PAPER *Special Section on Parallel, Distributed, and Reconfigurable Computing, and Networking*

# Traffic-Independent Multi-Path Routing for High-Throughput Data Center Networks

**Ryuta KAWANO**[†a)], ***Member***, **Ryota YASUDO**[††], ***Nonmember***, **Hiroki MATSUTANI**[†††], ***Member***, **Michihiro KOIBUCHI**[††††], ***Senior Member***, ***and*** **Hideharu AMANO**[†††], ***Fellow***

**SUMMARY** Network throughput has become an important issue for big-data analysis on Warehouse-Scale Computing (WSC) systems. It has been reported that randomly-connected inter-switch networks can enlarge the network throughput. For irregular networks, a multi-path routing method called $k$-shortest path routing is conventionally utilized. However, it cannot efficiently exploit longer-than-shortest paths that would be detour paths to avoid bottlenecks. In this work, a novel routing method called $k$-optimized path routing to achieve high throughput is proposed for irregular networks. We introduce a heuristic to select detour paths that can avoid bottlenecks in the network to improve the average-case network throughput. Experimental results by network simulation show that the proposed $k$-optimized path routing can improve the saturation throughput by up to 18.2% compared to the conventional $k$-shortest path routing. Moreover, it can reduce the computation time required for optimization to $1/2760$ at a minimum compared to our previously proposed method.

***key words:*** *interconnection networks, warehouse-scale computing, data centers*

## 1. Introduction

Recently, the size of data centers for big-data processing has grown rapidly. They form a new computer class called *warehouse-scale* computers which will provide more than hundreds of thousands of nodes. Conventional interconnection networks for such a huge scale system is difficult to be formed with economic commodity switches. Also, large throughput to cope with a large request level parallelism is difficult to be satisfied.

Novel-class random networks have been proposed for such warehouse-scale computers. Since the throughput is more important than the latency for homogeneous warehouse-scale computers, interconnection networks using random regular graphs are efficient. Random topologies such as ring networks with random short-cut links [1] and Jellyfish topologies based on a random regular graph [2] can exploit many alternative paths for each source-destination pair, which leads to their larger throughput than that of Fat-Trees [3] commonly used in data centers.

These random graphs ensure their large throughput by multiple paths between nodes. However, the throughput is practically dependent on adopted packet routing algorithms. Conventional *equal cost multiple paths* routing (*ECMP*) [4] cannot make use of the variety of paths in the random graphs to improve the throughput. A routing algorithm called *k-shortest path routing* [5] has been proposed to utilize high throughput of random regular graphs. This method uses the shortest $k$ paths between two nodes to achieve comparative or better throughput than Fat-Trees using ECMP.

In this work, we consider the trade-offs between the network performance and the required routing table size. Although increasing the number of paths available for each source-destination pair would relax the limitation of our proposed algorithm to improve the network performance, the increased number of paths enlarges the routing table size, which leads to the difficulty of the implementation for large-scale networks.

For example, on Infiniband networks, a packet only can include the information of a destination node as a destination LID (Local IDentifier) in the header. One sub-set can use approximately 48,000 LIDs (i.e. destinations) for each intermediate switch at maximum. 3 paths available for each source-destination pair can support about $\frac{48{,}000}{3} = 16{,}000$ nodes, while 10 paths can support about only $\frac{48{,}000}{10} = 4{,}800$ nodes.

In this work, we explore methods for choosing multiple paths with their number fixed to maximize the throughput for arbitrary traffics. There is a linear programming model for obtaining the selection probability of paths which can maximize the worst-case throughput [6]. Our previous work [7] obtained the maximum worst-case throughput when $m$ paths are used between two nodes with the linear programming to select $k$ paths with the highest probability. We then used these $k$ paths to achieve *k-optimized path routing*, which maximizes the worst-case throughput. The problem of our previous method is the huge computational cost to obtain $k$ paths, which makes it difficult to apply to large-scale networks.

The difference of this paper from our previous work [7] is that we introduce a new linear program to reduce the computational complexity and increase the throughput. In this program, disjoint perfect matchings are defined for the complete bipartite graph of all source-destination pairs. The av-

Manuscript received January 7, 2020.
Manuscript revised May 22, 2020.
Manuscript publicized August 6, 2020.

[†]The author is with Japan Advanced Institute of Science and Technology, Nomi-shi, 923–1292 Japan.
[††]The author is with Hiroshima University, Higashihiroshima-shi, 739–8511 Japan.
[†††]The authors are with Dept. of ICS, Keio University, Yokohama-shi, 223–8522 Japan.
[††††]The author is with National Institute of Informatics, Tokyo, 101–8430 Japan.
a) E-mail: kawano@am.ics.keio.ac.jp

DOI: 10.1587/transinf.2020PAP0005

**Table 1** Qualitative comparison of alternative solutions ('+': good, 'o': fair, '-': poor).

| Routing | *k*-shortest [5] | *k*-opt.-WC [7] | *k*-opt.-PM (proposed) |
|---|---|---|---|
| Latency | + | o | o |
| Throughput | - | + | o |
| Computational cost | + | - | o |

erage throughput of traffics corresponding to the matchings is then optimized to obtain the $k$ paths. The number of the traffics to be optimized can be significantly reduced while considering all the source-destination pairs.

A qualitative comparison among conventional and our proposed algorithms is summarized in Table 1. In this table, our previous method to optimize Worst-Case throughput and our proposed method in this work to optimize with Perfect Matching are represented by "$k$-opt.-WC" and "$k$-opt.-PM", respectively.

The rest of the paper is organized as follows. Section 2 overviews conventional data center networks and high-throughput routing methods. In Sect. 3, several network measures are defined. Section 4 provides our two linear program models to obtain the optimum routing paths. Section 5 describes our routing framework called $k$-optimized path routing for high-throughput irregular networks. In Sect. 6, our proposed methods are evaluated for network throughput and computational complexity. Finally, we conclude this paper in Sect. 7.

## 2. Related Work

In this section, we review existing methods for measuring the throughput of the networks and routing methods for improving the throughput.

### 2.1 High-Throughput Large-Scale Networks

Although large-scale data center networks generally use Fat-Tree topologies [3], they are disadvantageous because there exist many communication hotspots that decrease network utilization [8]. One possible solution for eliminating such bottlenecks is to design reconfigurable networks such as optical wireless communications. However, they require high costs, and thus they are not currently reasonable.

DLN [1] and Jellyfish [2] apply random graphs [9] to the topology of interconnection networks. Slim Fly [10] and Xpander [11] are semi-optimal network topologies for given the number of switches and the number of ports per switch. These four networks above achieve balanced throughput for arbitrary traffics and communication loads. In particular, the network topologies adopting random graphs are promising because they make network sizes and the number of ports flexible. By using a fluid-flow model, Jyothi et al. show that such *random topologies* improve the throughput compared with Fat-Tree topologies if we use the optimal routing between end nodes [12].

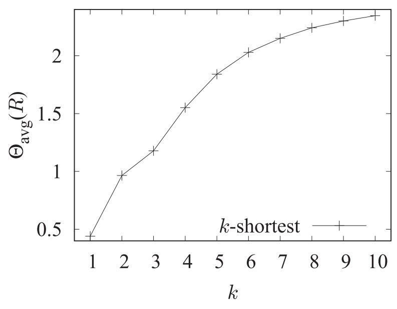


**Fig. 1** Average throughput $\Theta_{avg}(R)$ achieved by $k$-shortest path routing $R$ (64 nodes, degree of 4).

### 2.2 Maximizing Throughput by Routing

Wang et al. propose the multi-path routing algorithm called *SCRAT* [13], which increases the bandwidth compared to the conventional multi-path routing. However, SCRAT considers the bandwidth of an independent flow for each source-destination pair and does not consider the multiple concurrent flows among multiple source-destination pairs.

For Jellyfish, the *k-shortest path routing* is proposed to improve the network throughput [2]. Yuan et al. propose the *limited length spread k-shortest path routing* (*LLSKR*), which improves the $k$-shortest path routing by more effectively exploiting path diversity [14].

Figure 1 shows the relationship between the throughput and the number $k$ of paths used in the $k$-shortest path routing with 64 4-degree nodes. We show here the average throughput, $\Theta_{avg}(R)$, by $k$-shortest path routing algorithm $R$ for randomly generated 1,000 traffic patterns. Note that the strict definition of the throughput is shown in Sect. 3. The throughput increases with the value of $k$.

The $k$-shortest path routing and LLSKR improve communication throughput by reducing the number of occupied channels. On the one hand, longer-than-shortest paths increase the number of occupied channels, and consequently, they decrease the throughput, as shown in [14]. On the other hand, however, using longer paths instead of short paths may avoid the bottleneck of specific channels along the short paths to improve the network throughput.

This paper explores the use of longer-than-shortest paths for improving network throughput. More specifically, we measure the path utilization that maximizes network throughput. Based on the utilization, we can generate routing paths that may include longer-than-shortest paths to improve network performance.

## 3. Metrics for Measuring Network Throughput

Static routing algorithm design can be modeled as a multi-commodity flow problem (MCF). In this model, communication flows on the network so that the load on each channel does not exceed a certain capacity.

In this work, we explore a method for generating multiple paths to improve network throughput. We use the framework of the existing method [6] to measure the throughput under arbitrary traffics. In the same way as this framework, we provide some definitions of a routing method and the network throughput.

**Definition 1.** *A link is a set of two uni-directional channels.*

**Definition 2.** *A path $p$ is a set of channels $\{c_1, \ldots, c_{|p|}\}$ that a packet uses for a source-destination pair.*

First, a static routing algorithm defines a set of paths available in the network for each source-destination pair. To describe a routing algorithm $R$, we let $R(p)$ be the probability that a packet uses the path $p$ with. Then, $R$ can represent a valid static routing algorithm as long as the following conditions are satisfied.

$$\begin{aligned} &\textstyle\sum_{p\in P_{s,d}} R(p) = 1 && \forall s, d \in N, \\ &R(p) \geq 0 && \forall p \in P, \end{aligned}$$

$P$ and $P_{s,d}$ represent the set of all paths and the set of paths between source $s$ and destination $d$, respectively. Each path is a *simple path* that eliminates any loop and any revisit channel. This formulation creates a commodity flow for each of the $|N|^2$ source-destination pairs in the network.

Using this multi-commodity flow formulation, several network metrics can be defined. By our definition, the maximum throughput in a network can sustain under a given traffic pattern. The throughput is determined by the channel loads. That is, once the average load on a channel reaches the channel's capacity, that channel is saturated. The first channel to saturate becomes a bottleneck and thus determines network throughput.

The expected number of packets that cross a particular channel $c$ for each cycle, referred to as the load $\gamma_c$, is the sum of the loads contributed by each source-destination pair. In terms of the traffic matrix $\pi$ and the routing algorithm $R$,

$$\gamma_c(R, \pi) = \textstyle\sum_{s,d\in N} \pi_{s,d} \sum_{\substack{p:c\in p,\\ p\in P_{s,d}}} R(p),$$

where $\pi_{s,d}$ is a binary value that represents whether communication exists for the $(s, d)$-pair.

In the traffic matrix $\pi$, each source node sends packets to exactly one destination node, while each destination node receives packets from exactly one source node. That is,

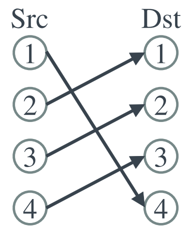
$$\begin{aligned} &\pi_{s,d} \in \{0, 1\} && \forall s, d \in N, \\ &\textstyle\sum_{d\in N} \pi_{s,d} = 1 && \forall s \in N, \\ &\textstyle\sum_{s\in N} \pi_{s,d} = 1 && \forall d \in N. \end{aligned}$$

Figure 2 (a) shows an example of a traffic matrix for four nodes and Fig. 2 (b) shows the corresponding source-destination pairs.

On typical networks, each channel's capacity (i.e. the maximum channel throughput rate) may be different from each other. In this work, we set the value of each channel's capacity $b_c = 1$, which is the same across all channels in the network. The normalized channel load $\frac{\gamma_c(R,\pi)}{b_c}$ is also the

| S\D | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | \ | 0 | 0 | 1 |
| 2 | 1 | \ | 0 | 0 |
| 3 | 0 | 1 | \ | 0 |
| 4 | 0 | 0 | 1 | \ |

(a) Traffic matrix.



(b) Source-destination pairs.

**Fig. 2** Traffic matrix $\pi$ for 4 nodes.

same across all channels, i.e. $\forall c \in C$ $\frac{\gamma_c(R,\pi)}{1} = \gamma_c(R, \pi)$. By applying the above formulations to each channel load, we define the maximum channel load across the whole network as

$$\gamma_{\max}(R, \pi) = \max_{c\in C} [\gamma_c(R, \pi)] .$$

This maximum channel load defines the maximum throughput $\Theta(R, \pi)$ of the network under the traffic matrix $\pi$ as

$$\Theta(R, \pi) = \gamma_{\max}(R, \pi)^{-1}.$$

Each source-destination pair can send and receive packets at up this ratio of each channel's capacity under the traffic matrix $\pi$ without saturating all channels in the network.

In the above formulations, the channel load $\gamma_c$ is linear in the routing algorithm $R$. Moreover, since it is the maximum value in a set of $\gamma_c$ for all channels, the maximum channel load $\gamma_{\max}$ is convex in $R$. The existing method [6] uses these properties to establish a linear program that can obtain the optimal channel load for a routing algorithm $R$ and an arbitrary traffic matrix $\pi$.

## 4. Optimization of Throughput with Linear Program

In this section, we propose two linear program models that minimize the channel load to extract the utilization ratio of each path. Considering the trade-off of the achieved network performance and the algorithmic computational complexity, we propose two different definitions of throughput to optimize.

### 4.1 Worst-Case Throughput

Firstly, we consider the problem of designing a routing algorithm $R$ with the optimal worst-case throughput under adversarial traffic. The maximum channel load for $R$ under any traffic is defined as $\gamma_{\text{wc}}(R)$. For some channel $c$ and traffic matrix $\pi$, $\gamma_c(R, \pi) = \gamma_{\text{wc}}(R)$ is satisfied. This equality induces a new variable $w$ that satisfies $w \geq \gamma_{\text{wc}}(R)$. This variable is used as an objective function in the following linear program:

$$\begin{aligned} \text{minimize} \quad & w \\ \text{subject to} \quad & \textstyle\sum_{p\in P_{s,d}} R(p) = 1 && \forall s, d \in N, \\ & R(p) \geq 0 && \forall p \in P, \\ & \gamma_c(R, \pi) \leq w && \forall c \in C, \pi \in \Pi, \end{aligned}$$

where a set of traffic matrices $\Pi$ is $|N|$-permutations for a set of nodes $N$. The size of $\Pi$ becomes $|N|!$, which makes the linear program difficult to solve in a practical time.

In order to reduce the number of constraints to a polynomial number, the Lagrange dual function and the Birkhoff–von Neumann theorem can be utilized [6]. We introduce new variables $v$ and $u$ to reformulate the problem. The resulting linear program is:

$$\begin{array}{lll} \text{minimize} & w & \\ \text{subject to} & \sum_{p \in P_{s,d}} R(p) = 1 & \forall s,d \in N, \\ & R(p) \geq 0 & \forall p \in P, \\ & \sum_{\substack{p: c \in p, \\ p \in P_{s,d}}} R(p) \leq v_{d,c} - u_{s,c} & \forall s,d \in N, \\ & & c \in C, \\ & \sum_{d \in N} v_{d,c} - \sum_{s \in N} u_{s,c} = w & \forall c \in C. \end{array}$$

### 4.2 Average-Case Throughput under Traffics of Disjoint Perfect Matchings

Although the linear program shown in Sect. 4.1 can be executed with the polynomial time and memory, the computational complexity is still large. In this section, we will consider not only all possible source-destination pairs but also the reduced number of traffics to be optimized.

We start from generating a vector of nodes $(n_1, n_2, \dots, n_{|N|})$ and randomly shuffle the elements to get a new vector $L$. From this vector $L$, we can obtain the following set of traffic matrices $X$,

$$X := \{\pi^i \mid i \in \{1, 2, \cdots, |N| - 1\}\}.$$

Each traffic matrix is defined as

$$\pi^i_{s,d} = \begin{cases} 1 & (\text{if } L^{-1}(d) = L^{-1}(s) + i \bmod |N|), \\ 0 & (\text{otherwise}), \end{cases}$$

where $L^{-1}(s)$ denotes the index of the source node $s$ in the vector $L$.

Figure 3 shows an example of traffic generation. We start from the complete bipartite graph with a set of source nodes and another set of destination nodes[†]. The alignment for each source-destination pair is changed along with the traffic index $i$ to generate disjoint perfect matchings $\{\pi^i \mid i \in \{1, 2, \cdots, |N| - 1\}\}$.

For this set of traffic matrices $X$, the average-case throughput is optimized by the following linear program model.

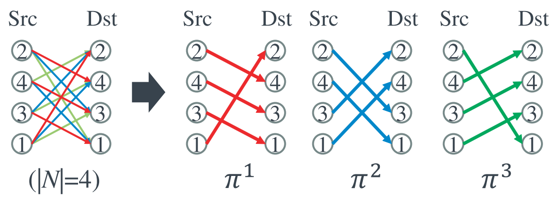


**Fig. 3** Example of traffic generation for 4 nodes.

[†]We eliminate edges between the nodes of the same index.

$$\begin{array}{lll} \text{minimize} & \frac{1}{|X|} \cdot \sum_{\pi \in X} w_\pi & \\ \text{subject to} & \sum_{p \in P_{s,d}} R(p) = 1 & \forall s,d \in N, \\ & R(p) \geq 0 & \forall p \in P, \\ & \gamma_c(R, \pi) \leq w_\pi & \forall c \in C, \pi \in X. \end{array}$$

Note that we approximately maximize the average-case throughput by minimizing the average channel load under the traffics. Compared with the optimization of the worst-case throughput in Sect. 4.1, we can reduce the number of traffics to be optimized from $|N|!$ to $|N| - 1$.

## 5. Generating $k$-Optimized Paths

In this section, we introduce $k$-optimized path routing that can replace the conventional $k$-shortest path routing. The proposed method uses the linear program shown in the previous section. In a similar way to the $k$-shortest path routing, the resulted routing uses $k$ paths for each source-destination pair. To determine the $k$ paths, firstly we generate $m$-shortest paths, where the number $m$ is enough larger than the number $k$. Then, we obtain $k$ paths that can maximize the throughput from these $m$ paths.

**Definition 3.** *The utilization ratio of each path $p \in P$ is equal to the resulted variable $R(p)$ (i.e. the probability that a packet uses the path $p$ with) that maximizes the network throughput in the linear program shown in Sect. 4.*

By introducing the definition above, a path search with linear programming is performed by the following procedures.

(i) For a set of available paths $P$, the maximum channel load is calculated using the linear program shown in Sect. 4.

(ii) The utilization ratio of each path is extracted in the optimization result. The top $k$ paths for each source-destination pair are then selected from the $m$ paths.

The top $k$ paths may include longer-than-shortest paths that are not in $k$-shortest paths. These paths can improve the network throughput by using them as substitutes for the shortest paths with low utilization.

Figure 4 shows an example of the path selection. The step (i) obtains the channel loads when using $m$-shortest paths for each source-destination pair. At the same time, the utilization ratio of each path is calculated in the optimization. The step (ii) extracts detour paths that can avoid bottlenecks. These paths are included in $k$ available paths to generate routing optimized for the network throughput. In the example of Fig. 4, the $k = 3$ paths that have the highest utilization ratio are selected from the $m = 8$ paths.

In the optimization step (ii), we adopt one of the two linear program models shown in Sect. 4 to generate $k$-optimized paths. Hereinafter, the resulted routing by optimizing the worst-case throughput (Sect. 4.1) is

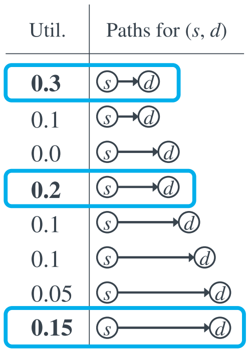
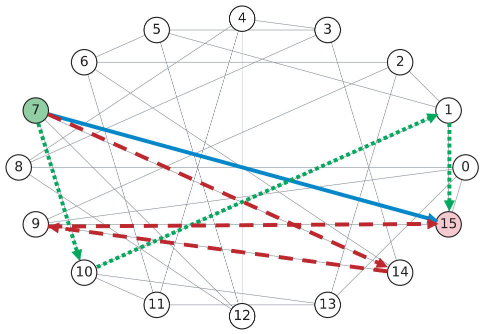
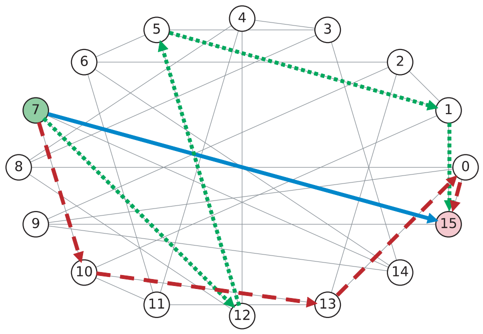


**Fig. 4** Example of path selection ($m$ = 8, $k$ = 3).



(a) $k$-shortest paths between nodes #7 and #15.



(b) $k$-optimized paths between nodes #7 and #15.

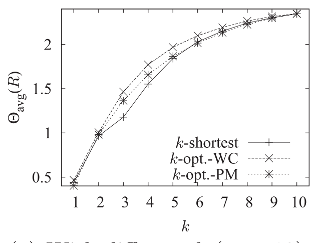
**Fig. 5** Example of paths ($k$ = 3).

called *k-optimized-WC*. Similarly, the routing by optimizing the average-case throughput of disjoint perfect matchings (Sect. 4.2) is called *k-optimized-PM*.

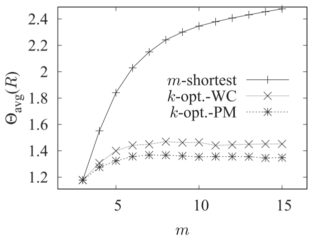
Figure 5 shows an example of paths generated by $k$-shortest path routing and $k$-optimized path routing. When $k$ = 3, the maximum path length between nodes #7 and #15 used in the $k$-shortest path routing is 3 (7-10-1-15, 7-14-9-15). On the other hand, the maximum path length between nodes #7 and #15 used in the $k$-optimized path routing is 4 (7-10-13-0-15, 7-12-5-1-15). By using these non-shortest paths, it is possible to prevent load congestion on specific channels to improve the overall throughput.

## 6. Evaluation

The proposed $k$-optimized path routing introduced in Sect. 5



(a) With different $k$ ($m$ = 10).



(b) With different $m$ ($k$ = 3).

**Fig. 6** Average throughput for degree-8 64 nodes.

is compared with the conventional $k$-shortest path routing. Gurobi [15] is used as a linear program solver. A barrier method is used for the optimization method in the linear program.

Our two routing methods $k$-optimized-WC and $k$-optimized-PM are compared with the conventional $k$-shortest path routing. We set the default number of paths for each source-destination pair to be searched to $m$ = 10, while we set the default number of paths used in the resulted routing to $k$ = 3.

### 6.1 Average Throughput on Random Regular Graphs

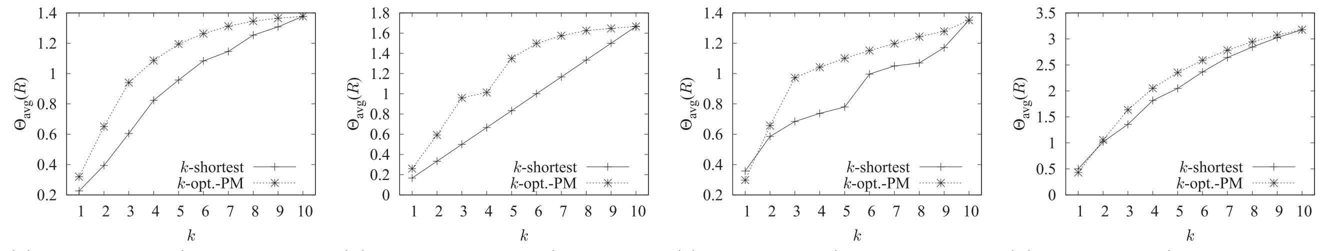
In this section, we apply the conventional $k$-shortest path routing and the proposed routing methods $k$-optimize-WC ($k$-opt.-WC) and $k$-optimize-PM ($k$-opt.-PM) to random regular graphs that have the same degree of each node. We then evaluate the average throughput for 1,000 randomly generated traffic matrices. We evaluate routing algorithm on 64-node degree-8 and 256-node degree-16 random regular graphs.

Note that as mentioned in Sect. 1, we assume our proposed algorithm can be applied to warehouse-scale computing systems that have hundreds of thousands of nodes. Although we set the evaluated network size to 256 at most, we can fill the gap by utilizing commodity high-radix switches to aggregate a large number of computing hosts. For example, 256 48-port switches can configure degree-16 random regular graphs where each switch has 32 computing hosts. In this case, the resulted system size can be 32×256 = 8,192.

Figure 6 (a) shows the evaluation results of the average throughput when the parameter $k$ is varied for 64 nodes. As shown in this figure, $k$-opt.-WC, which optimizes the worst-case throughput, achieves the best average throughput. When $k$ = 3, it improves the throughput by up to 24.3% compared to the conventional $k$-shortest path routing.

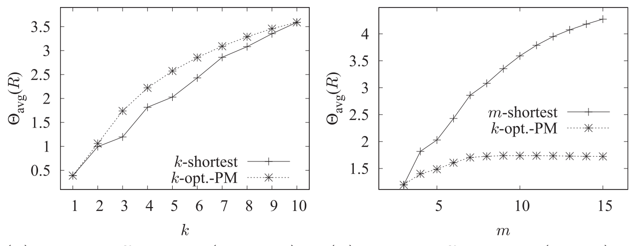
On the other hand, $k$-opt.-PM, which optimizes the average-case throughput by perfect matching, slightly degrades the average throughput compared to $k$-opt.-WC. This is because the number of traffics to be optimized is reduced. Nevertheless, by considering all source-destination pairs by disjoint matchings, it improves the average throughput by up to 15.8% compared to the $k$-shortest path routing when $k$ = 3.

Figure 6 (b) shows the evaluation results of the average throughput when the parameter $m$ is changed for 64



(a) 3-D Torus (degree 6, 64 switches). (b) 12-ary Fat-Tree (degree 12, 72 edge switches). (c) Dragonfly (degree 8, 114 switches). (d) Slim Fly (degree 11, 98 switches).

**Fig. 8** Average throughput on conventional non-random topologies with different $k$ ($m$ = 10).



(a) With different $k$ ($m = 10$). (b) With different $m$ ($k = 3$).

**Fig. 7** Average throughput for degree-16 256 nodes.

nodes. Ideally, as the number $m$ of paths to be searched increases, the average throughput also increases. However, as shown in this figure, the average throughputs do not increase monotonically in both the $k$-opt.-WC and $k$-opt.-PM methods. This is because both methods heuristically select $k$ paths that have the highest utilization ratio among $m$ paths in the optimization phase.

For $m = 15$, $k$-opt.-WC and $k$-opt.-PM degrade the average throughput by 41.4% and 45.6% compared to $m$-shortest path routing, respectively. These results show that increasing the number of available paths is significantly more effective for improving the throughput than optimizing the available paths with the number of paths fixed. Compared to the conventional $k$-shortest path routing, the maximum improvement rates of the average throughputs of $k$-opt.-WC and $k$-opt.-PM are 24.8% and 16.6%, respectively.

As shown in Fig. 7 (a) and 7 (b), the same tendency can be seen for degree-16 256-node networks. Note that $k$-opt.-WC is not evaluated in this section because as shown in Sect. 6.3, the calculation cost for optimizing $k$-opt.-WC is impractically large. When $k = 3$ and $m = 10$, the average throughput is improved by 45.5% compared to the conventional $k$-shortest path routing. When $k = 3$ and $m = 15$, the average throughput degrades 59.7% compared to $m$-shortest path routing.

## 6.2 Average Throughput on Conventional Non-Random Topologies

In this section, we apply the conventional $k$-shortest path routing and the proposed routing method $k$-optimize-PM ($k$-opt.-PM) to conventional non-random topologies as follows:

- 3-D Torus (degree 6, 64 switches) [16]
- 12-ary Fat-Tree (degree 12, 72 edge switches) [17]
- Dragonfly (degree 8, 114 switches) [18]
- Slim Fly (degree 11, 98 switches) [10]

It is notable that 12-ary Fat-Tree is an indirect topology, which contains 72 edge switches that hosts are directly connected to and 108 intermediate switches that hosts are not connected to (180 switches in total).

Figure 8 shows the evaluation results. In a quasi-optimal diameter-2 Slim Fly topology, using the shortest paths directly improves the throughput. The performance improvement rate of the $k$-opt.-PM compared to the $k$-shortest is suppressed to 20.6%. On the other hand, in the other topologies Torus, Fat-Tree, and Dragonfly, there is comparatively large room for routing optimization. Especially in an indirect Fat-Tree topology, there are many alternative paths via intermediate switches, The $k$-optimized path routing can improve the network throughput by 91.8% compared to the $k$-shortest path routing.

## 6.3 Computational Costs

In this section, we evaluate the memory size and the execution time required for the two linear program models introduced in Sects. 4.1 and 4.2. The Barrier method is used in a linear programming solver Gurobi. The specification of the machine for the evaluation is Intel® Xeon® CPU E5-2470 @ 2.30 GHz (2 Sockets × 8 Cores, 32 Threads) with 128 GB Memory.

Note that in this evaluation we assume that the $m$-shortest paths are given before the optimization phase. The calculation time to generate $m$-shortest paths is not included in this evaluation. This is why $k$-shortest path routing and $m$-shortest path routing are not evaluated in this section.

Figure 9 shows the evaluation results of the relationship between the network size (i.e. the number of nodes) and the required computation cost for the degree-8 random regular graphs with $k = 3$ and $m = 10$. As shown in Fig. 9 (a), $k$-opt.-WC that optimizes the worst-case throughput exceeds 16 GB of the required memory size for preserving variables and constraints on a linear program for the network size of 128 nodes. Besides, as shown in Fig. 9 (b), it takes more than 3 hours of the calculation time for 96 nodes. These high computational costs are mainly due to a large number

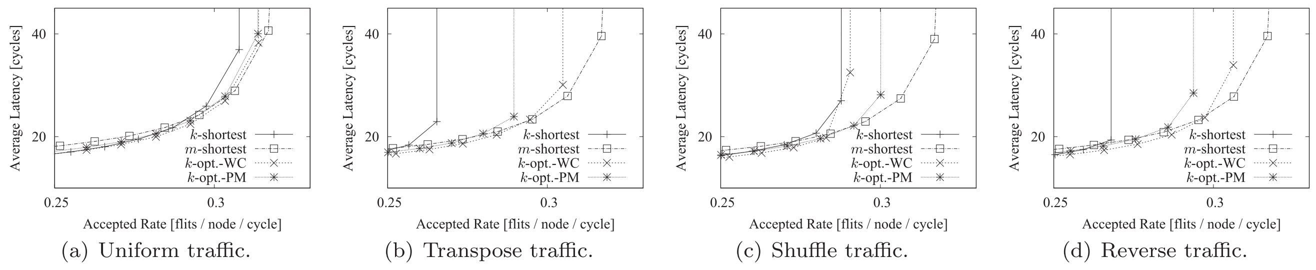


**Fig. 10** Network performance of $k$-optimized path routing for 64 nodes.

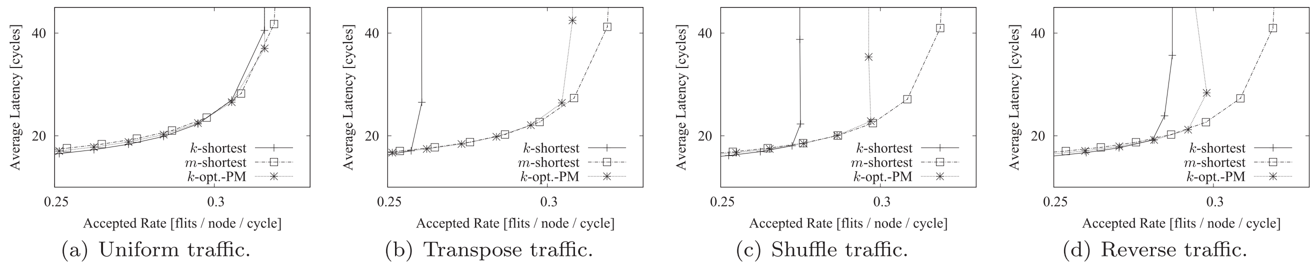


**Fig. 11** Network performance of $k$-optimized path routing for 256 nodes.

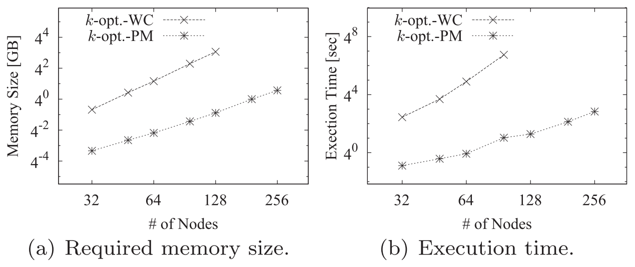


**Fig. 9** Calculation costs for degree-8 random regular graphs ($k = 3$, $m = 10$).

**Table 2** Network parameters.

| | |
|---|---|
| Simulation period | 10,000 cycles |
| Packet size | 1 flit |
| Number of VCs | 2 |
| Buffer size per VC | 8 flits |
| Router latency | 4 cycles/hop |
| Escape path [19] | up*/down* routing [20] |

of inequalities, $O(|N|^3)$, under the constraints of the linear program introduced in Sect. 4.1.

On the other hand, as shown in Fig. 9 (a), $k$-opt.-PM, which optimizes the average throughput by perfect matchings, achieves a significant reduction in the required memory size by reducing the constraints in the linear program model in Sect. 4.2. A similar tendency is observed in Fig. 9 (b) for the execution time. $k$-opt.-PM reduces the memory size to 1/240 and the computation time to 1/2760 compared to $k$-opt.-WC at a minimum. We can conclude that $k$-opt.-PM that is newly proposed in this paper can achieve both high throughput and reduced computation cost.

## 6.4 Network Performance

A cycle-accurate network simulator Booksim [21] is used for evaluation. Network parameters for the simulation are shown in Table 2. In order to avoid deadlocks, Duato's protocol [19] is used. In this protocol, we adopt the up*/down* routing [20] with the spanning trees optimization method [22] as the escape paths. In the simulation, we adopt uniform random traffic and three kinds of synthetic traffic of transpose, shuffle, and reverse traffic. For comparison with the conventional $k$-shortest path routing, we evaluate the proposed $k$-opt.-WC and $k$-opt.-PM for 64 nodes, while we only evaluate $k$-opt.-PM for 256 nodes because of the memory size limitation of the machine for the optimization.

Figure 10 shows the network performance of the $k$-shortest path routing and the proposed $k$-opt.-WC and $k$-opt.PM routing methods. As seen in this figure, both of the proposed routing methods achieve higher saturation throughput than the $k$-shortest path routing regardless of adopted traffic. Especially in reverse traffic, the throughput is improved by 14.3% and 9.6%, respectively.

These improvement rates are smaller than the rates of the average throughput shown in Fig. 6. This is because Duato's protocol induces another deadlock-free network layer for up*/down* routing. That is, the result values of the bandwidth are raised by that of the deadlock-free network.

It is also notable that both of the proposed $k$-optimized path routing methods achieve almost the same network latency at the low accepted flit rate as the conventional $k$-shortest path routing. This is due to Duato's protocol applied in this simulation. The proposed $k$-optimized path routing degrades the network latency because of longer-than-shortest paths as default, while the conventional $k$-shortest path routing also degrades the network latency when the accepted rate is comparatively high because of longer-than-shortest deadlock-free paths in the up*/down* routing.

In the three kinds of synthetic traffic, the saturation throughput by $k$-optimized path routing is deteriorated by 4.8% compared with $m$-shortest path routing. On the other hand, in uniform traffic, the deterioration rate of the saturation throughput is suppressed to 1.0%.

Similarly, as shown in Fig. 11, especially in transpose traffic, $k$-opt.-PM improves the throughput by 18.2% compared to the conventional $k$-shortest path routing for 256 nodes. The $k$-opt.-PM with reduced number of traffic matrices to be optimized can significantly improve the saturation throughput compared with the conventional $k$-shortest path routing. Compared with the $m$-shortest path routing, the deterioration rate of the saturation throughput is suppressed to 0.9%.

## 7. Conclusion

In this study, we explored multi-path routing methods to maximize network throughput for high-performance random regular graphs. We introduced $k$-optimized path routing that uses $k$-multiple paths that may include longer-than-shortest paths for each source-destination pair.

The $k$-optimized path routing method that optimizes the worst-case throughput had a problem in its feasibility that the memory size and computation time required for the linear program became quite large. In this research, a new method called $k$-optimized-PM (Perfect Matching) was proposed. It achieves both the same throughput as the previous method and a drastic reduction in the required memory size. This can be realized by reducing the number of traffics to be optimized while considering all source-destination pairs.

The $k$-optimized-PM proposed in this paper has reduced the computation time required for optimization to 1/2760 at a minimum compared to our previously proposed $k$-optimized-WC. Furthermore, evaluation results by network simulation show that the saturation throughput has been improved by up to 18.2% compared to the conventional $k$-shortest path routing.

## Acknowledgments

A part of this work was supported by JSPS KAKENHI Grant Number JP 20K19788.

### References

[1] M. Koibuchi, H. Matsutani, H. Amano, D.F. Hsu, and H. Casanova, “A case for random shortcut topologies for HPC interconnects,” Proc. International Symposium on Computer Architecture (ISCA), pp.177–188, June 2012.

[2] A. Singla, C.Y. Hong, L. Popa, and P.B. Godfrey, “Jellyfish: Networking data centers randomly,” Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI), pp.225–238, April 2012.

[3] C.E. Leiserson, “Fat-trees: Universal networks for hardware-efficient supercomputing,” IEEE Trans. Comput. (TC), vol.34, no.10, pp.892–901, Oct. 1985.

[4] C. Hopps, “RFC 2992 – Analysis of an equal-cost multi-path algorithm,” Tech. Rep., Internet Engineering Task Force, 2000.

[5] J.Y. Yen, “Finding the *K* shortest loopless paths in a network,” Management Science, Theory Series, vol.17, no.11, pp.712–716, July 1971.

[6] B. Towles, W.J. Dally, and S. Boyd, “Throughput-centric routing algorithm design,” Proc. 15th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA), pp.200–209, June 2003.

[7] R. Kawano, R. Yasudo, H. Matsutani, and H. Amano, “$k$-optimized path routing for high-throughput data center networks,” Proc. 6th International Symposium on Computing and Networking (CANDAR), pp.99–105, Nov. 2018.

[8] S. Kassing, A. Valadarsky, G. Shahaf, M. Schapira, and A. Singla, “Beyond fat-trees without antennae, mirrors, and disco-balls,” Proc. Conference of the ACM Special Interest Group on Data Communication (SIGCOMM), pp.281–294, Aug. 2017.

[9] D.J. Watts and S.H. Strogatz, “Collective dynamics of ‘small-world’ networks,” Nature, vol.393, no.6684, pp.440–442, June 1998.

[10] M. Besta and T. Hoefler, “Slim fly: A cost effective low-diameter network topology,” Proc. International Conference for High Performance Computing, Networking, Storage and Analysis (SC), pp.348–359, Nov. 2014.

[11] A. Valadarsky, G. Shahaf, M. Dinitz, and M. Schapira, “Xpander: Towards optimal-performance datacenters,” Proc. 12th International on Conference on emerging Networking EXperiments and Technologies (CoNEXT), pp.205–219, Dec. 2016.

[12] S.A. Jyothi, A. Singla, P.B. Godfrey, and A. Kolla, “Measuring and understanding throughput of network topologies,” Proc. International Conference for High Performance Computing, Networking, Storage and Analysis (SC), pp.761–772, Nov. 2016.

[13] H. Wang, K. Qian, C. Hu, C. Zhang, and Y. Zhou, “Routing optimization for server-centric data center networks,” Computer Science and Information Systems, vol.13, no.2, pp.593–608, June 2016.

[14] X. Yuan, S. Mahapatra, W. Nienaber, S. Pakin, and M. Lang, “A new routing scheme for Jellyfish and its performance with HPC workloads,” Proc. International Conference on High Performance Computing, Networking, Storage and Analysis (SC), pp.1–11, Nov. 2013.

[15] Gurobi Optimization Inc., “Gurobi optimizer reference manual,” http://www.gurobi.com

[16] W.J. Dally, “Performance analysis of k-ary n-cube interconnection networks,” IEEE Trans. Comput., vol.39, no.6, pp.775–785, 1990.

[17] M. Al-Fares, A. Loukissas, and A. Vahdat, “A scalable, commodity data center network architecture,” Proc. ACM Conference on Data Communication (SIGCOMM), pp.63–74, Aug. 2008.

[18] J. Kim, W.J. Dally, S. Scott, and D. Abts, “Technology-driven, highly-scalable dragonfly topology,” Proc. International Symposium on Computer Architecture (ISCA), pp.77–88, June 2008.

[19] F. Silla and J. Duato, “High-performance routing in networks of workstations with irregular topology,” IEEE Trans. Parallel Distrib. Syst. (TPDS), vol.11, no.7, pp.699–719, July 2000.

[20] M.D. Schroeder, A.D. Birrell, M. Burrows, H. Murray, R.M. Needham, T.L. Rodeheffer, E.H. Satterthwaite, and C.P. Thacker, “Autonet: A high-speed, self-configuring local area network using point-to-point links,” IEEE J. Sel. Areas Commun., vol.9, no.8, pp.1318–1335, Oct. 1991.

[21] N. Jiang, D.U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, D.E. Shaw, J. Kim, and W.J. Dally, “A detailed and flexible cycle-accurate network-on-chip simulator,” Proc. IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pp.86–96, April 2013.

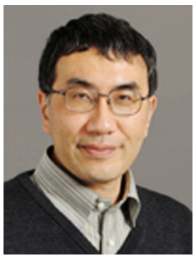
[22] H. Matsutani, P. Bogdan, R. Marculescu, Y. Take, D. Sasaki, H. Zhang, M. Koibuchi, T. Kuroda, and H. Amano, “A case for wireless 3D NoCs for CMPs,” Proc. Asia and South Pacific Design Automation Conference (ASP-DAC), pp.22–28, Jan. 2013.

**Ryuta Kawano** received the BE, ME and Ph.D. degrees from Keio University, Yokohama, Japan, in 2013, 2015 and 2018, respectively. He is currently an assistant professor in Japan Advanced Institute of Science and Technology. His research interests include the area of high-performance computing and interconnection networks. He is a member of the IEEE, IEICE, and IPSJ.

**Ryota Yasudo** received the BE, ME, and Ph.D. degrees from Keio University, Japan, in 2014, 2016, and 2019, respectively. He is currently an assistant professor at Hiroshima University. His current research interests include interconnection networks, parallel computing, and reconfigurable computing.

**Hiroki Matsutani** received the BA, ME, and PhD degrees from Keio University in 2004, 2006, and 2008, respectively. He is currently an associate professor in the Department of Information and Computer Science, Keio University. From 2009 to 2011, he was a research fellow in the Graduate School of Information Science and Technology, The University of Tokyo, and awarded a Research Fellowship of the Japan Society for the Promotion of Science.

**Michihiro Koibuchi** received the BE, ME, and PhD degrees from Keio University, Yokohama, Japan, in 2000, 2002 and 2003, respectively. Currently, he is an associate professor in the Information Systems Architecture Research Division, National Institute of Informatics and the Graduate University of Advanced Studies, Tokyo, Japan. His research interests include the area of high-performance computing and interconnection networks. He is a member of the IEEE and a senior member of IEICE and IPSJ.



**Hideharu Amano** received Ph.D degree from the Department of Electronic Engineering, Keio University, Japan in 1986. He is currently a professor in the Department of Information and Computer Science, Keio University. His research interests include the area of parallel architectures and reconfigurable systems.