# Implementation and Experimental Evaluation of Flow Diffusion Algorithms for Folded Clos Networks

SATORU OHTA
Department of Information Systems Engineering, Faculty of Engineering
Toyama Prefectural University
5180 Kurokawa, Imizu, Toyama
JAPAN

DAICHI MIYAMOTO
Department of Information Systems Engineering, Faculty of Engineering
Toyama Prefectural University
5180 Kurokawa, Imizu, Toyama
JAPAN

*Abstract:* Folded Clos networks (FCNs) play a significant role in constructing high-performance data center networks. To fully utilize the high bandwidth of FCNs, traffic load must be uniformly diffused between links. To achieve this, a method called the rebalancing algorithm was proposed. Previous studies have demonstrated the effectiveness of this algorithm through theoretical analyses and computer simulations. However, its practical feasibility has not yet been elucidated. Moreover, its performance has not been confirmed in a physical experimental environment. This study demonstrates the implementation of the rebalancing algorithm on switches built on PCs using software tools supported by the Linux OS. This confirms that it is relatively easy to implement the rebalancing algorithm. In addition, the performance of the implemented rebalancing algorithm was evaluated and compared with that of its simplified version, called the balancing algorithm, as well as a conventional method, through experiments. According to the results, the rebalancing algorithm is more advantageous in avoiding traffic congestion for heavy traffic than the conventional method. It was also confirmed that the balancing algorithm performs as effectively as the rebalancing algorithm, with less processing load.

*Key-Words:* - data center network; load balancing; switching network; routing

Received: March 28, 2022. Revised: October 23, 2022. Accepted: November 17, 2022. Published: December 31, 2022.

## 1 Introduction

This paper details the implementation and experiments of flow diffusion algorithms for Clos networks by extending our previous conference paper, [1]. Clos networks have been used as a topology for data center networks [2-4]. Because the performance of a data center network significantly affects the quality of information services, it is essential to investigate the Clos network topology. For the data center network application, the topology is advantageous owing to its scalability and high throughput.

A Clos network was originally presented as a three-stage switching network by Charles Clos, [5]. The topology used in data center networks is obtained by folding the original three-stage structure in its center. Thus, the topology is referred to as a folded Clos network (FCN) hereafter.

This study assumes the data center network application of an FCN. An FCN is constructed by connecting multiple input/output and middle switches through duplex links. Through this network configuration, hosts exchange data packets. These hosts are attached to input/output switches. A packet sent from a host is delivered to its destination host through a source-side input/output switch, a certain middle switch, and a destination-side input/output switch.

To fully utilize the bandwidth of an FCN, packet traffic must be uniformly distributed over the links. A traditional way of doing this is random routing, which randomly diffuses packets over usable routes. As a more sophisticated method, the rebalancing algorithm, [6-8], was proposed. The algorithm diffuses traffic on a per-flow basis. Here, a flow is defined as a packet stream identified by a set of fields in the packet header, [9],. The algorithm

manages the number of flows passing usable routes for each destination switch. Then, the route of a newly generated flow is chosen to equalize the number of flows for each route. When a certain flow is completed, another existing flow may be rerouted to minimize the difference among flows assigned to routes. A feature of this method is that the number of flows passing a link is mathematically upper-bounded. This means that flow congestion is theoretically avoided by using the rebalancing algorithm. The algorithm uses information that is locally obtainable at each switch. This leads to the simplicity of implementing the method as it is unnecessary to manage the entire network or exchange control information between switches.

Previous studies [6-8] on the rebalancing algorithm and related methods have focused on the theoretical aspect and evaluation through computer simulations. However, the practical feasibility of the rebalancing algorithm is unclear in previous studies. To confirm its feasibility, it is important to implement and operate the algorithm in a real-world system. In addition, the effectiveness of the algorithm must be more concretely confirmed through an experimental evaluation in a physical network environment.

The contributions of this study are as follows. First, the study explores how to implement the rebalancing algorithm on PC (personal computer) switches using common software tools supported by the Linux OS. This confirms that the algorithm is relatively easy to implement. Second, the implemented rebalancing algorithm is tested on an experimental FCN. The rebalancing algorithm is compared with conventional random routing through experiments. The results indicate the advantage of the rebalancing algorithm in avoiding traffic congestion. In addition to the results of our previous study [1], this study presents the following new experimental findings.

- The performance of each traffic diffusion algorithm is evaluated for multiple traffic datasets, which differ in the flow size distribution.
- The balancing algorithm, a simplified version of the rebalancing algorithm, is evaluated by experiments, along with the rebalancing algorithm and random routing. It is shown that the balancing algorithm performs as effectively as the rebalancing algorithm with less processing load.

The remainder of this paper is organized as follows. Section 2 reviews previous related studies. As preliminaries, Section 3 explores FCNs and the rebalancing algorithm. The implementation of the rebalancing algorithm is detailed in Section 4. Then, Section 5 presents the setting and results of the experiment. Finally, Section 6 concludes the paper.

## 2 Related Work

In 1953, Charles Clos presented a three-stage switching network [5], which is known as the Clos network. Clos networks have been investigated and used for various important applications, such as telephone switching [5], cross-connect systems [10, 11], systems-on-chip [12, 13], and data center networks [2-4, 14, 15].

An FCN is configured by folding a three-stage Clos network in its center stage. This configuration is also presented in Clos's study [5] as a triangular array. FCNs have been considered for various applications, including systems-on-chip [12, 13] and data center networks [2-4, 6-8, 14, 15].

In a packet-switching environment such as data center networks, the traffic load offered to an FCN must be evenly distributed among links to avoid congestion. In the architecture investigated by Al-Fares et al. ,[14], packets are forwarded depending on the destination host identifiers. If many equally loaded hosts are operated, this method evenly distributes traffic load in the network. However, this condition does not always hold.

As an alternative, random routing is a simple method of diffusing network load. This method randomly selects a middle switch, to which the packets of a flow are forwarded. Greenberg et al. [4] used random routing, referred to as Valiant Load Balancing [16], in their FCN. Random routing is advantageous in terms of its ease of implementation.

The average number of flows in a link is evenly distributed via random routing. However, random routing may significantly increase the number of flows on certain links with a substantial probability. Therefore, congestion is not sufficiently avoidable with random routing in some cases.

Zahavi et al., [15], proposed an alternative to random routing for FCNs. In their method, routes for flows are first semi-randomly selected at source switches. Then, destination switches identify excessively loaded links. Next, destination switches notify the source switches of flows passing the excessively loaded links. With this notification, source switches reroute those flows that cause excessive load. This rerouting process is repeated until there are no excessively loaded links. As a result, congestion is removed. However, for their method, it is not theoretically known how many

times flows should be rerouted for the deletion of excessive link loads in the worst case. It is also unclear whether excessive link loads are certainly removed by this method. In conclusion, this method may not be practical.

Ohta, [6-8], presented an alternative: the rebalancing algorithm. With this method, the number of flows passing a link does not exceed a theoretically derived upper bound. This means that the traffic load does not excessively grow heavy on any links. The effectiveness of this method has been confirmed through a flow-level computer simulation for the uniformness of flows in links [6-8]. In [17], a packet-level computer simulation was performed to evaluate the TCP throughput achieved by several algorithms, including a simplified version of the rebalancing algorithm. The simplified version differs from the original rebalancing algorithm in that it omits the rerouting process. The results of this simulation indicate that the flow concentration is avoided and the ratio of flows with decreased throughput is reduced by the simplified version.

Although the rebalancing algorithm is a promising technique, two technical problems remain. First, it is unknown whether the algorithm is practically implementable on real-world switches. The second problem is that its performance advantage has not been tested in an experimental environment, where actual packets are exchanged in a physical FCN. Thus, it is necessary to implement and evaluate the algorithm through experiments.

# 3 Preliminaries

## 3.1 Folded Clos Network

Fig.1 presents an example of an FCN. As shown in Fig.1, an FCN is composed of $r$ input/output switches and $m$ middle switches ($r$, $m > 1$). A middle switch is connected to each input/output switch through a duplex link. An input/output switch has ports, to which hosts are connected.

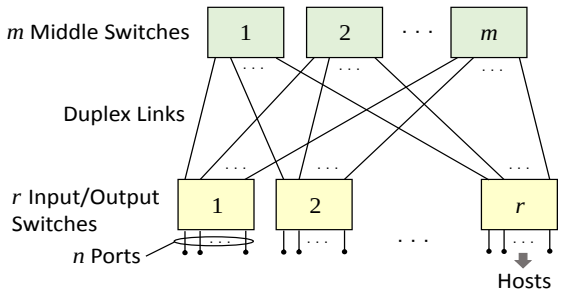


Fig.1 An example of FCN.

For the data center network application, packets are exchanged between hosts through an FCN. As shown in Fig.1, a packet can be forwarded from a source to a destination through one of the $m$ middle switches. In other words, the network has $m$ different routes, which can be used to forward the packet between input/output switches. Thus, a source-side input/output switch must select one of the $m$ routes to forward a packet. If this routing is inadequate, the traffic load congests some links. This traffic congestion degrades performance.

## 3.2 The Rebalancing Algorithm

The rebalancing algorithm [6-8] is executed independently at each input/output switch to determine the route of a flow using locally obtainable information in an FCN. Assume that input/output switches are indexed as 1, 2, …, $r$, whereas middle switches are indexed as 1, 2, …, $m$. At each input/output switch $i$ $(1 \le i \le r)$, the algorithm maintains the number of flows that go from $i$ to another input/output switch $k$ $(1 \le k \le r)$ in each route. Because a route is identified by a middle switch, the number of routes is $m$ for a single-destination input/output switch. The number of input/output switches other than $i$ is $r - 1$. Therefore, switch $i$ manages $m(r - 1)$ routes.

Let $F(i, j, k)$ denote the number of flows that travel from $i$ to the destination input/output switch $k$ through the middle switch $j$ $(1 \le j \le m)$. Then, at an input/output switch $i$, the algorithm selects a middle switch $j$ for a new flow so as to minimize the variation among $F(i, 1, k)$, $F(i, 2, k)$, …, $F(i, m, k)$. The middle switch index $j$ of this flow must be recorded for the process performed at flow completion. When a certain flow from $i$ to $k$ through $j^*$ is completed, $F(i, j^*, k)$ decreases by 1. This change may increase the variation among $F(i, j, k)$'s. If this happens, a certain flow from $i$ to $k$ is rerouted to the middle switch $j^*$ to offset the increased variation. Through these processes, the variation among $F(i, j, k)$'s is minimized. As a result, the total number of flows is theoretically upper-bounded for every link, [7]. The process is performed at an input/output switch $i$ by identifying the destination switch $k$ of a new flow and the routes of existing flows. Thus, the algorithm is executable using locally obtainable information. This means that it is not necessary to manage global network information or to exchange any control information between switches.

# 4 Implementation

We implemented the rebalancing algorithm as a program that runs on a Linux PC. The PC is equipped with a multiport network interface and

functions as a Layer 3 switch. The implementation methodology is based on the flow identification by packet capture as well as the policy routing technique supported by the Linux OS. In the implementation, a flow is defined as a TCP connection, which is identified by the five-tuples of protocol, source/destination addresses, and source/destination ports in a packet header. This means that the program is dedicated to the distribution of TCP traffic across the network. Because many applications are provided on the TCP, it is rational to focus on TCP flows as the first step of the study. To implement the rebalancing algorithm for TCP flows, it is important to address the following points:

- How to detect the generation and completion of a flow.
- How to find the index (= $k$) of a destination switch from the packets of a flow.
- How to route packets of a flow to a middle switch, which is selected by the algorithm and generally not the same as that for a different flow with the same destination.

The implemented program addressed these points by capturing packets using the *pcap* library, [18] and the Linux policy routing mechanism, [19]. From the captured packets, the program detects the start and completion of a flow. The destination switch of a flow can be identified from the destination address of a captured packet by appropriately associating the host-side network addresses to the input/output switch index. The Linux policy routing mechanism enables the program to forward packets to the middle switch selected by the algorithm, depending on not only the destination address but also other flow identifier elements, such as the source address and port numbers.

The program uses the *pcap* library to capture either TCP SYN or FIN packets, which are destined for hosts connected to other input/output switches. Through an SYN packet arrival, the program detects the start of a new flow.

From the destination address of an SYN packet, the program identifies the destination switch $k$ through the association between the destination network address and switch index $k$. In our implementation, the host-side network addresses of an input/output switch $k$ are set to the range of $192.168.p + qk.0/24$ to $192.168.p + q(k + 1) – 1.0/24$, where $p$ and $q$ represent integers such that $p + q(m + 1) < 256$. Then, if the destination address is $192.168.x.y$ $(0 \le x, y \le 255)$, switch index $k$ is immediately computable by $(x – p) / q$.

When an SYN packet is detected at an input/output switch $i$ and its destination switch $k$ is identified as mentioned above, the rebalancing algorithm is executed to determine a middle switch $j$ from $F(i, 1, k)$, $F(i, 2, k)$,…, $F(i, m, k)$. Furthermore, the route for the flow is set to forward packets to $j$ and updates $F(i, j, k)$. The program of each input/output switch $i$ manages $F(i, j, k)$ using a two-dimensional array indexed by $j$ and $k$. It is unnecessary for switch $i$ to know the $F(i', j, k)$ of other switches $i'$ $(i' \neq i)$. The flow identifier of a new flow, index $j$ of the selected middle switch, and index $k$ of the destination switch are stored in a hash table $T$. The information recorded in $T$ is necessary for flow completion and rerouting. The flow identifier is also stored in a list $L$, which is used in the flow rerouting process.

The completion of a flow is detected by the receipt of a FIN packet. The program extracts the flow identifier from the packet header and searches for its middle switch $j$ and destination switch $k$ from table $T$. Then, the hash table entry is deleted. Moreover, the routing (i.e., packet marking rule, explained below) for that flow is removed from the "*mangle*" table through the "*iptables*" command. This removal of the marking rule is performed one or more seconds later after the FIN detection to forward the ACK packet for the FIN packet from the remote host. With the completion of the flow, $F(i, j, k)$ is updated. The rerouting of an existing flow is performed if necessary. The flow to be rerouted is selected from the flow list $L$. The list $L$ is periodically updated by removing completed flows.

Routing is performed through $m$ routing tables $rt_1$, $rt_2$, …, $rt_m$, where $rt_j$ is set to route packets to a middle switch $j$. Each table is associated with a rule, which sets packets marked with $j$ to refer to $rt_j$. The rules are configured through the Linux "*ip rule*" command. The rule-setting process is performed in the initialization phase of the program. Provided that the rules are properly set, assume that the algorithm selects a flow to be routed to the middle switch $j$. In this situation, the program marks the packets of the flow with $j$. This marking is achieved by applying the "*iptables*" command to the "*mangle*" table, with the "*--set-mark*" option. Through these operations, the packets of the flow are successfully routed to the middle switch $j$.

A drawback of this scheme is that the SYN packet is not marked. Thus, it is dropped because a proper table is not found for the packet. To avoid this, the program sets every SYN packet to be sent to a default route. In addition, RST packets are also

forwarded to this default route. Thus, these packets are not distributed by the algorithm. The load imbalance caused by the routing of the small-sized SYN and RST packets is negligible because the route determined by the algorithm is used by most packets.

The functions of the implemented program are illustrated in Fig.2. The program was coded with the C language and developed on CentOS 7.9.2009, gcc 4.8.5, libpcap 1.5.3, and iptables 1.4.21. In the experiment, the program was run on a PC switch that has Ryzen 5 3400 CPU and 16GB RAM.

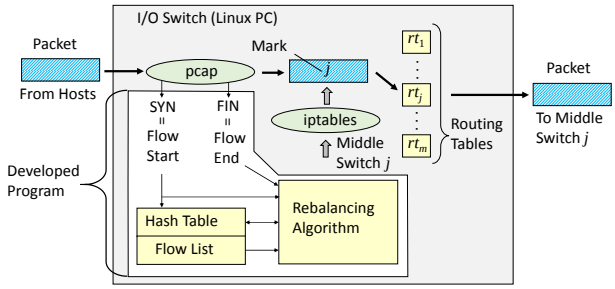


Fig.2 Schematic of the implementation of the algorithm.

## 5 Experiments

The implemented algorithm was tested on a small-sized experimental FCN, as shown in Fig.3. As illustrated in the figure, the network consists of three middle switches and three input/output switches. In the network, each switch is a Linux PC, to which a multiport network interface is attached. A middle switch is connected to each input/output switch through 1 Gb/s Ethernet. Each input/output switch is connected to a Linux host through 10 Gb/s Ethernet. Thus, the inner links of the FCN are bottlenecks. This setting is essential to see the difference among the traffic load uniformity achieved by the algorithms without being affected by the host-side bandwidth limitation.

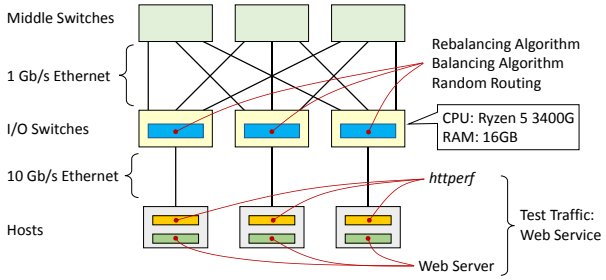


Fig.3 Experimental network.

The performance of the experimental network was measured for the following flow diffusion methods.

- Rebalancing algorithm [7]
- Balancing algorithm [17]
- Conventional random routing

Among these, the balancing algorithm is a simplified version of the rebalancing algorithm. The rerouting process in the rebalancing algorithm is eliminated in the balancing algorithm even though flow routes are determined in the same procedure as the rebalancing algorithm. Every method was implemented according to the implementation scheme described in the previous section.

The flow diffusion methods were evaluated through TCP connections generated by a web service. To do this, the *Apache* web server was run on hosts, whereas the web benchmark tool *httperf* [20] was also executed on hosts to request page data and measure the characteristics of the generated TCP connections.

Among various metrics provided by *httperf*, this study uses the average time of a TCP connection as the metric of traffic uniformity, i.e., congestion level. Namely, the metric is the time taken from a user request to download completion. Suppose that congestion occurs in a certain link due to load imbalance. Then, the throughput will decrease for flows passing through the congested link. The decreased throughput increases the TCP connection time to send the page data. Thus, the congestion level is estimated from the TCP connection time.

In the operation of a flow diffusion method, the computational load on the CPU of an input/output switch may be an issue. To evaluate this point, the CPU load was measured using the *top* command at three input/output switches. In addition, the distribution of flow throughputs was measured by capturing packets at hosts. For this purpose, custom throughput measurement software was developed. The program was executed at each host.

As page data, the following two datasets were used.

- Dataset #1: 1000 files, where the size of a file is randomly determined via the Pareto distribution
- Dataset #2: 1000 files, where the size of a file is constant. The file size was set equal to the average file size of dataset #1.

Predictably, various sizes of data brought by TCP connections will coexist in a real-world FCN. Dataset #1 was used to emulate such a situation through the web service. Particularly, the dataset causes very different connection times of flows by using the Pareto distribution, which is known as a

heavy-tailed distribution [21]. The probability density function $f(x)$ and the average $E[X]$ of the Pareto distribution are as follows:

$$f(x) = \frac{\alpha}{k}\left(\frac{k}{x}\right)^{\alpha+1}, \quad E[X] = \frac{\alpha}{\alpha - 1}k .$$

In dataset #1, the parameter $\alpha$ was set to 1.5, and the average size of a file was set to 30 MB. For these parameters, files were generated as follows. First, the file size, denoted by $N$, is determined using the inverse transform method. Then, $N$ ASCII characters are written to the file. By repeating this process, file sizes have the Pareto distribution.

It is interesting to test the algorithms for dataset #2. By comparing the results for datasets #1 and #2, the effects of the difference between flow size distributions on the performance of each algorithm will be clarified.

The experiment was conducted in the following setting. In the configuration presented in Fig.3, the web server program, *Apache*, runs on every host. On each host, two *httperf* processes were simultaneously launched. Each process requests page data from one of the other two hosts. Thus, traffic is exchanged between every host pair, whereas six instances of *httperf* outputs are obtained for a single measurement. Each *httperf* process generates 10,000 TCP connections. The interval between requests for page data is exponentially distributed. The average of the interval was set between 0.1818 and 0.6667 s, which means the request rate varies from 2 to 5.5 requests/s for a single *httperf* process. This means that the total bit rate of the traffic for the network ranges from 2.88 to 7.92 Gb/s. For each average interval value, the measurements were repeated five times. Consequently, 6 × 5 = 30 values of the connection time were obtained for a single value of the request rate. Then, the average of these 30 values was computed.

The results for dataset #1 are presented in Fig.4. The x-axis is the request rate of each *httperf* process, whereas the y-axis is the average connection time. As presented in the figure, the connection time increases for larger request rates due to decreased throughput. The figure indicates that the rebalancing algorithm outperforms random routing. For example, when the request rate is 5.5 requests/s, the average connection time of random routing is twice that of the rebalancing algorithm. The smaller connection time of the rebalancing algorithm suggests its effectiveness in avoiding congestion. Thus, the result confirms the advantage of the rebalancing algorithm over the conventional technique.

The performance of the balancing algorithm is very comparable to that of the rebalancing algorithm. Thus, the balancing algorithm also outperforms conventional random routing. The rebalancing algorithm slightly outperforms the balancing algorithm when the request rate is 5.5 requests/s. This suggests that the rerouting process of the rebalancing algorithm improves the performance for heavy loads. However, the performance difference between the balancing and rebalancing algorithms is almost negligible in most cases. Therefore, it is implied that the effectiveness of the rerouting process is not significant, except in the case of extremely heavy loads.

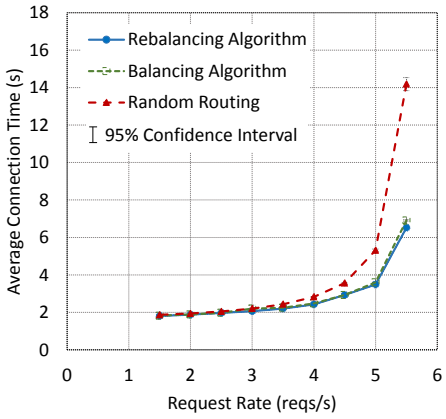


Fig.4 Relationship between the request rate and the average connection time of the algorithms for dataset #1.

Fig.5 depicts the results for dataset #2. The x-axis represents the request rate of each *httperf* process, whereas the y-axis represents the average connection time. In the figure, the characteristic of each method does not significantly differ from that for dataset #1. When the rate is less than 5 requests/s, the performance difference between the algorithms is smaller than for the case of dataset #1. However, when the rate is 5.5 requests/s, the connection time increases significantly more for random routing than for the rebalancing and balancing algorithms, similar to the case of dataset #1. Therefore, this result implies that the rebalancing and balancing algorithms more effectively diffuse traffic than conventional random routing, independent of the flow size distribution.

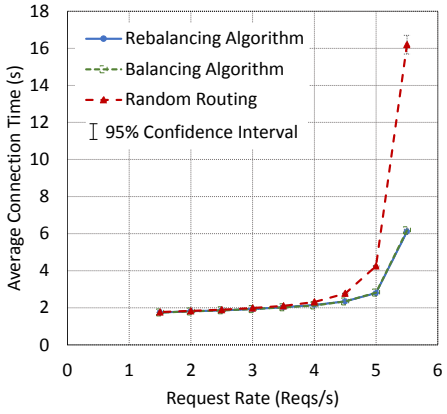


Fig.5 Relationship between the request rate and the average connection time of the algorithms for dataset #2.

Figs.6 and 7 compare the distribution of flow throughputs. Fig.6 shows the case of dataset #1, whereas Fig.7 shows the case of dataset #2. In these figures, the request rate was set at 5.5 requests/s for a single *httperf* process. In these figures, the x-axis represents the throughput of a flow, whereas the y-axis represents the cumulative percentage of the number of flows. Both figures indicate that the portion of flows with decreased throughput is significantly smaller for the rebalancing and balancing algorithms than for random routing. For random routing, Fig.6 shows that 60% of flows have throughputs that are smaller than 25 Mb/s. When the rebalancing algorithm is used, the ratio of flows with such decreased throughputs is as small as 1.1%. For the balancing algorithm, the ratio of flows with throughputs smaller than 25 Mb/s is 2.5%, which is larger than that for the rebalancing algorithm and significantly smaller than that for random routing. For the result shown in Fig.7, the throughputs of 77% of flows are smaller than 25 Mb/s when random routing is used. Meanwhile, the ratio of such flows is 0.3% for the rebalancing algorithm and 1.2% for the balancing algorithm. From these results, the advantage of the rebalancing and balancing algorithms in reducing the number of degraded flows is obvious.

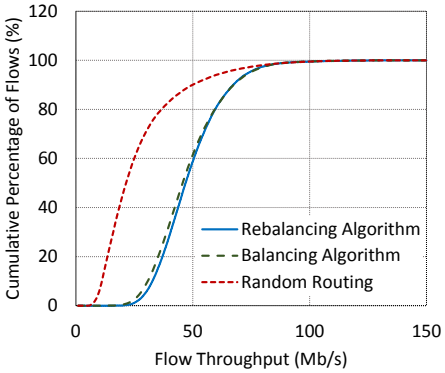


Fig.6 The cumulative percentage of the number of flows versus throughput for dataset #1.

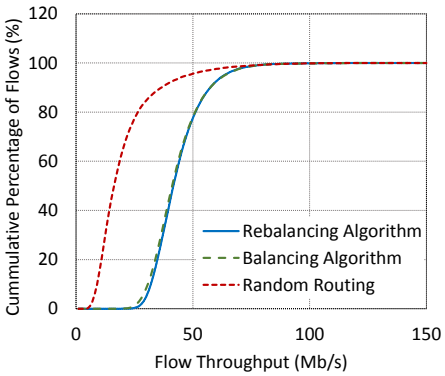


Fig.7 The cumulative percentage of the number of flows versus throughput for dataset #1.

Fig.8 compares the utilization of a single CPU thread for the rebalancing algorithm, the balancing algorithm, and random routing. The figure shows the result when using dataset #1 and setting the request rate at 5 requests/s for a single httperf process. The CPU utilization is the average of 300 values, measured by the *top* command every 3 s. As depicted in the figure, the processing load of the rebalancing algorithm is larger than that of random routing. This is because the computation of the rebalancing algorithm is more complex for the rerouting of flows. The processing load of the balancing algorithm almost equals that of random routing.

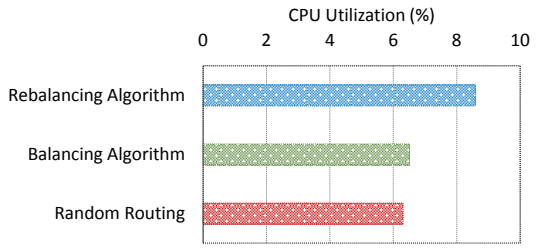


Fig.8 CPU utilization of input/output switch for the rebalancing algorithm, the balancing algorithm, and random routing.

As depicted in Figs.4–7, the balancing algorithm outperforms random routing in terms of connection time and flow throughput. Meanwhile, its processing load is smaller than that of the rebalancing algorithm and as small as that of random routing. Therefore, it is concluded that the balancing algorithm is a more practical method because of its performance and low processing load.

# 6 Conclusion

This study presented the implementation and experimental results of the rebalancing algorithm, which evenly diffuses traffic load in an FCN. The results are summarized as follows:

- The practical feasibility of the rebalancing algorithm was verified through implementation and experiments.
- The rebalancing algorithm is implementable using commonly available software.
- The experimental result clarifies that the rebalancing algorithm outperforms conventional random routing for heavy loads. For heavy loads, we observed that the ratio of flows with decreased throughputs is significantly smaller for the rebalancing algorithm than for random routing.
- The performance of the balancing algorithm, a simplified version of the rebalancing algorithm, is very comparable to that of the rebalancing algorithm. Considering its lighter processing load, we conclude that the balancing algorithm is a promising technique.

It is expected that these results will contribute to the future progress of data center networks. The techniques investigated in this paper will practically enhance the performance of those networks.

The implementation presented in this study assumes that the transport layer protocol of every flow is TCP, and a flow is defined as a TCP connection. Because of this assumption, the start and completion of a flow are easily detectable through the SYN and FIN flags in packet headers. This method does not apply to other transport layer protocols. Therefore, further study is necessary to clarify how to efficiently implement the rebalancing algorithm independently of transport protocols.

*References:*

[1] S. Ohta and D. Miyamoto, Implementation and experimental evaluation of the rebalancing algorithm for folded Clos networks, *In Proc. of ITNAC 2022*, pp. 312-315, Wellington, New Zealand, Dec. 2022.

[2] N. Farrington and A. Andreyev, Facebook's data center network architecture, *In Proc. OI 2013*, pp. 49-50, Santa Fe, NM, USA, May 2013.

[3] A. Singh et al., Jupiter rising: a decade of Clos topologies and centralized control in Google's datacenter network, *In Proc. 2015 ACM Conference on Special Interest Group on Data Communication*, pp. 183-197, London, UK, Aug. 2015.

[4] A. Greenberg et al., VL2: a scalable and flexible data center network, *Communications of the ACM*, Vol. 54, No.3, pp. 95-104, Mar. 2011.

[5] C. Clos, A study of nonblocking switching networks, *Bell System Technical Journal*, Vol. 32, No. 2, pp. 406-424, Mar. 1953.

[6] S. Ohta, Flow diffusion algorithms based on local and semi-local information for folded Clos networks, *In Proc. ICESS2018*, pp. 46-54, Takamatsu, Japan, Nov. 2018.

[7] S. Ohta, Flow diffusion algorithms for folded Clos networks, *IEEJ Trans. on Electronics, Information and Systems*, Vol. 139, No. 11, pp. 1224-1233, Nov. 2019.

[8] S. Ohta, Techniques for enhancing the rebalancing algorithm for folded Clos networks, *IARIA Int. J. on Advances in Networks and Services*, Vol. 12, No. 3&4, pp. 69-80, Dec. 2019.

[9] C. Estan, G. Varghese, and M. Fisk, Bitmap algorithms for counting active flows on high speed links, *In Proc. IMC '03*, pp. 153-166, Miami Beach, FL, USA, Oct. 2003.

[10] N. Fujii, Application of a rearrangement algorithm for digital cross-connect system control, *In Proc. IEEE INFOCOM '89*, pp. 228-233, Ottawa, Canada, 23 Apr. 1989.

[11] M.K. Panda, T. Venkatesh, V. Sridhar, and Y.N. Singh, Architecture for a class of scalable optical cross-connects, *In Proc. BROADNETS '04*, San Jose, CA, USA, Oct. 2004.
[12] A. Zia, S. Kannan, G. Rose, and H.J. Chao, Highly-scalable 3D Clos NOC for many-core CMPs, *In Proc. NEWCAS2010*, pp. 229-232, Montreal, QC, Canada, June 2010.
[13] A. Joshi et al., Silicon-photonic Clos networks for global on-chip communication, *In Proc. NOCS '09*, San Diego, CA, USA, May 2009.
[14] M. Al-Fares, A. Loukissas, and A. Vahdat, A scalable, commodity data center network architecture, *In Proc. ACM SIGCOMM '08*, pp. 63-74, Seattle, WA, USA, Aug. 2008.
[15] 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.
[16] L.G. Valiant, A scheme for fast parallel communication, *SIAM Journal on Computing*, Vol. 11, No.2, pp. 350-361, May 1982.
[17] S. Ohta, TCP throughput achieved by a folded Clos network controlled by different flow diffusion algorithms, *Int. J. of Information and Electronics Engineering*, Vol. 10, No. 1, pp. 16-21, Mar. 2020.
[18] T. Carstens, Programming with pcap. *online:* https://www.tcpdump.org/pcap.html (accessed on 19 Dec. 2022).
[19] B. Hubert et al, Linux advanced routing & traffic control HOWTO, *online:* https://tldp.org/HOWTO/Adv-Routing-HOWTO/index.html (accessed on 19 Dec. 2022).
[20] D. Mosberger and T. Jin, httperf – A tool for measuring web server performance, *ACM SIGMETRICS Performance Evaluation Review*, Vol. 26, No. 3, pp. 31-37, Dec. 1998.
[21] W. Stallings, *High Speed Networks – TCP/IP and ATM Design Principles*, Prentice Hall, Upper Saddle River, NJ, USA, pp. 191-192, 1998.

**Contribution of Individual Authors to the Creation of a Scientific Article (Ghostwriting Policy)**

Satoru Ohta planned the methodology, implemented the software, and executed the experiments.
Daichi Miyamoto built the experimental environment and collected a part of data.

**Sources of Funding for Research Presented in a Scientific Article or Scientific Article Itself**
This work was supported by JSPS KAKENHI Grant Number JP19K11928.

**Conflict of Interest**
The authors have no conflicts of interest to declare that are relevant to the content of this article.