Techniques to Improve a Flow Diffusion Algorithm for Folded Clos Networks

Satoru Ohta
Department of Electrical and Computer Engineering, Faculty of Engineering
Toyama Prefectural University
Imizu, Japan
e-mail: ohta@pu-toyama.ac.jp

***Abstract*—Folded Clos networks (FCNs) are important as topologies for data center networks. To achieve high performance with an FCN, it is necessary to establish a routing method that uniformly diffuses flows between links. To satisfy this requirement, a previous study proposed a method, called the "rebalancing algorithm," which is a distributed algorithm based on locally obtainable information. An advantage of this method is that the number of flows on a link is upper bounded by a theoretically derived constant. Therefore, the link load does not grow heavier than this bound when using the rebalancing algorithm. This paper presents two techniques to improve the rebalancing algorithm. Applying these techniques, the algorithm can more uniformly diffuse flows. In addition, when these techniques are employed, the upper bound on the number of flows remains valid. The effectiveness of the two techniques is confirmed via computer simulations.**

Keywords—network; algorithm; routing; data center; packet.

I. INTRODUCTION

The importance of data center networks is obvious because most popular information services are provided via data centers. Therefore, it is essential to establish topologies for high performance data center networks. To satisfy this requirement, studies on data center networks have been performed based on several topologies including the Clos network [1], fat-tree [2], DCell [3], and BCube [4]. Of these, the Clos network is a particularly interesting topology because it can achieve high throughput for arbitrary traffic patterns. Therefore, various data center networks based on the Clos network topology have been implemented and operated [1][5]–[7].

A Clos network is a three-stage non-blocking switching network originally investigated by Charles Clos in 1953 [8]. In data center network applications, the network appears in the form of a folded Clos network (FCN). An FCN is essentially equivalent to a three-stage network; however, it is constructed by folding the corresponding three-stage Clos network at its center.

To apply an FCN to data center networks, the routing of a packet is important. Inadequate routing may cause load imbalances between the links. Such imbalances may cause traffic congestion and degrade the performance. Meanwhile, if the load is uniformly distributed between the links, an FCN can achieve high throughput by fully utilizing the bandwidth of every link.

As a routing method, several past studies [6][7][9] have employed the idea of forwarding a packet to a randomly selected route. This method is rational to some extent because it uniformly distributes the average number of flows between the links. However, with this method, the load on a given link may grow excessively large with a substantial probability. Consequently, due to heavily loaded links, traffic congestion may occur. Such congestion degrades the network performance. As pointed out in [10], this problem may become critical for big data applications, which require high bandwidth transmission. Therefore, it is important to develop a routing algorithm that diffuses the traffic load more uniformly than random routing.

Meanwhile, a routing algorithm for an FCN should be executable in a distributed manner to decrease the processing overhead and handle frequent route decisions. In addition, the algorithm should work without global information of the entire network to eliminate the communication overhead associated with gathering information. Routing can be performed on either a per-packet basis or a per-flow basis. This study examines a method based on per-flow routing because packet reordering is unavoidable for per-packet routing.

Reference [11] presented two distributed algorithms that diffuse flows in FCNs. Using computer simulations, it was shown that these methods more uniformly diffuse flows than random routing. These methods are called the rebalancing algorithm and the load sum algorithm. Of the two, the rebalancing algorithm works with information that is locally obtainable at the source switch of a flow. Meanwhile, the load sum algorithm is less practical due to the communication overhead between switches, even though it performs better with respect to load equality. Therefore, if the rebalancing algorithm is improved to more uniformly diffuse flows, a more practical and efficient algorithm will be obtained.

This paper presents techniques to improve the rebalancing algorithm with respect to load equality. These techniques are based on information that is locally obtainable at the source switch of a flow. The first technique modifies the algorithm to more evenly distribute the uplink loads. The second technique focuses on the fact that the algorithm has a process for scanning middle switch indices for routing and rerouting. Therefore, with the second method, the order of scanning the middle switch indices is determined so as to uniformly diffuse flows.

 ISBN: 978-1-61208-695-8

An advantage of the rebalancing algorithm is that an upper bound is theoretically derived for the number of flows on a link. When using the presented improvement techniques, this upper bound is not affected. Therefore, the worst-case link load is limited as in the case where these techniques are not applied. The effectiveness of the two techniques is confirmed via computer simulations.

The remainder of the paper is organized as follows. In Section II, FCN is explored. Section III reviews related work. Section IV explains the rebalancing algorithm, which is investigated for improvement. Two modification techniques are presented in Section V. The effectiveness of the techniques is evaluated in Section VI. Finally, Section VII concludes the paper.

II. FOLDED CLOS NETWORK

A Clos network is a three-stage switching network originally investigated by Charles Clos [8]. An FCN is essentially equivalent to a three-stage Clos network. However, an FCN is constructed by folding the three-stage network at its center. An example of an FCN is shown in Figure 1.

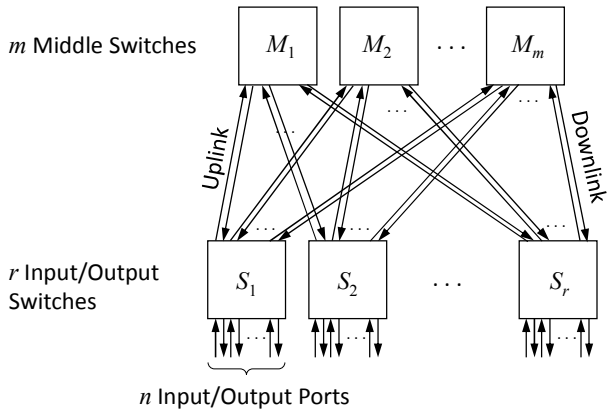

Figure 1. An example of a FCN.

As shown in Figure 1, an FCN is constructed from r input/output switches S_1, S_2, …, S_r that are connected by m middle switches M_1, M_2, …, M_m via links. Each middle switch is connected to every input/output switch via an uplink and a downlink. An uplink is set from an input/output switch to a middle switch, while a downlink is set in the reverse direction.

Because a middle switch is connected to every input/output switch, a packet can reach its destination switch from an arbitrary middle switch via a downlink. Therefore, the source switch can transmit a packet to the destination switch via any middle switch. However, the traffic load on an uplink or downlink depends on the routing at the source switch. If the routing is inadequate, traffic congestion occurs. This degrades the performance. Congestion is avoided if the traffic is evenly diffused between the uplinks and downlinks in an FCN. Therefore, it is important to establish a routing method that is executed at the source switch of a packet.

This paper assumes that routing is performed on a flow basis. A flow is a packet stream identified by a set of fields in the packet header. A frequently used field set is {*source address, destination address, protocol, source port, destination port*}, which is associated with an IP socket. Needless to say, other field sets can also be used as flow identifiers. If a fixed route is assigned to a flow, packet reordering does not occur. This is advantageous because packet reordering leads to throughput degradation. This paper considers the case where the FCN connects many hosts and processes via its $N = nr$ input/output ports. In this situation, many concurrent flows exist between ports.

III. RELATED WORK

A common idea for diffusing traffic in an FCN is to route a packet to a randomly selected middle switch. This idea, called Valiant load balancing, was employed in [6] and originally presented in [12]. Reference [9] explores the method of computing routing table entries from indices of switches and host identifiers. This is equivalent to randomly assigning route flows using the output of a hash function fed with switch indices and host identifiers. The architecture reported in [7] employs a per-packet adaptive routing mechanism as well as per-flow deterministic routing. For the deterministic routing, flows are diffused to random middle switches via a hash function fed with input ports and destinations.

The idea of randomly routing flows is rational to some extent because the average number of flows is balanced between the links. However, the worst-case load on a certain link can grow excessively large with substantial probability. This may cause traffic congestion and degrade the performance, e.g., via the packet latency or network throughput. The adaptive routing proposed in [10] may reduce this disadvantage of random routing. Initially, this method semi-randomly selects routes for the flows at the source switches. Then, the destination switches identify bad links, which are excessively loaded by this initial routing. Then, the destination switches notify the source switches of the flows, passing the bad links as bad flows. With this notification, the source switches to reroute the bad flows. The rerouting is repeated until there are no bad links. In [10], the convergence of the rerouting was evaluated using an analysis based on Markov chain models and computer simulation. Unfortunately, it is not clear theoretically how many times the flows should be rerouted to eliminate all the bad links in the worst case. It is also unclear whether bad links are definitively removed by the method of [10]. Due to these difficulties, this method may not be practical.

Reference [11] presented two flow-based routing methods that more uniformly diffuse flows than random routing. With these methods, a flow is routed (or rerouted) at its source switch in a distributed manner. One of these methods is the rebalancing algorithm, which runs using locally obtainable information. The other method is the load sum algorithm, which requires communication between the source and destination switches. The simulation results show that these methods both outperform random routing. It is also shown that the load sum algorithm more uniformly diffuses flows than the rebalancing algorithm. The simulation result reported in [11] shows that every flow equality metric is smaller for the

 ISBN: 978-1-61208-695-8

load sum algorithm than for the rebalancing algorithm. However, the load sum algorithm may be inefficient with respect to the communication overhead between switches, particularly in the case of short-duration flows. Namely, the traffic amount exchanged by a short-duration flow may become comparable to or smaller than that by the communication between switches. This is very inefficient. From this viewpoint, the rebalancing algorithm is likely more practical. By improving this method with respect to the uniformity of the flow diffusion, it is thought that the rebalancing algorithm will become advantageous.

IV. REBALANCING ALGORITHM

This paper presents techniques to improve the rebalancing algorithm presented in [11]. This algorithm is actually a packet stream version of the method shown in [13]. The algorithm assumes that the route of a newly generated flow is determined when its first packet arrives at the input switch. It may not be easy to strictly implement such a mechanism with currently available technologies. However, it is important to investigate potentially implementable methods that perform better than conventional routing.

This section explores some definitions, identifies local information, and details the algorithm.

A. Definitions

Throughout the paper, the following variables are employed.

- $F(i, j, k)$: the number of flows that go through a source switch S_i, a middle switch M_j, and a destination switch S_k $(1 \le i, k \le r, 1 \le j \le m)$.
- $U(i, j)$: the number of flows on the uplink set from S_i to M_j.
- $D(j, k)$: the number of flows on the downlink set from M_j to S_k.

Obviously, $U(i, j)$ and $D(j, k)$ are related to $F(i, j, k)$ as follows:

$$U(i,j) = \sum_{k=1}^{r} F(i,j,k), \quad (1)$$

$$D(j,k) = \sum_{i=1}^{r} F(i,j,k). \quad (2)$$

The algorithm is described using these variables.

B. Locally obtainable information

For data center network applications, flows may be generated and completed very frequently in the FCN. For such a situation, the routing of a flow should be executed in a distributed manner because the load offered by frequent route decisions will become excessively heavy for concentrated computations. In addition, it is impractical to perform communication between switches. This is because there may be very short flows that consist of only a few packets. As described in Section III, it is clearly inefficient to exchange packets between switches for the routing of such short flows. Therefore, the route of a flow should be decided at its source switch using locally obtainable information.

An input/output switch is able to obtain the headers of the packets, which arrive from its input port and are forwarded to middle switches. From these headers, the switch can identify the flows to which the packets belong. Because the switch decides the routes for the flows as the source, it can count how many flows go to each middle switch. Therefore, $U(i, j)$ can be managed at the source switch S_i. Moreover, the switch can also extract the destination switch of the flows from the packet headers. Using this information, the source switch S_i will be able to count $F(i, j, k)$ as well. However, $D(j, k)$ is not known at S_i because flows from switches other than S_i may enter the downlink to S_k.

Suppose that a new flow is generated and that its source switch is S_i. Then, assume that S_i can detect the arrival of a new flow. This is possible by comparing the flow identifiers to the routing table. It is also possible for S_i to detect the completion of a flow by timeout. Therefore, S_i can launch routing or rerouting processes at a flow arrival or completion.

C. Basic algorithm

The pseudocode for the rebalancing algorithm [11] is described in Figure 2.

```
Algorithm rebalancing
// α: a positive integer
1.  if a new flow arrives at switch Si and its destination is Sk then
2.      Find J such that F(i,J,k) ≤ F(i,j,k) for any j (1 ≤ j ≤ m);
3.      Route the flow to MJ;
4.      F(i, J, k):= F(i, J, k) + 1;
5.  end if
6.  if a flow passing through source Si and destination Sk is completed
    then
7.      Mx:= the middle switch that the completed flow was routed
        through;
8.      Find J such that F(i,j,k) ≤ F(i,J,k) for any j (1 ≤ j ≤ m);
9.      if F(i,J,k) − F(i,x,k) ≥ α then
10.         Find a flow that goes through MJ and Sk;
11.         Reroute the flow to Mx;
12.         F(i, J, k):= F(i, J, k) − 1;
13.     else F(i, x, k):= F(i, x, k) − 1;
14.     end if
15. end .
```

Figure 2. Rebalancing algorithm [11].

As shown in Figure 2, the algorithm decides the route for a new flow so as to decrease the difference between the $F(i, j, k)$s for a fixed pair of i and k. In addition, if the difference between the $F(i, j, k)$s exceeds a constant α by the flow completion, a flow is rerouted so as to decrease this difference. As a result, the rebalancing algorithm has the following property.

Property: *With the rebalancing algorithm,*

$$F(i,j,k) \le F(i,j',k) + \alpha, \quad (3)$$

for $1 \le j, j' \le m, 1 \le i, k \le r$.

Proof: This property is proved via induction on the flow arrival and completion events. Assume that (3) holds after the *K*-th event. Then, for a new flow arrival between S_i and S_k, $F(i, J, k)$ increases by 1 and the other $F(i, j, k)$s are unchanged. Meanwhile, $F(i, J, k)$ is not larger than $F(i, j, k)$ for an arbitrary value of j prior to the flow arrival. Therefore, $F(i, J, k)$ is not larger than $F(i, j, k) + 1$ for any j after the flow arrival. This means that (3) holds after the flow arrival event. If a flow that goes through S_i, M_x, S_k is completed, $F(i, x, k)$ decreases by 1. Therefore, if $F(i, J, k)$ is the maximum of the $F(i, j, k)$s, the difference between $F(i, J, k)$ and $F(i, x, k)$ may exceed α. However, if this happens, the algorithm reroutes a flow from M_J to M_x. Then, $F(i, x, k)$ does not change due to the flow completion and the maximum of the $F(i, j, k)$s does not increase. Therefore, (3) holds after the flow completion. Consequently, (3) is valid after the $(K + 1)$-th event. Under the initial state, no flows exist in the network, and therefore the $F(i, j, k)$s are 0 for all i, j, and k. This satisfies (3). Therefore, the property is proved. □

An advantage of the rebalancing algorithm is that an upper bound exists for the number of flows on an uplink or downlink. Let f_0 denote the maximum number of flows given to an input or output port. Then, as shown in [11],

$$U(i, j) \le \frac{nf_0 - \alpha(r-1)}{m} + \alpha(r-1), \text{ and} \quad (4)$$

$$D(j, k) \le \frac{nf_0 - \alpha(r-1)}{m} + \alpha(r-1) \quad (5)$$

The above equations are derived from (3). The proof of the bound is detailed in [11]. Due to this characteristic, it is assured that the load on a link does not grow extremely heavy.

In the rebalancing algorithm, the parameter α determines the frequency of rerouting as well as the uniformity of flow diffusion. If α is smaller, flows will be more frequently rerouted and more uniformly diffused. If α is large, rerouting never occurs. In this case, flows are diffused via the route decision when they arrive at their source switches. Unfortunately, if rerouting is omitted by setting α to a large value, the number of flows on a link can become considerably large in the worst case. However, simulation results show that the algorithm works well even without rerouting. Following [11], a rebalancing algorithm that omits the rerouting process is referred to as a "balancing algorithm" hereafter.

V. MODIFICATION TECHNIQUES

This section presents two modification techniques to improve the load equality of the rebalancing algorithm. These techniques add criteria to select the middle switch index J in steps 2 and 8 of the algorithm. However, they do not change the conditions that $F(i, J, k)$ and other $F(i, j, k)$s should satisfy. Therefore, (3) holds even if these techniques are applied. Consequently, the upper bound shown by (4) or (5) is unchanged by these techniques.

A. Uplink flow diffusion

The algorithm described in the previous section uses $F(i, j, k)$ and the events of flow arrival and completion in the local information. Therefore, of the available local information, $U(i, j)$ remains unused. Even though the rebalancing algorithm decreases the difference between the $F(i, j, k)$s for a particular pair of i and k, the uplink load $U(i, j)$ is not necessarily uniformly distributed. In step 2 of the rebalancing algorithm, the middle switch M_J is selected such that $F(i, J, k)$ will be the minimum of the $F(i, j, k)$s. In this process, there may be two or more candidates for J. Suppose that we select J from the candidates so that $U(i, J)$ will be the minimum of the candidates. Then, flows will be more uniformly distributed between the uplinks. This does not necessarily improve the load equality between the downlinks. However, the performance will at least be improved for the uplinks.

Similarly, flow diffusion via rerouting can also be modified using $U(i, j)$. In step 8 of the algorithm, M_J is selected such that $F(i, J, k)$ will be the maximum of the $F(i, j, k)$s. Suppose that there are two or more such indices J. Then, it is possible to use the index that maximizes $U(i, J)$. We refer to this modification using $U(i, j)$ as "modification 1."

B. Start index for scanning the middle switches

The order of searching for index J in steps 2 and 8 also affects the performance of the rebalancing algorithm. Assume that J is scanned in the order of 1, 2, …, m in step 2. Then, a smaller index is more likely to be selected as J. Therefore, $F(i, j, k)$ will be larger for a smaller index j with a high probability even though the differences between the $F(i, j, k)$s are bounded by α for fixed i and k. According to (1) and (2), this implies that $U(i, j)$ and $D(j, k)$ will also tend to be larger for a smaller value of j. To avoid this unbalance between $U(i, j)$ and $D(j, k)$, the scanning of J should start from a different index depending on k for a fixed value of i. Similarly, the start index should differ depending on i for a fixed value of k. In addition, the start index should be evenly distributed between 1, 2, …, m for different values of i or k. To satisfy this requirement, let us examine the following start index j_s:

$$j_s = (i + k)\lceil m / r \rceil. \quad (6)$$

In the above equation, the term $\lceil m / r \rceil$ is necessary to evenly distribute j_s between 1, 2, …, m for the case of $m \ge 2r$. In step 2 of the algorithm, the index was scanned in the order of $j_s, j_s + 1, j_s + 2, \ldots$, if the index reaches $m + 1$, it wraps to 1.

For step 8 of the algorithm, it was found from simulation results that the index should be started from $(j_s + m)$ mod m and then decreased. If the index reaches 0, it wraps to m. The reason for this scheme is explained as follows. This scheme aims to generate the situation where $F(i, j_s, k)$ is not less than $F(i, j_s + 1, k)$, $F(i, j_s + 1, k)$ is not less than $F(i, j_s + 2, k)$, and so on. To maintain this situation, it is favorable to select J from later elements of the sequence j_s, $j_s + 1$, $j_s + 2, \ldots$, $(j_s + m)$ mod m because $F(i, J, k)$ decreases due to rerouting.

The employment of the start index stated above is called "modification 2" hereafter.

 ISBN: 978-1-61208-695-8

VI. EVALUATION

The effectiveness of the improvements was evaluated using computer simulations. The simulations examined the rebalancing and balancing algorithms to which modifications 1 and 2 were applied. For comparison, the original rebalancing and balancing algorithms reported in [11] were also evaluated. In the rebalancing algorithm, the parameter α was set to 1.

In the simulations, the following two network models were employed:

- FCN1: $r = 48$, $m = n = 24$, and
- FCN2: $r = 24$, $m = n = 48$.

The parameters used for FCN1 are the same as those found in the model examined in [10]. Thus, it is considered that the parameters are adequate to simulate a realistic network. FCN2 was also examined to assess the algorithm behavior for a different topology with the same scale.

The degree of the load equality was estimated using the following metrics, which were also used in [11]:

- Maximum: the maximum number of flows in the links at a certain measurement time,
- Variance: the variance in the flow numbers in the links at a certain measurement time, and
- Bad links: the number of links, in which the number of flows exceeds a threshold C.

The threshold for bad links, C, was set to 105. This value was slightly larger than the average number of flows under the given traffic condition. The values of the above metrics will be smaller if the flows are more uniformly diffused.

A flow was generated by opening a socket between the hosts a and z. These hosts are connected to two randomly selected input/output switches. By opening a socket, two flows are generated for the direction from a to z as well as for the reverse direction.

Reference [6] reports that an average machine has ten concurrent flows in a real-world data center. By aggregating the traffic from 10 such machines, the average number of flows will be 100 for a port of each input/output switch. This situation was simulated by the following traffic model. The interval of opening sockets was randomly determined by an exponential distribution with an average of 0.001 s. The duration of a socket was also a random value according to an exponential distribution with an average of 57.6 s. For this traffic condition and the network models, the average number of flows in the network was estimated to be 100.

The sockets were opened 2×10^6 times. The metrics were measured every 1 s in the period from 401 s to 1900 s. The system was considered to be in equilibrium during this period. The averages of the metrics were computed from the measured data.

The simulation was performed by a custom event-driven simulation program. Thus, any existing simulation platform was not employed. The program was built using C language, and compiled by gcc 4.8.5. The simulation was performed on a Core i3/16GB RAM PC, which runs on CentOS 7.

The simulation result for the rebalancing algorithm and FCN1 is summarized in Table I. Table II shows the result for the balancing algorithm and FCN1.

TABLE I. RESULT FOR THE REBALANCING ALGORITHM AND FCN1.

Algorithms	Maximum	Variance	Bad Links
Original Version	111.678	10.838	122.841
Modification 1	111.085	7.348	66.934
Modification 2	109.942	9.254	92.203
Modifications 1 & 2	110.347	6.848	56.835

TABLE II. RESULT FOR THE BALANCING ALGORITHM AND FCN1.

Algorithms	Maximum	Variance	Bad Links
Original Version	113.537	14.796	187.919
Modification 1	112.617	9.666	102.452
Modification 2	110.869	11.413	126.949
Modifications 1 & 2	112.437	9.411	98.201

Tables I and II show that the load equality is successfully improved by modifications 1 and 2. As shown in the tables, every metric decreased when applying the modifications. In particular, modification 1 effectively improved the variance and bad links metrics. Therefore, this modification is effective even though it does not affect the equality between the $D(j, k)$s. The improvement due to modification 2 is not large in comparison to that due to modification 1. However, every metric also gets smaller when using modification 2. By applying both modifications 1 and 2, the best result was obtained for the variance and bad links metrics. Particularly, the improvement in the bad links metric is obvious. This implies that the number of flows is concentrated into a narrow range for most links.

Tables III and IV list the results for FCN2. Table III shows the case of the rebalancing algorithm, while Table IV shows the case of the balancing algorithm.

TABLE III. RESULT FOR THE REBALANCING ALGORITHM AND FCN2.

Algorithms	Maximum	Variance	Bad Links
Original Version	106.827	4.223	10.817
Modification 1	106.493	2.989	5.363
Modification 2	105.829	3.650	5.768
Modifications 1 & 2	106.014	2.767	3.319

TABLE IV. RESULT FOR THE BALANCING ALGORITHM AND FCN2.

Algorithms	Maximum	Variance	Bad Links
Original Version	107.517	5.049	19.756
Modification 1	107.138	3.544	9.947
Modification 2	106.255	4.176	9.428
Modifications 1 & 2	107.016	3.439	8.553

Tables III and IV show that every metric decreases due to the modifications for the case of FCN2 as well. This implies that the modifications will be effective in general for various

 ISBN: 978-1-61208-695-8

networks with different parameters. It also confirms that the definition of j_s is adequate for modification 2 because other definitions do not necessarily yield such a result.

In a comparison of the rebalancing and balancing algorithms, we find that the former is always superior to the latter for any case. However, the rerouting performed by the rebalancing algorithm may cause packet reordering, which may decrease the throughput. Meanwhile, the proposed modifications considerably improve the load equality of the balancing algorithm, which does not perform rerouting. Therefore, a practical solution is to use the balancing algorithm including the proposed modifications.

VII. Conclusion and Future Work

This paper investigated techniques to improve the rebalancing algorithm [11], which diffuses flows in an FCN. The first technique decreases the difference between the uplink loads by adding a criterion to decide on the middle switch used in the routing or rerouting processes. In addition, it was inferred that the load equality depends the order of the scanning of the middle switch indices. Based on this, the second technique decides the start index for scanning so as to balance the loads. The two techniques were applied to the rebalancing algorithm as well as the balancing algorithm and evaluated using computer simulations. Here, the balancing algorithm is a version of the rebalancing algorithm that is modified to omit the rerouting process. The results show that the presented techniques successfully improve the load equality.

Further study is necessary in the future to determine how the load equality provided by the presented techniques affects the packet-level performances such as the packet latency. The implementation of these techniques is also an important future work. Nevertheless, it is concluded that the rebalancing and balancing algorithms become more practical when employing the presented techniques.

References

[1] F. Hassen and L. Mhamdi, “High-capacity Clos-network switch for data center networks,” in proc. ICC 2017, paper NGNI07-1, pp. 1–7, Paris, France, May 2017.

[2] Z. Guo and Y. Yang, “On Nonblocking Multicast Fat-Tree Data Center Networks with Server Redundancy,” IEEE Trans. on Computers, 64, 4, pp. 1058–1073, Apr. 2014.

[3] C. Guo et al., “DCell: a scalable and fault-tolerant network structure for data centers,” in proc. ACM SIGCOMM ’08, pp. 75–86, Seattle, WA, USA, Aug. 2008.

[4] C. Guo et al., “BCube: a high performance, server-centric network architecture for modular data centers,” in proc. ACM SIGCOMM ’09, pp. 63–74, Barcelona, Spain, Aug. 2009.

[5] N. Farrington and A. Andreyev, “Facebook’s data center network architecture,” in proc. 2013 Optical Interconnects Conference, pp. 49–50, Santa Fe, NM, USA, May 2013.

[6] A. Greenberg et al., “VL2: a scalable and flexible data center network,” Communications of the ACM, 54, 3, pp. 95–104, Mar. 2011.

[7] S. Scott, D. Abts, J. Kim, and W.J. Dally, “The BlackWidow high-radix Clos network,” in proc. ISCA ’06, pp. 16–28, Boston, MA, USA, June 2006.

[8] C. Clos, “A study of nonblocking switching networks,” Bell System Technical Journal, 32, 2, pp. 406–424, Mar. 1953.

[9] M. Al-Fares, A. Loukissas, and A. Vahdat, “A scalable, commodity data center network architecture,” in proc. SIGCOMM ’08, pp. 63–74, Seattle, WA, USA, Aug. 2008.

[10] E. Zahavi, I. Keslassy, and A. Kolodny, “Distributed adaptive routing for big-data applications running on data center networks,” in proc. ANCS ’12, pp. 99–110, Austin, Tx, USA, Oct. 2012.

[11] S. Ohta, “Flow diffusion algorithms based on local and semi-local information for folded Clos networks,” in proc. ICESS 2018, pp. 46–54, Takamatsu, Japan, Nov. 2018.

[12] L. G. Valiant, “A scheme for fast parallel communication,” SIAM J. Computing, 11, 2, pp. 350–361, May 1982.

[13] S. Ohta, "A simple control algorithm for rearrangeable switching networks with time division multiplexed links,” IEEE J. on Selected Areas in Communications, SAC-5, 8, pp.1302–1308, Oct. 1987.

 ISBN: 978-1-61208-695-8