patterns in data centers, Hedera has to build demand matrixes over and over again in a short period of time, which imposes a significant overhead on the system.

Wang et al. in [23] proposed an adaptive mechanism called Freeway for flow scheduling in data center networks. This mechanism partitions the paths between hosts into *low latency* and *high throughput* paths. It then transmits mice flows through low latency paths and elephant flows through high throughput paths. Although Freeway performs better than ECMP and Hedera, in practice it may leave much of the network's capacity unused [1].

Based on the ant colony optimization algorithm, the authors of [16] proposed a centralized scheduling mechanism for transmitting the flows in data center networks. Their algorithm divides the elephant flows into k segments and sends them through k edge-disjoint paths. Since the flows are broken in this algorithm, the problem of re-ordering packets arises.

Authors in [24] have modeled the elephant flow scheduling problem as a multi-knapsack problem and proposed a mechanism based on hybrid Genetic and Simulated Annealing algorithm to solve it. Simulation results confirm that their algorithm provides higher bisection bandwidth and lower latency in comparison with similar methods. However, since their approach is similar to Hedera, it has same drawbacks.

3- Background

The topologies proposed for data center networks over the past few years provide multiple paths between each pair of hosts [2, 5, 21, 25]. Although our proposed mechanism can be applicable to any topology, in this paper, we focus on the well-known and common Fat-tree topology [2]. In this section, we first briefly describe the Fat-tree topology. Then, we will focus on the characteristics of flows in data center networks. We then investigate the mechanisms for detection of mice flows and elephant flows. Finally, we illustrate the problem of collision of elephant flows in data center networks with a detailed example.

3-1- Fat-tree topology

Fat-tree is a hierarchical multi-root tree topology that contains three layers of switches called Top of Rack (ToR), aggregation, and core. In this topology, the switches are homogeneous and the degree of each switch is determined by the parameter n. Fat-tree consists of n pods, each pod having two layers and each layer has $n/2$ switches that form a complete bipartite graph. Figure 1 shows an example of a Fat-tree topology with 4-port switches ($n = 4$).

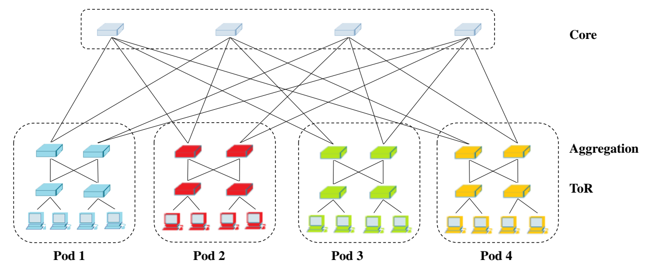


Fig. 1 Fat-tree topology with 4-port switches

In this work, we define the Fat-tree topology as a directed graph $G = (V, E)$ where, V represents the switches and E represents the links. Also, links that connect lower layer switches to higher layer switches are called *uphill* and links that connect higher layer switches to the lower layer switches are called *downhill* links.

3-2- Flow properties in data center networks

Each flow contains several packets that are chained together. In data center networks, if a flow contains a lot of packets, or it takes a long period of time, or its traffic is more than a threshold value, it is known as an elephant flow. On the other hand, flows with low information volumes or low number of packets are called mice flows [26]. In terms of number of flows in a data center network, typically more than 90% of them are mice, while only less than 10% of them are elephant flows. Nevertheless, on the other hand, more than 90% of the data volume belongs to the elephant flows and only 10% to the mouse flows [5]. This paradox highlights the importance of elephant flows.

3-3- Mechanisms to detect Elephant and mice flows

The mechanisms for detecting elephant flow from mice are divided into two main categories [27]:

Detection by edge switches: In this case, edge switches are responsible for detecting elephant flows. Hedera [7] and DiFS [19] use this method.

Detection by hosts: In this method, the detection of elephant from the mice flows is the responsibility of the host itself. Mahout [27] and DARD [4] are some of the algorithms that use this method.

3-4- The problem of collision between elephant flows

As previously mentioned in Section 2, DiFS is a greedy distributed mechanism for scheduling elephant flows in a data center network. In **Fig. 2**, suppose that hosts A and B produce 16 elephant flows in total, where the destination of 8 of these flows is host C or D (within the pod) while, other 8 flows are toward outside the pod. In switch *SW1*, DiFS distributes the flows quite evenly. But in the worst case, all of the eight flows that enter *SW3* may be intra-pod flows. This situation leads to improper equilibrium of the elephant flows in the links between aggregation and core layers. Having a macroscopic view, one can easily achieve the proper load balance in the network (see **Fig. 3**).

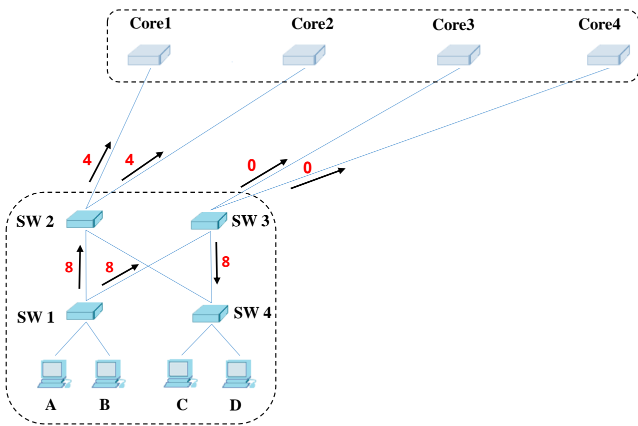


Fig. 2 The problem of load-balance in DiFS

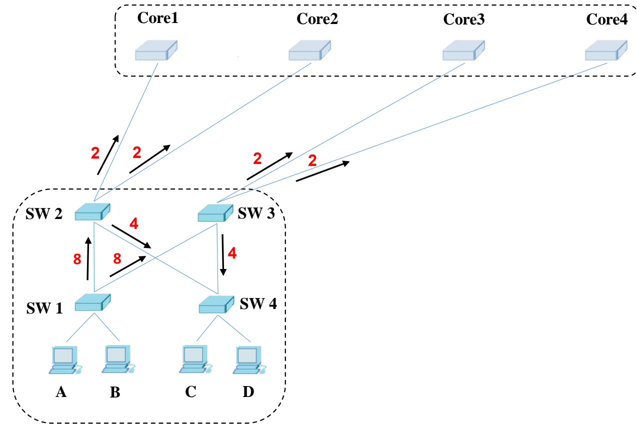


Fig. 3 Achieving a proper load balancing using macroscopic view

4- Proposed Method

In this section, we present an efficient mechanism for the flow scheduling problem in data center networks. To this end, we first give an overview of the proposed mechanism and then describe it.

4-1- Overview of the proposed method

The proposed algorithm has two main objectives. First, it aims to evenly distribute the load across the network. Second, it does not impose too much overhead on the central controller to achieve the first goal and can schedule the elephant flows at an acceptable speed. To manage traffic on a data center network, the proposed mechanism uses per-flow approach. This approach prevents out-of-ordering of packets in end-hosts. As a result, we will not confront a degradation in TCP performance and end-host memory usage. Our mechanism combines the advantages of both distributed and centralized systems. Due to their global view, centralized systems are very suitable for routing elephant flows, while distributed systems are the best option for routing mice flows to avoid overloading the central controller.

For the centralized system, we use a bidirectional search algorithm for scheduling elephant flows, which we describe in the following subsection. While for the distributed system, we use a simple yet efficient ECMP algorithm for mice flows. It is worth noting that in the proposed method, such as the mechanism presented in [27], the elephant flows are detected in the end-hosts. Similar to many of the existing works [4, 19, 27], we consider flows with a volume less than 100KB as mice flows and assume the others as elephant flows. In this work, we use the number of elephant flows as a load balancing parameter and the goal is to keep the number of active elephant flows on the network links as equal as

possible. Although other parameters such as *current bandwidth consumption* can be used for this purpose, the results presented in [19] show that this parameter would give us similar performance in practice. Fig. 4 shows the flowchart of the proposed method.

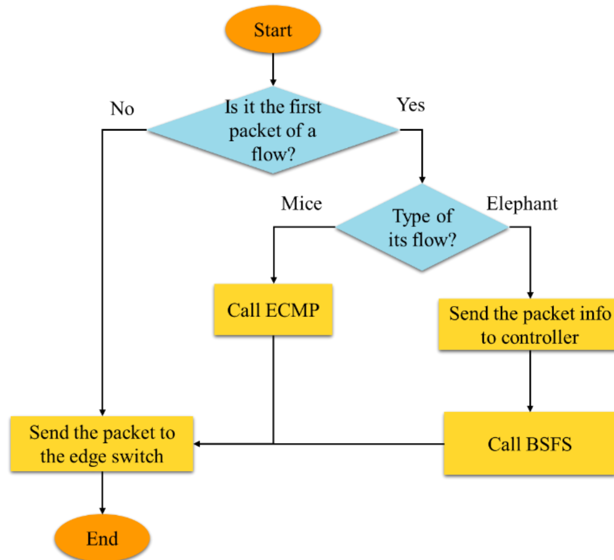


Fig. 4 The flowchart of the proposed method

4-2- BSFS

For elephant flows, we use bidirectional search algorithm to find an optimal path for each of them. When an elephant flow is detected by the source host and its destination host does not have the same edge switch as the source host, the packets of that flow are labeled with an "*E*", indicating that the flow is elephant. Upon arrival of the first packet from an elephant flow to the edge switch, that switch sends the source and destination address of the packet to the controller to find the appropriate route. The controller executes the proposed BSFS algorithm and, through the OpenFlow protocol, installs routing information on the switches in the path suggested by the algorithm. On the other hand, when the last packet of a flow is processed by the source edge switch, a request to update the network traffic information is sent to the controller. The details of the proposed algorithm are discussed below.

As mentioned previously in subsection 3.1, we use a directed graph to represent the Fat-tree topology. Based on this graph, we create a cost matrix $M_{m\times 2n}$, where m is the number of network switches and n is the number of ports per switch. Each element of this matrix represents the number of active elephant flows on each network link. The reason for using $2n$ numbers for each switch is that we use a directed graph to model the topology; n numbers for uphill links and n numbers for downhill links.

When a packet from an elephant flow is sent to the controller for routing, depending on the source and destination address of the packet, the controller can determine whether the two hosts are in the same pod or they are located in separate pods. If two hosts are in the same pod, the BSFS algorithm selects the best aggregation switch as the intermediate switch using a simple bidirectional search. But if the two hosts are located in two separate pods, the proposed BSFS algorithm starts two searches simultaneously; first one from the source edge switch to the core switches, searching between uphill links, and the other one from the destination edge switch to the core switches, searching between the downhill links. The aggregated result of these two searches is obtained for each of the core switches, and finally, using a simple linear search, the core switch that gives us a smaller value is chosen for routing. It is worth mentioning that in the Fat-tree topology, when the core switch is specified, there will be only one path between each pair of hosts [2]. It is also important to note that since the two searches are completely independent of each other, they can be run in parallel, which significantly reduces the execution time. On the other hand, when the last packet of an elephant flow is reported to the controller, the cost matrix is immediately updated; That is, one unit is reduced from the cost of all links that were along that flow. **Algorithm 1** shows the pseudo-code of the proposed BSFS.

The algorithm takes the cost matrix, packet p (the first or last packet of a flow), the source address, and the destination address of the host as input. If p is the first packet of a flow, using the bidirectional search method in the cost matrix, the path with the lowest cost is found for that flow (lines 1 to 6). Otherwise, if p is the last packet of a flow, the cost matrix is updated (lines 7 to 9); that means the cost of all the links along that flow is decreased by one.

Algorithm 1. BSFS: Bidirectional Search Algorithm for Flow Scheduling

Input: Cost Matrix, packet *p*, *p.src*, *p.des*

Output: Optimal Path

```
1.  if p is the first packet of a flow then
2.     if p.src and p.des belong to the same pod then
3.        Find an aggregation switch with minimum cost
          using BS // BS stands for Bidirectional Search
4.     else
5.        Find a core switch with minimum cost using BS
6.     end if
7.  else if p is the last packet of a flow then
8.     Update the Cost Matrix and the flow table of related
       switches
```

9. **end if**

4-3- Time Complexity Analysis

Here, we analyze the time complexity of our proposed BSFS method. For the first packet of each elephant flow, the time complexity of the proposed algorithm is as follows. If the source and destination host of a packet have the same pod, our algorithm searches among the aggregation switches inside that pod to find a switch with minimum cost (lines 2 and 3). Since the number of aggregation switches is equal to $n/2$, this step needs $O(n)$. However, when the source and destination host of a packet are within different pods, our algorithm must find a core switch with minimum cost (lines 4 and 5). Regarding to the fact that the number of core switches in a Fat-tree topology is $n^2/4$ [2], so the time complexity is $O(n^2)$. As a result, the time complexity of the proposed algorithm is $O(n^2)$ for a Fat-tree topology with n-port switches. We should mention that in updating the cost matrix (lines 7 and 8), only three (in intra-pod case) or six switches (in inter-pod case) are involved for each flow, which is constant numbers.

It is worth to note that our BSFS method runs only for the first packet of elephant flows. Since the number of elephant flows in a data center usually is very low, the time complexity of our algorithm is reasonable.

5- Performance Evaluation

In this section, we evaluate the performance of our proposed BSFS algorithm. We compare it with Static [2], ECMP [18] and DiFS [19] in various respects. It should be noted that in this work we neglect DiFS performance degradation due to packet re-ordering.

5-1- Simulation settings

In this work, evaluation of the proposed algorithm on Fat-tree topology with 8-port switches is performed. C++ programming language has been used for simulation of the proposed method. In the literature, there are many works that use custom simulators [7, 9, 28, 29]. Experiments have been performed on a computer having Intel® Core™ i5 CPU 2.3 GHz and 16 GB of memory.

The event-driven simulation is developed on a packet level. The length of each packet is assumed to be 1KB. For each port, a buffer of size 64KB is assumed. The capacity of all network links is equal and set to 1Gbps. For the transmission delay, we consider 8μs while the propagation delay is ignored. In this work, the queuing delay has been considered.

In our experiments, each server generates 20 flows continuously. We consider each flow with the probability of 90% as a mice flow and with the probability of 10% as an elephant flow. The size of mice flows is chosen randomly from the values 2KB, 10KB or 100KB. For the elephant flows, we assume the fixed size of 10MB.

5-2- Traffic patterns

We have used the following synthetic traffic patterns to perform the experiments [7, 13, 19]:

Stride(i): This pattern sends a flow from host x to another host with the number $(x + i)\% N$; where, N represents the number of hosts in the network.

Random: In this traffic pattern, a host with index x sends a flow randomly with uniform probability to another host y anywhere in the network, such that, $x \neq y$.

Staggered(P_{host}, P_{pod}): In this pattern, a host sends its flows with the probability of P_{host} to another host connected to the same edge switch, and with the probability of P_{pod} to another host in the same pod. It also sends the flows to other hosts with different pods with the probability of $1 - P_{host} - P_{pod}$.

5-3- Evaluation criteria

We use the following criteria to evaluate and compare our proposed method with other mechanisms:

Flow Completion Time (FCT): This criterion specifies the end time of a flow. In fact, it indicates the time when all packets of a flow are received by the destination.

Delay: Indicates average network latency. This criterion tells us that how long it takes in average for a packet to reach its destination.

Aggregate Throughput: This criterion measures the utilization of network links. In fact, it indicates the average rate at which the network delivers the packets.

5-4- Simulation results

Here, we evaluate the results of simulations. **Fig. 5**(a) and **Fig. 5**(b) show the average delay and network aggregate throughput under different traffic patterns, respectively. As can be seen, for Stride(2) and Staggered(0.5,0.3) almost all methods have low delay and high throughput. However, in Stride(i), by increasing the value of i and in Staggered(P_{host}, P_{pod}) by decreasing the values of P_{host} and then P_{pod}, BSFS performs much better. For the Random traffic pattern, since it is more likely for elephant flows to collide and most of the flows will be out of the pods, our proposed algorithm performs best by establishing a proper balance between the elephant flows.

Table 1. The Flow Completion Time of different algorithms under Stride (4)

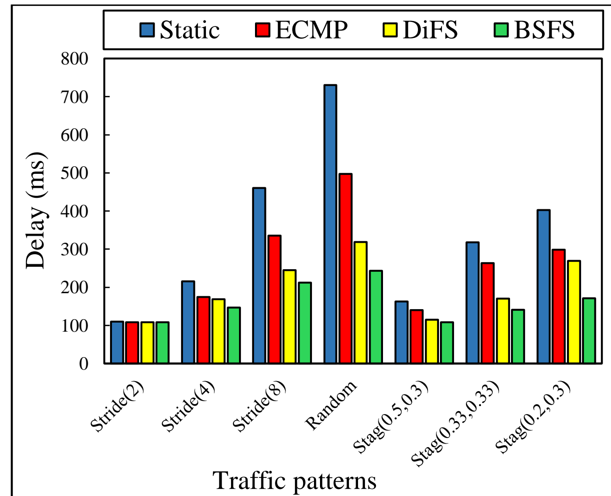
Algorithm	Time (in millisecond)										
	0	50	100	150	200	400	600	800	1000	1200	1400
Static	0	969	969	969	1087	1899	2088	2413	2528	2541	2560
ECMP	0	1011	1011	1011	1421	1913	2120	2454	2549	2551	2560
DiFS	0	1031	1031	1031	1561	2005	2299	2479	2551	2560	-
BSFS	**0**	**1030**	**1030**	**1030**	**1695**	**2120**	**2356**	**2560**	-	-	-

Table 2. The Flow Completion Time of different algorithms under Random

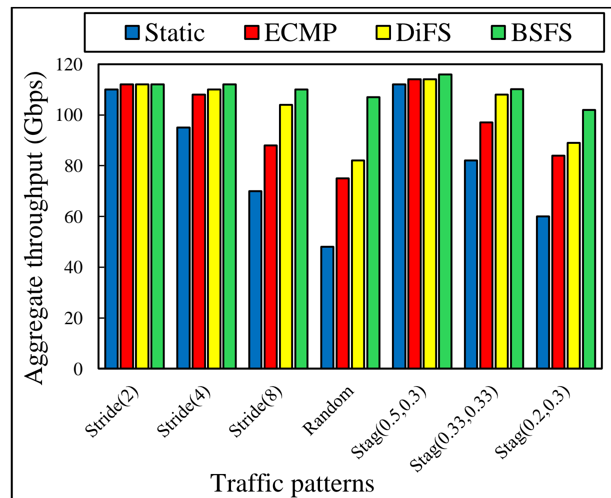
Algorithm	Time (in millisecond)															
	0	50	100	150	200	400	600	800	1000	1200	1400	1600	1800	2000	2200	2400
Static	0	976	980	984	1129	1310	1582	1671	1860	2088	2329	2434	2490	2519	2540	2560
ECMP	0	983	988	989	1207	1441	1740	1949	2081	2219	2416	2480	2529	2539	2560	-
DiFS	0	971	973	974	1439	1741	1961	2134	2262	2386	2481	2537	2549	2560	-	-
BSFS	**0**	**905**	**916**	**918**	**1393**	**1968**	**2389**	**2555**	**2560**	-	-	-	-	-	-	-

In particular, for the Random traffic pattern, BSFS provides about 20% higher throughput than DiFS. Therefore, we claim that our proposed mechanism performs better for non-local traffic than other mechanisms.

Table 1 and Table 2 illustrate the cumulative distribution function of number of completed flows for different algorithms under two traffic patterns Stride(4) (Table 1) and Random (Table 2). It can be clearly seen that BSFS terminates the flows earlier. For the Random traffic pattern, this superiority is much more impressive. This is due to the balanced distribution of elephant flows by the proposed method. While, in other algorithms, there is a longer flow completion time due to the frequent collisions between the elephant flows. For the Random traffic pattern, our BSFS delivers all the flows to their destination hosts below one second, while DiFS delivers about 88%, ECMP 81% and Static only deliver 72% of flows at this time.



(a) delay



(b) Aggregate throughput

Fig. 5 Performance comparison of algorithms under different traffic patterns

6- Conclusion and Future Work

In this paper, we proposed an efficient mechanism to achieve load balancing in data center networks. The proposed mechanism uses the ECMP algorithm to send mice flows, while it takes advantage of the bidirectional search algorithm in the central controller to schedule the elephant flows. Simulation results under various traffic patterns show that the proposed mechanism can balance the network load more efficiently and provide better performance in comparison with the Static, ECMP and DiFS mechanisms. The less locality in network traffic, the higher the advantage of our approach is. Specifically, for the Random traffic model, our mechanism provides 20% higher throughput than DiFS. As a possible future research direction, one can take into account the priority of flows when scheduling them. Furthermore, the proposed mechanism can be extended by considering failures in data center.

References

[1] Zhang, J., Yu, F. R., Wang, S., Huang, T., Liu, Z., Liu, Y.: Load Balancing in Data Center Networks: A Survey, *IEEE Communications Surveys & Tutorials,* vol. 20, no. 3, pp. 2324-52, 2018.

[2] Al-Fares, M., Loukissas, A., Vahdat, A.: A scalable, commodity data center network architecture, In *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4, pp. 63-74, 2008.

[3] Niranjan Mysore, R., Pamboris, A., Farrington, N., Huang, N., Miri, P., Radhakrishnan, S., Subramanya, V. and Vahdat, A.: Portland: a scalable fault-tolerant layer 2 data center network fabric, In *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4, pp. 39-50, 2009.

[4] Wu, .X, Yang, X.: Dard: Distributed adaptive routing for datacenter networks, In *32nd International Conference on Distributed Computing Systems (ICDCS)*, pp. 32-41, 2012, IEEE.

[5] Greenberg, A., Hamilton, J.R., Jain, N., Kandula, S., Kim, C., Lahiri, P., Maltz, D.A., Patel, P. and Sengupta, S.: VL2: a scalable and flexible data center network, In *ACM SIGCOMM computer communication review*, vol. 39, no. 4, pp. 51-62, 2009.

[6] Zahavi, E., Keslassy, I., Kolodny, A.: Distributed adaptive routing for big-data applications running on data center networks, In *Proceedings of the eighth ACM/IEEE Symposium on Architectures for networking and communications systems*, pp. 99-110, 2012.

[7] Al-Fares, M., Radhakrishnan, S., Raghavan, B., Huang, N., Vahdat, A.: Hedera: Dynamic Flow Scheduling for Data Center Networks, In *NSDI*, vol. 10, pp. 19-19, 2010.

[8] Zats, D., Das, T., Mohan, P., Borthakur, D., Katz, R.: DeTail: reducing the flow completion time tail in datacenter networks, In *Proceedings of the ACM SIGCOMM conference on Applications, technologies, architectures, and protocols for computer communication*, pp. 139-150, 2012.

[9] Sen, S., Shue, D., Ihm, S., Freedman, M.J.: Scalable, optimal flow routing in datacenters via local link balancing, In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, pp. 151-162, 2013.

[10] Modi, T., Swain, P., FlowDCN: Flow Scheduling in Software Defined Data Center Networks. In *IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, pp. 1-5, 2019.

[11] Alizadeh, M., Edsall, T., Dharmapurikar, S., Vaidyanathan, R., Chu, K., Fingerhut, A., Matus, F., Pan, R., Yadav, N. and Varghese, G.: CONGA: Distributed congestion-aware load balancing for datacenters, In *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 503-514, 2014.

[12] He, K., Rozner, E., Agarwal, K., Felter, W., Carter, J. and Akella, A.: Presto: Edge-based load balancing for fast datacenter networks, In *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 465-478, 2015.

[13] Zhang, J., Ren, F., Huang, T., Tang, L., Liu, Y.: Congestion-aware adaptive forwarding in datacenter networks, *Computer Communications,* vol. 62, pp. 34-46, 2015.

[14] Cui, W., Qian, C.: Difs: Distributed flow scheduling for adaptive routing in hierarchical data center networks, In *Proceedings of the tenth ACM/IEEE Symposium on Architectures for networking and communications systems*, pp. 53-64, 2014.

[15] Ghorbani, S., Yang, Z., Godfrey, P., Ganjali, Y. and Firoozshahian, A.: DRILL: Micro load balancing for low-latency data center networks, In *Proceedings of*

the Conference of the ACM Special Interest Group on Data Communication, pp. 225-238, 2017.

[16] Wang, C., Zhang, G., Chen, H. and Xu, H.: An ACO-based elephant and mice flow scheduling system in SDN, In *2nd International Conference on Big Data Analysis (ICBDA)*, pp. 859-863, 2017, IEEE.

[17] Perry, Perry, Balakrishnan, H., Shah, D.: Flowtune: Flowlet Control for Datacenter Networks, In *NSDI*, pp. 421-435, 2017.

[18] Hopps, C.E.: Analysis of an equal-cost multi-path algorithm, 2000.

[19] Cui, W., Yu, Y., Qian, C.: DiFS: Distributed Flow Scheduling for adaptive switching in FatTree data center networks, *Computer Networks,* vol. 105, pp. 166-179, 2016.

[20] Ghorbani, S., Godfrey, B., Ganjali, Y. and Firoozshahian, A.: Micro load balancing in data centers with DRILL, In *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*, p. 17, 2015.

[21] Azizi, S., Hashemi, N., Khonsari, A.: A flexible and high-performance data center network topology, *The Journal of Supercomputing,* vol. 73, no. 4, pp. 1484-1503, 2017.

[22] Guo, C., Wu, H., Tan, K., Shi, L., Zhang, Y. and Lu, S.: Dcell: a scalable and fault-tolerant network structure for data centers, In *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4, pp. 75-86, 2008.

[23] Wang, W., Sun, Y., Salamatian, K., Li, Z.: Adaptive Path Isolation for Elephant and Mice Flows by Exploiting Path Diversity in Datacenters, *IEEE Transactions on Network and Service Management,* vol. 13, no. 1, pp. 5-18, 2016.

[24] Li, P., Xu, H., Wang, R., Luo, B.: Data center network flow scheduling mechanism based on HGSAFS algorithm, In *Proceedings of the High Performance Computing Symposium*, 2019.

[25] Li, D., Wu, J.: On data center network architectures for interconnecting dual-port servers, *IEEE Transactions on Computers,* vol. 64, no. 11, pp. 3210-3222, 2015.

[26] Marron, J., Hernandez-Campos, F., Smith, F.: Mice and elephants visualization of internet traffic, In *Proceedings of Computational Statistics*, pp. 47-54, 2002.

[27] Curtis, A.R., Kim, W., Yalagandula, P.: Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection, In *Proceedings IEEE INFOCOM*, pp. 1629-1637, 2011.

[28] Wischik, D., Raiciu, C., Greenhalgh, A., Handley, M.: Design, Implementation and Evaluation of Congestion Control for Multipath TCP, In *NSDI*, vol. 11, pp. 8, 2011.

[29] Singla, A., Hong, C.-Y., Popa, L., Godfrey, P.B.: Jellyfish: Networking Data Centers, Randomly, In *NSDI*, vol. 12, pp. 1-6, 2012.

Hasibeh Naseri is a master graduated in artificial intelligence from University of Kurdistan, Sanandaj, Iran. Currently, she is a member of the Internet of Things research laboratory at the University of Kurdistan. Her research interests include Cloud computing, Data center networks, Flow scheduling, Heuristic and meta-heuristic algorithms.

Sadoon Azizi is an assistant professor in the department of computer engineering and IT, University of Kurdistan, Sanandaj, Iran. He received his Ph.D degree in computer science, with focus on Cloud Data Centers, from Amirkabir University of Technology, Tehran, Iran, in 2016. He also received his M.Sc. degree in computer science, with focus on High Performance Computing (HPC), from Amirkabir University of Technology, Tehran, Iran, in 2012. His main research interests include Cloud computing, Fog computing, Internet of Things, Data center networks, and Design and analysis of algorithms. He is the director of the Internet of Things research laboratory and manager at the High Performance Computing (HPC) center at the University of Kurdistan.

Alireza Abdollahpouri received his Ph.D. from University of Hamburg, Germany in 2012, and now he is an associate professor of computer networks at the Department of Computer Engineering, University of Kurdistan, Sanandaj, Iran. His main research interests are in the field of social network analysis, IPTV modeling, and quality of service in wireless networks.