Optimal Oblivious Routing for Structured Networks

Sucha Supittayapornpong[†], Pooria Namyar[‡], Mingyang Zhang[‡], Minlan Yu[§], Ramesh Govindan[‡]
[†]Vidyasirimedhi Institute of Science and Technology, Thailand
[‡]University of Southern California, United States
[§]Harvard University, United States

***Abstract*—Oblivious routing distributes traffic from sources to destinations following predefined routes with rules independent of traffic demands. While finding optimal oblivious routing is intractable for general topologies, we show that it is tractable for structured topologies often used in datacenter networks. To achieve this, we apply graph automorphism and prove the existence of the optimal automorphism-invariant solution. This result reduces the search space to targeting the optimal automorphism-invariant solution. We design an iterative algorithm to obtain such a solution by alternating between two linear programs. The first program finds an automorphism-invariant solution based on representative variables and constraints, making the problem tractable. The second program generates adversarial demands to ensure the final result satisfies all possible demands. Since, the construction of the representative variables and constraints are combinatorial problems, we design polynomial-time algorithms for the construction. We evaluate proposed iterative algorithm in terms of throughput performance, scalability, and generality over three potential applications. The algorithm i) improves the throughput up to 87.5% over a heuristic algorithm for partially deployed FatTree, ii) scales for FatClique with a thousand switches, iii) is applicable to a general structured topology with non-uniform link capacity and server distribution.**

I. Introduction

Topology design for datacenter networks has gained attention due to the need to construct high capacity datacenters at low cost and low management complexity [1]–[12]. Although several topologies have been proposed, only the folded-Clos family of topologies [1]–[4] achieves designed capacity with existing routing solutions including Equal-Cost Multi Path (ECMP) [13] and Valiant Load-Balancing (VLB) [14], [15]. For topology designs that deviate from folded-Clos [6]–[11] the design of scalable routing algorithms that can achieve their designed capacity is an open question.

Routing inside datacenter networks can be categorized into traffic-aware routing and oblivious routing. Traffic-aware routing reduces network congestion and improve overall throughput by regularly adjusting routes and fractions of traffic demands over the routes according to queue occupancy or traffic demands [6], [7], [16]. These advantages come at a cost of specialized hardware and routing complexity. Alternatively, oblivious routing is much simpler. Traffic demands are distributed according to predefined routes and shares of the demands over each route. In particular, the routing is oblivious to the current traffic demand, so regular configuration of routes is unnecessary [15]. Because of this simplicity, oblivious routing, including ECMP and VLB, is deployed in several large-scale datacenter networks [1]–[4]. A real-world measurement at Facebook also suggests oblivious techniques work well leaving little room for improvement using advanced traffic-aware routing [17].

This work is supported by VISTEC's startup grant and the U.S. National Science Foundation under grant No. CNS-1955422.

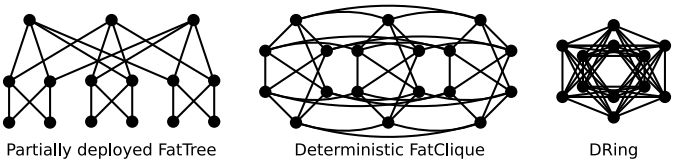

Fig. 1. Examples of highly structured topologies

Although existing oblivious routing is widely deployed, it achieves designed capacity only for the folded-Clos and clique topologies. Specifically, ECMP can achieve the designed capacity of folded-Clos topologies including FatTree [1], Google's Jupiter [3] and Facebook's Fabric [4]. VLB achieves designed capacity for a clique topology and works reasonably well in Microsoft's VL2 [2]. However, recent datacenter topology designs have moved away from the folded-Clos family [8]–[12] and, for these, existing oblivious routing approaches cannot be used.

Of these new designs, highly structured topologies, such as FatClique [11] and DRing [12] (Fig. 1), are more manageable (*i.e.*, they are easier to install and expand) than random topologies [8], [9]. For these structured topologies, this paper develops a general oblivious routing algorithm that relies on hardware support for multi-path routing (WCMP) already available in commodity switches.

For any network, oblivious routing can be formulated as a robust multi-commodity flow problem in which the number of constraints grows factorially with the number of switches in the network. Even for a small network, the problem size, in terms of numbers of variables and constraints, can easily overwhelm an optimization solver's memory and renders the problem intractable.

For structured topologies, however, we show that we can use the topological structure and graph automorphism to reduce the problem size to the point that is tractable for any off-the-shelf solver running on commodity hardware.

To do this, we first prove the existence of an optimal solution that is invariant to automorphism–the solution is a permutation of itself. Based on this result, we reduce the search space of an optimal solution to the solution that is automorphism-invariant. Using graph automorphism, we formulate a robust multi-commodity flow problem with significantly fewer variables and constraints to target this optimal solution. While the

formulation has robust constraints, we observe that the optimal solution is in a much lower dimensional space compared to the original robust formulation, due to the automorphism-invariant property of the optimal solution. Therefore, we only need to consider a smaller subset of traffic demands instead of considering all possible demands.

This leads to the design of an iterative algorithm that alternates between two linear programs. The first linear program finds an automorphism-invariant routing solution. The second linear program generates adversarial traffic demands to make sure the end result of the iterative algorithm is optimal and satisfies all possible traffic demands. The two linear programs are based on representative variables and constraints, which significantly reduce the problem sizes. However, the construction of representative variables and constraints is a combinatorial problem associated with the exponentially large number of automorphisms. We design polynomial-time algorithms to construct these representatives by utilizing the generators of the automorphisms.

We evaluate our iterative algorithm in terms of throughput, scalability, and generality over three potential applications. i) The algorithm improves the throughput by up to 87.5% over a heuristic algorithm for partially deployed FatTree topologies. ii) The algorithm is scalable and provides the optimal oblivious routing solution for a FatClique topology with a thousand switches. iii) We demonstrate the generality of the algorithm by considering a structured topology with non-uniform link capacity and server distribution.

The contributions of this work are threefold.

- We prove the existence of an automorphism-invariant optimal solution in every structured topology. This reduces the search space of optimal solutions.
- We design the iterative algorithm that targets the automorphism-invariant optimal solution using graph automorphism. The algorithm is tractable in comparison to solving the intractable oblivious routing formulation.
- We develop the polynomial-time construction of the algorithm and illustrate three applications of the algorithm.

The paper is organized as follows. Section II models a datacenter network and formulates the oblivious routing problem. Section III proves the existence of the optimal solution that is automorphism invariant. Section IV uses this insight to develop the iterative algorithm targeting the automorphism-invariant optimal solution. Section V provides an efficient construction of representative variables and constraints and Section VI describes our evaluation.

II. System Model

This section formally models a datacenter network, traffic demands, and the oblivious routing formulation. These models are general and should fit most practical datacenter networks.

A. Datacenter network model

A datacenter network is an interconnection of servers and ethernet switches. Each server has a single full-duplex port for bi-directional communication and is connected to a switch[1]. Each network switch has a finite number of full-duplex ports for interconnection with servers and other switches. For example, Broadcom's Tomahawk 4 Ethernet switch chip can be configured as 256 ports at 100Gbps [20]. Two devices are physically connected by connecting ports from both ends. In practice, there could be multiple physical links between two devices to increase communication capacity. Our model considers a logical link between any two devices, and the logical capacity equals the combined capacity of all physical links between the devices. Since the capacity of physical links are identical, we normalize the logical link capacity by the physical link capacity[2], and we only consider the number of physical links between devices. This model based on logical links is illustrated in Fig. 2.

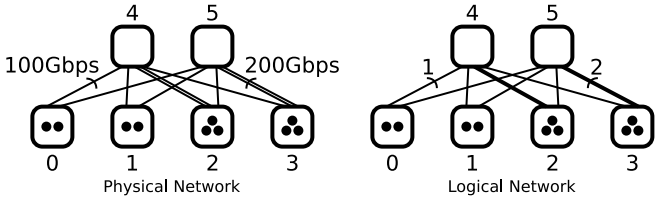

Fig. 2. Aggregated link capacity in the physical network is normalized to the number of physical links in the logical network. Switches are labeled by 0 to 5. Switches 0 to 3 are connected to 2, 2, 3, 3 servers respectively.

A datacenter network is a directed graph with the set of switches $\mathcal{S}$ and the set of logical links $\mathcal{L}$. Every logical link connects two switches. The graph is assumed to be connected. A directed link (i, j) connects switch i to switch j with capacity C_{ij} equaling the number of physical links between the switches. Because of the full-duplex ports, link $(i, j) \in \mathcal{L}$ if and only if $(j, i) \in \mathcal{L}$, and both links have identical capacity, i.e., $C_{ij} = C_{ji}$ for every $(i, j) \in \mathcal{L}$. To account for servers, let H_s be the number of servers connected to switch s for every $s \in \mathcal{S}$. Then, we define $\mathcal{S}_k$ as the set of switches with k servers attached, $\mathcal{S}_k = \{s \in \mathcal{S} : H_s = k\}$ for every non-negative integer k. The set $\mathcal{S}_0$ contains all switches with no servers attached. We further define a set of switches with server(s) attached as $\mathcal{H} = \mathcal{S} \backslash \mathcal{S}_0$.

We adopt the multi-commodity model to directly measure throughput between every pair of source and destination switches with server(s) attached. Denote the set of commodities by $\mathcal{C} = \big\{(u, v) \in \mathcal{H}^2 : u \neq v\big\}$. This set is also used for traffic demand modeling.

B. Traffic model

Traffic inside datacenter network is a combination of demands generated by servers attached to different switches. Since a commodity is defined at the switch level, the demand from a switch is the aggregate demand from its attached servers to servers on other switches. This demand is limited by

[1]When a server has multiple ports and routing capability, as in server-centric topologies [18], [19], such a server can be viewed as an ethernet switch attached with multiple servers.

[2]When the capacities of physical links are not identical, the normalization uses the greatest common divisor.

the capacity of the server-facing links. From the normalization, every switch can source and receive traffic demands at most the number of servers attached it. We denote the traffic demand of commodity (u, v) by t^{uv} for every $(u, v) \in \mathcal{C}$. A combination of traffic demands from every commodity forms a traffic matrix $[t^{uv}] \in \mathbb{R}_+^{|\mathcal{H}|^2}$ where $\mathbb{R}_+$ is a set of non-negative reals. The set of all possible traffic matrices is denoted by $\mathcal{T}$ where

$$\mathcal{T} = \left\{ [t^{uv}] \in \mathbb{R}_+^{|\mathcal{H}|^2} : \begin{array}{ll} \sum_{v \in \mathcal{H}} t^{uv} \leq H_u & \forall u \in \mathcal{H} \\ \sum_{u \in \mathcal{H}} t^{uv} \leq H_v & \forall v \in \mathcal{H} \\ t^{uu} = 0 & \forall u \in \mathcal{H} \end{array} \right\}.$$

The first and second constraints ensure every switch can source and receive traffic at most the total capacity of all server-facing links. The last constraint ensures that a switch internally forwards traffic between servers connected to it.

This traffic set is used to design optimal oblivious routes between every commodity with predictable throughput performance. For example, if all routes can deliver every traffic matrix in $\mathcal{T}$, no congestion from capacity violation will occur.

C. Oblivious routing formulation

Oblivious routing distributes traffic demand of every commodity over links in the network. We assume traffic can be split arbitrary at any intermediate switch similar to [14], [15]. Every splitting proportion is independent of the traffic demand. For example, with the $1:2$ split proportion, 1 unit of traffic demand is split to $1/3$ and $2/3$, and 2 units are split to $2/3$ and $4/3$. Hence, the routing is oblivious to traffic demands. Next, we formulate an oblivious routing optimization problem.

Recall that we adopt the multi-commodity model with the commodity set $\mathcal{C}$ where a commodity is a pair of source and destination switches. For each commodity (u, v), we decouple traffic demand t^{uv} from routing and splitting by defining f_{ij}^{uv} as a share of the demand over a link (i, j) such that the actual traffic over the link is $t^{uv} f_{ij}^{uv}$.

Since, in general, a datacenter's traffic demand can be any traffic matrix in $\mathcal{T}$, we compute the worst-case throughput as described in [10], [15], [21]. We denote the worst-case throughput of commodity (u, v) by a factor θ^{uv}, and the overall worst-case throughput is $\min_{(u,v)\in\mathcal{C}} \theta^{uv}$. For example, the overall worst-case throughput equals $1/2$ means any traffic matrix $T \in \mathcal{T}/2$ does not violate any link capacity. Our formulation maximizes these throughput hierarchically: i) maximize the overall worst-case throughput and then ii) maximize the marginal throughput of each commodity as follows:

$$\begin{aligned} \text{Maximize } & A \min_{(u,v)\in\mathcal{C}} \theta^{uv} + \sum_{(u,v)\in\mathcal{C}} \theta^{uv} \\ \text{Subject to } & \sum_{j\in\mathcal{O}(i)} f_{ij}^{uv} - \sum_{j\in\mathcal{I}(i)} f_{ji}^{uv} = \\ & \qquad \theta^{uv} \left\{ \mathbb{I}\left[i = u\right] - \mathbb{I}\left[i = v\right] \right\}, \quad \forall i \in \mathcal{S}, \forall (u, v) \in \mathcal{C} \\ & \sum_{(u,v)\in\mathcal{C}} t^{uv} f_{ij}^{uv} \leq C_{ij}, \qquad \forall (i, j) \in \mathcal{L}, \forall [t^{uv}] \in \mathcal{T} \\ & f_{ij}^{uv}, \theta^{uv} \in \mathbb{R}_+, \qquad \forall (u, v) \in \mathcal{C}, \forall (i, j) \in \mathcal{L}, \end{aligned} \tag{1}$$

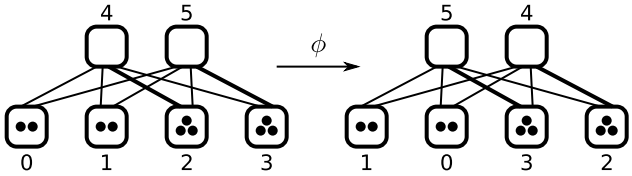

Fig. 3. The right network is an automorphism of the left network with $\phi(x) = x + 1$ when x is even and $\phi(x) = x - 1$ when x is odd. Both networks have the same adjacency and distributions of link capacities and servers.

where $\mathcal{O}(i)$ and $\mathcal{I}(i)$ are respectively the sets of switches that switch i has out-going links to and in-coming links from, and A is a sufficiently large constant. The first constraint is a conservation of share at every switch. The second constraint is a robust link capacity that considers all possible traffic matrices.

Although the formulation can be transformed to a linear program with the extreme points of the traffic set (similar to the technique used in [22]), the sheer numbers of commodities $O(|\mathcal{H}|^2)$, directed links $O(|\mathcal{L}|)$, and the extreme points $O(|\mathcal{H}|!)$ at the scale of datacenter networks can easily overwhelm the available memory of any off-the-shelf solver and its ability to obtain optimal solutions. In particular, the numbers of variables and constraints are respectively $O(|\mathcal{H}|^2 |\mathcal{L}|)$ and $O(|\mathcal{H}|^2 |\mathcal{S}| + |\mathcal{L}| |\mathcal{H}|!)$. Another formulation in [23] can reduce these numbers to polynomial; even so, the formulation can still overwhelm available memory easily. In the next section, we leverage topological structure in some datacenter networks to achieve tractability.

III. Characterization of Optimal Solutions

This section first introduces graph automorphisms that identify "similar" structure in datacenter networks. We then prove the existence of an optimal oblivious routing solution that also has a "similar" structure.

A. Graph automorphism

Graph automorphism is a permutation of nodes in the graph such that the adjacency between nodes before and after the permutation is the same. For a datacenter network, we extend the adjacency preservation of graph automorphism to include the preservation of servers at a switch and link capacity as illustrated in Fig. 3, and as defined formally below.

Definition 1 (Automorphism)**.** *Given a network topology with switch set $\mathcal{S}$, link set $\mathcal{L}$, link capacities $\{C_{ij}\}$, and the numbers of servers at every switch $\{H_u\}$, an automorphism $\phi : \mathcal{S} \to \mathcal{S}$ preserves:*

1) *Adjacency: $(\phi(i), \phi(j)) \in \mathcal{L}$ for every $(i, j) \in \mathcal{L}$.*
2) *Link capacity: $C_{ij} = C_{\phi(i)\phi(j)}$ for every $(i, j) \in \mathcal{L}$.*
3) *Number of Servers: $H_u = H_{\phi(u)}$ for every $u \in \mathcal{S}$.*

The set of all automorphisms is denoted by Φ. While the size of this set could grow exponentially large, it is finite, i.e., $|\Phi| < \infty$. Next, we use these automorphisms to reduce the search space for an optimal solution of the formulation in (1).

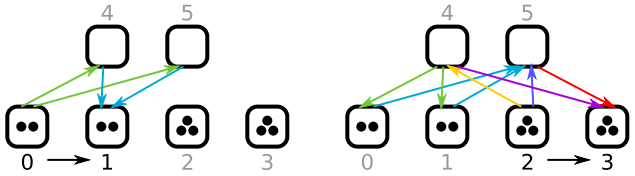

Fig. 4. The share variables of commodities $(0,1)$ and $(2,3)$ in an automorphism-invariant optimal solution are shown. Each arrow represents a share variable. For each commodity, the shares having the same value are assigned an identical color. They are automorphism invariant, e.g., $f_{04}^{01*} = f_{05}^{01*}$ under the automorphism that permutes switches 4 and 5. For commodity $(2,3)$, we have $f_{40}^{23*} = f_{41}^{23*}$ under the automorphism that permutes switches 0 and 1. For throughput variables, we have $\theta^{01*} = \theta^{10*}$ and $\theta^{23*} = \theta^{32*}$ under the automorphism that permutes switch pairs $(0,1)$ and $(2,3)$.

B. Existence of an automorphism-invariant optimal solution

We will show the existence of an automorphism-invariant optimal solution of the formulation in (1). In an automorphism-invariant optimal solution, every decision variables in a group under automorphisms in Φ takes the same value as shown in Fig. 4.

We first show that applying an automorphism to an optimal solution results in another optimal solution.

Lemma 1. *Suppose $\left\{f_{ij}^{uv*}, \theta^{uv*}\right\}$ is an optimal solution of the formulation in* (1)*. Given any automorphism $\phi \in \Phi$, the solution $\left\{f_{ij}^{uv}, \theta^{uv}\right\}$ is also an optimal solution such that $f_{ij}^{uv} = f_{\phi(i)\phi(j)}^{\phi(u)\phi(v)*}$ and $\theta^{uv} = \theta^{\phi(u)\phi(v)*}$.*

Proof. To show that the solution from an automorphism is also an optimal solution, we show that the solution leads to the optimal objective value and the feasibility of all constraints.

Proving the objective is optimal is straightforward as

$$A \min_{(u,v)\in\mathcal{C}} \theta^{\phi(u)\phi(v)*} + \sum_{(u,v)\in\mathcal{C}} \theta^{\phi(u)\phi(v)*} = A \min_{(u,v)\in\mathcal{C}} \theta^{uv*} + \sum_{(u,v)\in\mathcal{C}} \theta^{uv*}.$$

The equality follows the facts that the automorphism is a bijective mapping function and that the commodity set $\mathcal{C}$ contains every commodity. More precisely, every $(\phi(u), \phi(v)) \in \mathcal{C}$ is mapped to every $(u,v) \in \mathcal{C}$, so the minimum and the summation are calculated over the same set of variables. Therefore, the objective value under the solution equals to the optimal value, which is the objective value under the optimal solution. We then consider the feasibility of constraints.

From a share conservation constraint at switch i and commodity (u,v) in (1), the difference between the out-going shares and the in-coming shares is

$$\begin{aligned}\Delta(u,v,i) &= \sum_{j\in\mathcal{O}(i)} f_{\phi(i)\phi(j)}^{\phi(u)\phi(v)*} - \sum_{j\in\mathcal{I}(i)} f_{\phi(j)\phi(i)}^{\phi(u)\phi(v)*} \\ &= \sum_{j\in\mathcal{O}(\phi(i))} f_{\phi(i)j}^{\phi(u)\phi(v)*} - \sum_{j\in\mathcal{I}(\phi(i))} f_{j\phi(i)}^{\phi(u)\phi(v)*}. \quad (2)\end{aligned}$$

This is because the adjacency preservation of the automorphism in Definition 1. Since the share conservation holds true for the optimal solution, the difference in (2) becomes:

$$\begin{aligned}\Delta(u,v,i) &= \theta^{\phi(u)\phi(v)*}\left\{\mathbb{I}\left[\phi(i) = \phi(u)\right] - \mathbb{I}\left[\phi(i) = \phi(v)\right]\right\} \\ &= \theta^{uv}\left\{\mathbb{I}\left[i = u\right] - \mathbb{I}\left[i = v\right]\right\}.\end{aligned}$$

This shows that the share conservation constraint at switch i and commodity (u,v) under the solution from an automorphism is feasible. This holds true for every switch $i \in \mathcal{S}$ and every commodity $(u,v) \in \mathcal{C}$.

For the link capacity constraint at link (i,j) and traffic matrix $[t^{uv}]$ in (1), the left-hand side under the solution from an automorphism equals

$$\begin{aligned}\sum_{(u,v)\in\mathcal{C}} t^{uv} f_{\phi(i)\phi(j)}^{\phi(u)\phi(v)*} &= \sum_{(u,v)\in\mathcal{C}} t^{\phi^{-1}(u)\phi^{-1}(v)} f_{\phi(i)\phi(j)}^{uv*} \\ &\le C_{\phi(i)\phi(j)} = C_{ij}.\end{aligned}$$

The first equality follows from the re-indexing of terms in the summation over the commodity set. The inequality uses the fact that the capacity constraint of link $(\phi(i), \phi(j))$ under the optimal solution is feasible when the traffic matrix is $[t^{\phi^{-1}(u)\phi^{-1}(v)}]$, which is a member of the traffic set $\mathcal{T}$. The last equality follows the capacity preservation in Definition 1. Therefore, the link capacity constraint at link (i,j) and traffic matrix $T = [t^{uv}]$ under the solution from an automorphism is feasible. Again, this holds true for every link $(i,j) \in \mathcal{L}$ and every traffic matrix $[t^{uv}] \in \mathcal{T}$.

In short, we have shown that the solution from an automorphism yields the optimal objective value and leads to feasibility of all constraints. Thus, the solution is optimal. □

Lemma 1 implies that the automorphism of an optimal solution is another optimal solution. Next, we show that there exists an optimal solution that is invariant to automorphism.

Theorem 1. *There exists an automorphism-invariant optimal solution $\left\{\hat{f}_{ij}^{uv*}, \hat{\theta}^{uv*}\right\}$ to the formulation in* (1) *such that $\hat{f}_{ij}^{uv*} = \hat{f}_{\phi(i)\phi(j)}^{\phi(u)\phi(v)*}$ and $\hat{\theta}^{uv*} = \hat{\theta}^{\phi(u)\phi(v)*}$ for every $(u,v) \in \mathcal{C}$, every $(i,j) \in \mathcal{L}$, and every $\phi \in \Phi$.*

Proof. The formulation in (1) always has an optimal solution since the feasible set is compact and non empty. Let $\left\{f_{ij}^{uv*}, \theta^{uv*}\right\}$ be such an optimal solution. From Lemma 1, the solution $\left\{f_{\phi(i)\phi(j)}^{\phi(u)\phi(v)*}, \theta^{\phi(u)\phi(v)}\right\}$ is an optimal solution for every automorphism $\phi \in \Phi$. We construct the automorphism-invariant solution $\left\{\hat{f}_{ij}^{uv*}, \hat{\theta}^{uv*}\right\}$ as follows:

$$\hat{f}_{ij}^{uv*} = \frac{1}{|\Phi|}\sum_{\phi\in\Phi} f_{\phi(i)\phi(j)}^{\phi(u)\phi(v)*}, \qquad \forall (u,v)\in\mathcal{C}, \forall (i,j)\in\mathcal{L}$$

$$\hat{\theta}^{uv*} = \frac{1}{|\Phi|}\sum_{\phi\in\Phi} \theta^{\phi(u)\phi(v)*}, \qquad \forall (u,v)\in\mathcal{C}$$

We then show that this solution is automorphism-invariant. Specifically, it holds for any $\phi' \in \Phi$ that

$$\hat{f}^{\phi'(u)\phi'(v)*}_{\phi'(i)\phi'(j)} = \frac{1}{|\Phi|}\sum_{\phi\in\Phi} f^{\phi(\phi'(u))\phi(\phi'(v))*}_{\phi(\phi'(i))\phi(\phi'(j))} = \frac{1}{|\Phi|}\sum_{\phi\in\Phi} f^{\phi(u)\phi(v)*}_{\phi(i)\phi(j)} = \hat{f}^{uv*}_{ij}.$$

The second equality holds because the automorphism set Φ is a group, so i) the composition of two automorphism mapping functions gives an automorphism in the group and ii) such composition over Φ yields the same Φ. Similarly, the same holds true for $\hat{\theta}^{uv*}$.

$$\hat{\theta}^{\phi'(u)\phi'(v)*} = \frac{1}{|\Phi|}\sum_{\phi\in\Phi} \theta^{\phi(\phi'(u))\phi(\phi'(v))*} = \frac{1}{|\Phi|}\sum_{\phi\in\Phi} \theta^{\phi(u)\phi(v)*} = \hat{\theta}^{uv*}.$$

This construction proves the theorem. □

Theorem 1 implies that there is an optimal solution whose variables in the same group under automorphisms take the same value. We use this insight to reduce the search space of the optimal oblivious routing solution.

IV. Finding an automorphism-invariant optimal solution

We now formulate an optimization problem that targets an automorphism-invariant solution. The formulation defines a new set of representative variables representing groups of variables under the automorphisms. Furthermore, unnecessary constraints and traffic matrices are removed from the formulation, resulting in a tractable iterative algorithm.

A. Representative variables

From Theorem 1, we observe that the optimal values of throughput variables form groups under automorphisms. In particular, the values are identical for every commodity in the same group, whose members can be mapped to one another by some automorphisms, i.e., a group of commodities containing commodity (u,v) is $\{(\phi(u),\phi(v)) : \forall\phi\in\Phi\}$. Therefore, we can pick a commodity in each group as a representative commodity for the group. We denote the set of all representative commodities by $\hat{\mathcal{C}}$ and define a representative throughput variable $\hat{\theta}^{uv}$ for every representative commodity $(u,v)\in\hat{\mathcal{C}}$. Fig. 5 shows an example of representative commodities and throughput variables. Further, we define a function $\pi : \mathcal{C}\to\Phi$ such that the input commodity under the output automorphism is the representative commodity of the input, i.e., $\phi=\pi(u,v)$ and $(\phi(u),\phi(v)) \in \hat{\mathcal{C}}$. Section V-B describes an efficient algorithm to construct $\hat{\mathcal{C}}$ and π.

The representative shares can be defined using the same process. For each representative commodity $(u,v)\in\hat{\mathcal{C}}$, we observe from Theorem 1 that the optimal values of share variables form groups under automorphisms. In particular, the values are identical for every link in the same group, whose members can be mapped to one another by some automorphisms

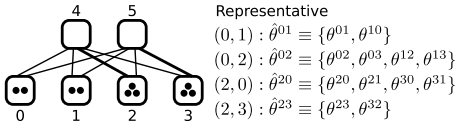

Fig. 5. Throughput variables form four groups. Each group is represented by a representative throughput variable and a corresponding commodity. The set of representative commodities is $\hat{\mathcal{C}} = \{(0,1),(0,2),(2,0),(2,3)\}$. Commodity $(0,1)$ has a representative throughput $\hat{\theta}^{01}$ representing variables θ^{01}, θ^{10}.

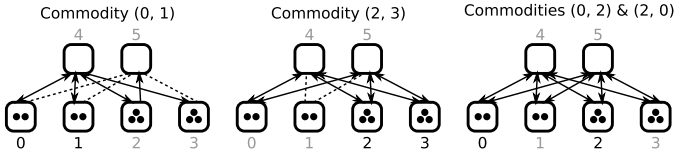

Fig. 6. Representative share variables of each representative commodity. An arrow represents a representative share variable in the direction of the arrow. For example, with commodity $(0,1)$, the link $(0,4)$ has two representative variables $\hat{f}^{01}_{04}, \hat{f}^{01}_{40}$, and the link $(2,5)$ has only one representative variable $\hat{f}^{01}_{25}$. Representative share variables for each representative commodity depends on the network structure and the commodity.

that does not affect (u,v), i.e., the group of links containing link (i,j) is $\{(\phi(i),\phi(j)) : (\phi(u),\phi(v)) = (u,v)\ \exists\phi\in\Phi\}$. Therefore, we can pick a link in each group as a representative link for the group. We denote the set of all representative links for a representative commodity (u,v) by $\hat{\mathcal{L}}^{uv}$ and define a representative share variable $\hat{f}^{uv}_{ij}$ for every $(u,v)\in\hat{\mathcal{C}}$ and every $(i,j)\in\hat{\mathcal{L}}^{uv}$. Fig. 6 shows representative share variables for a simple network. Further, we define a mapping function $\sigma^{uv} : \mathcal{L}\to\Phi$ such that the input link under the output automorphism is the representative link of the input, i.e., $\phi=\sigma^{uv}(i,j)$ and $(\phi(i),\phi(j))\in\hat{\mathcal{L}}^{uv}$. Section V-C describes an efficient algorithm to construct $\hat{\mathcal{L}}^{uv}$ and σ^{uv}.

Since the representative variables of throughput and share have been defined, the last step is to map every variable in (1) to these representatives. We define the mapping of share variables in (1) to their representatives as follows:

$$\varphi\left[f^{uv}_{ij}\right] = \hat{f}^{\phi(u)\phi(v)}_{\phi'(\phi(i)),\phi'(\phi(j))} \quad \forall(u,v)\in\mathcal{C}, \forall(i,j)\in\mathcal{L}$$
$$\text{where } \phi=\pi(u,v) \text{ and } \phi'=\sigma^{\phi(u)\phi(v)}(i,j).$$

Using the above mapping and the representative variables, we formulate an optimization problem targeting an automorphism-invariant optimal solution in Theorem 1 as follows:

$$\begin{aligned}
\text{Maximize } & A\min_{(u,v)\in\hat{\mathcal{C}}}\hat{\theta}^{uv} + \sum_{(u,v)\in\hat{\mathcal{C}}} N^{uv}\hat{\theta}^{uv} \\
\text{Subject to } & \sum_{j\in\mathcal{O}(i)}\varphi\left[f^{uv}_{ij}\right] - \sum_{j\in\mathcal{I}(i)}\varphi\left[f^{uv}_{ji}\right] = \\
& \qquad \hat{\theta}^{uv}\left\{\mathbb{I}[i=u]-\mathbb{I}[i=v]\right\}, \quad \forall i\in\mathcal{S}, \forall(u,v)\in\hat{\mathcal{C}} \\
& \qquad \sum_{(u,v)\in\mathcal{C}} t^{uv}\varphi\left[f^{uv}_{ij}\right] \le C_{ij}, \forall(i,j)\in\mathcal{L}, \forall[t^{uv}]\in\mathcal{T} \\
& \qquad \hat{f}^{uv}_{ij}, \hat{\theta}^{uv}\in\mathbb{R}_+, \qquad \forall(u,v)\in\hat{\mathcal{C}}, \forall(i,j)\in\hat{\mathcal{L}}^{uv},
\end{aligned} \tag{3}$$

where N^{uv} is the number of commodities represented by representative commodity (u,v).

The total numbers of variables and constraints in (3) are significantly reduced by considering the representative variables. Notice that the share conservation constraints are defined over representative commodities $\hat{\mathcal{C}}$ instead of $\mathcal{C}$. The next sections address the challenging robust link capacity constraint.

B. Removing unnecessary link constraints

The number of link capacity constraints in (3) grows like $O(|\mathcal{L}|\,|\mathcal{H}|!)$. However, many constraints are unnecessary and can be removed. We observe that traffic demands on link (i', j') can be aggregated for each representative share.

$$\sum_{(u,v)\in\mathcal{C}} t^{uv}\varphi\left[f^{uv}_{i'j'}\right] = \sum_{(u,v)\in\hat{\mathcal{C}}}\sum_{(i,j)\in\hat{\mathcal{L}}^{uv}} \hat{f}^{uv}_{ij} \sum_{(a,b)\in\mathcal{C}:\varphi\left[f^{ab}_{i'j'}\right]=\hat{f}^{uv}_{ij}} t^{ab}$$
$$= \sum_{(u,v)\in\hat{\mathcal{C}}}\sum_{(i,j)\in\hat{\mathcal{L}}^{uv}} \hat{f}^{uv}_{ij}\alpha^{uv}_{ij}, \tag{4}$$

where $\alpha^{uv}_{ij} = \sum_{(a,b)\in\mathcal{C}:\varphi\left[f^{ab}_{i'j'}\right]=\hat{f}^{uv}_{ij}} t^{ab}$.

Any two links that have the same set of coefficient alphas under some automorphism are duplicate, and we only need to consider one of them as a representative constraint. Let $\hat{\mathcal{L}}$ be the set of representative link constraints. Section V-D provides an efficient algorithm to construct this set. Therefore, the link capacity constraint in (3) can be replaced by

$$\sum_{(u,v)\in\mathcal{C}} t^{uv}\varphi\left[f^{uv}_{ij}\right] \le C_{ij}, \quad \forall(i,j)\in\hat{\mathcal{L}}, \forall[t^{uv}]\in\mathcal{T}. \tag{5}$$

Notice that the reduction from $\mathcal{L}$ to $\hat{\mathcal{L}}$ is by considering the representative variables and traffic demands. Ignoring them and only considering automorphisms of links can lead to an under-constrained formulation and a sub-optimal solution.

C. Traffic matrix selection

Including the entire traffic set $\mathcal{T}$ into the link capacity constraint is impractical due to the continuity of the set. While it is possible to consider the set of extreme points, the number of such points is $O(|\mathcal{H}|!)$ for a simple case of double stochastic matrices when the number of servers per switch is identical.

To alleviate this issue, we construct the traffic set iteratively based on two observations. First, the worst-case traffic load on different representative link is caused by a different set of traffic matrices. Therefore, we define a set of traffic matrices $\mathcal{T}_{ij}$ considered for representative link $(i,j)\in\hat{\mathcal{L}}$ and formulate the optimization parameterized by these sets as follows.

$$\mathsf{R}\left(\{\mathcal{T}_{ij}\}_{(i,j)\in\hat{\mathcal{L}}}\right):$$
$$\text{Maximize} \quad A\min_{(u,v)\in\hat{\mathcal{C}}}\hat{\theta}^{uv} + \sum_{(u,v)\in\hat{\mathcal{C}}} N^{uv}\hat{\theta}^{uv}$$
$$\text{Subject to} \sum_{j\in\mathcal{O}(i)}\varphi\left[f^{uv}_{ij}\right] - \sum_{j\in\mathcal{I}(i)}\varphi\left[f^{uv}_{ji}\right] =$$
$$\hat{\theta}^{uv}\left\{\mathbb{I}\left[i=u\right]-\mathbb{I}\left[i=v\right]\right\}, \quad \forall i\in\mathcal{S}, \forall(u,v)\in\hat{\mathcal{C}}$$
$$\sum_{(u,v)\in\mathcal{C}} t^{uv}\varphi\left[f^{uv}_{ij}\right] \le C_{ij}, \forall(i,j)\in\hat{\mathcal{L}}, \forall[t^{uv}]\in\mathcal{T}_{ij}$$
$$\hat{f}^{uv}_{ij}, \hat{\theta}^{uv}\in\mathbb{R}_+, \qquad \forall(u,v)\in\hat{\mathcal{C}}, \forall(i,j)\in\hat{\mathcal{L}}^{uv}. \tag{6}$$

While solving the above formulation with insufficient traffic matrices in the traffic sets $\{\mathcal{T}_{ij}\}$ leads to a sub-optimal oblivious routing solution, the sub-optimal solution can still guide which traffic matrices should be included in the sets. This leads to the second observation that traffic matrices also have automorphisms, and one matrix can represent many matrices. This observation can be seen from the coefficient alphas in (4). The implication is that including only a small subset of traffic matrices in each traffic set $\mathcal{T}_{ij}$ is sufficient to achieve an optimal oblivious routing solution. We therefore generate representative traffic matrices based on a solution of (6) by solving a simple linear program parameterized by the solution for every link $(i', j')\in\hat{\mathcal{L}}$ as follows.

$$\mathsf{T}_{i'j'}\left(\left\{\hat{f}^{uv}_{ij}\right\}^{(u,v)\in\hat{\mathcal{C}}}_{(i,j)\in\hat{\mathcal{L}}^{uv}}\right):$$
$$\text{Maximize} \sum_{(u,v)\in\mathcal{C}} t^{uv}\varphi\left[f^{uv}_{i'j'}\right]$$
$$\text{Subject to} \sum_{v\in\mathcal{H}\setminus\{u\}} t^{uv} \le H_u, \quad \forall u\in\mathcal{H} \tag{7}$$
$$\sum_{u\in\mathcal{H}\setminus\{v\}} t^{uv} \le H_v, \quad \forall v\in\mathcal{H}$$
$$t^{uv}\in\mathbb{R}_+, \qquad \forall(u,v)\in\mathcal{C}.$$

For every representative link (i', j'), the optimization in (7) finds the worst-case traffic matrix with respect to a given set of representative shares. The obtained traffic matrix is added to $\mathcal{T}_{ij}$, which is considered in (6) for later iterations. This process is summarized next.

D. Iterative algorithm

Algorithm 1: Optimal oblivious routing

```
Initialize D_ij ← {[t^uv]_init} for every (i,j) ∈ L̂
while ⋃_{(i,j)∈L̂} D_ij ≠ ∅ do
    T_ij ← T_ij ∪ D_ij for every (i,j) ∈ L̂
    {f̂^uv_ij}^{(u,v)∈Ĉ}_{(i,j)∈L̂^uv}, {θ̂^uv}^{(u,v)∈Ĉ} ← R({T_ij}_{(i,j)∈L̂})
    for (i',j') ∈ L̂ do
        [t^uv] ← T_{i'j'}({f̂^uv_ij}^{(u,v)∈Ĉ}_{(i,j)∈L̂^uv})
        if Σ_{(u,v)∈C} t^uv φ[f^uv_{i'j'}] > C_{i'j'} then
            D_{i'j'} ← {[t^uv]}
        else
            D_{i'j'} ← ∅
return {f̂^uv_ij}^{(u,v)∈Ĉ}_{(i,j)∈L̂^uv}, {θ̂^uv}^{(u,v)∈Ĉ}
```

In Algorithm 1, the traffic sets are initialized with an initial traffic matrix. The algorithm iteratively finds a routing solution according to these traffic sets. The routing solution is used to generate more traffic matrices, which in turn help improving the next routing solution, by including those that violate link

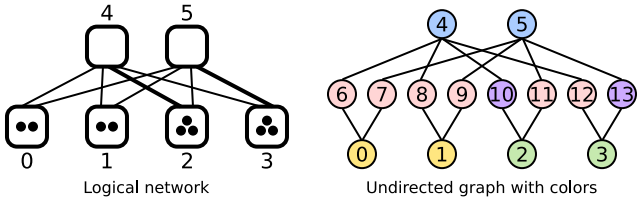

Fig. 7. A logical network is transformed to an undirected graph with colors.

capacity constraints into the traffic sets. This process continues until no traffic matrices violating the link capacity constraints are found. Thus, the final routing solution is optimal because it is an automorphism-invariant solution that optimizes the objective and satisfies all conservation constraints in (6) and that no traffic matrices violates capacity under this solution.

The initial traffic matrix $[t^{uv}]_{\text{init}}$ in Algorithm 1 can significantly reduce the number of iterations required to obtain the optimal solution. We set this matrix to the solution of the weighted bipartite matching problem, where the node sets are $\mathcal{U} = \mathcal{V} = \mathcal{H}$, and the weight between any two nodes $(u, v) \in \mathcal{U} \times \mathcal{V}$ equals the product of their minimum path length in the network topology times their maximum traffic load $\min(H_u, H_v)$. This initialization is inspired by the works in [21], [24], [25], in which all switches have the same number of servers.

V. Efficient Implementation

Algorithm 1 and its sub-routines in the previous section rely heavily on representative sets $\hat{\mathcal{C}}, \hat{\mathcal{L}}^{uv}, \hat{\mathcal{L}}$, and the mapping functions π and σ^{uv}. Constructing them from the automorphism set Φ, whose size grows exponentially, is a combinatorial problem. This section describes polynomial-time algorithms for efficient construction, based on automorphism generators.

A. Generators of automorphisms

Generators are a smaller set of automorphisms that can generate a whole set of automorphisms. Specifically, every automorphism in the set is a combination of the generators. This set can be obtained from off-the-shelf software, such as `nauty` [26], for an undirected graph with vertices, edges, and a set of colors assigned to the vertices.

Our datacenter network model with link capacities and numbers of servers at each switch can be transformed to an undirected graph with colors as shown in Fig. 7. Every switch is translated to a vertex in the graph. To preserve the number of servers at each switch under automorphism, an identical color is assigned to the vertices whose number of servers are the same, and different colors are assigned to vertices having different numbers of servers. For links and capacities, each link is transformed to an auxiliary vertex with two edges. One end of both edges is connected to the auxiliary vertex, and the other ends connect to the two switches to which the original link is adjacent. To preserve link capacity under automorphism, the auxiliary vertices of links with the same capacity are assigned the same color, and auxiliary vertices of links with different capacities are assigned different colors.

The above transformation yields an undirected graph with a set of colors as an input to the off-the-shelf software that outputs the generators. We denote the set of these generators by $\hat{\Phi}$, whose generated automorphisms satisfy Definition 1. It is used to efficiently generate representative sets and mapping functions used extensively in the previous section.

B. Representative commodity

The representative commodity set $\hat{\mathcal{C}}$ and the mapping functions π defined in Section IV-A are constructed efficiently by Algorithm 2.

Algorithm 2: Representative commodity construction

```
Initialize empty sets Q, P, Ĉ and dictionaries D, π
for (u, v) ∈ C do
    if (u, v) ∈ P then
        continue
    Ĉ ← Ĉ ∪ {(u, v)}
    D(u, v) ← φ_identity
    Q ← Q ∪ {(u, v)}
    while Q is not empty do
        Pop (a, b) from Q
        P ← P ∪ {(a, b)}
        for φ ∈ Φ̂ do
            if (φ(a), φ(b)) ∉ P then
                Q ← Q ∪ {(φ(a), φ(b))}
                D(φ(a), φ(b)) ← φ(D(a, b))
for (u, v) ∈ C do
    π(u, v) ← (D(u, v))^{-1}
return Representative set Ĉ and function π
```

Algorithm 2 searches over the commodity set. It picks a commodity, assigns it as a representative commodity, and finds all represented commodities. The search process utilizes the generator set $\hat{\Phi}$, instead of the exponentially large automorphism set Φ, to find represented commodities. The algorithm visits each commodity once, so the complexity is $O\left(|\mathcal{C}|\left|\hat{\Phi}\right|\right)$. Once all represented commodities of the picked representative are found, the algorithm picks an unpicked commodity and continues the process. In the process, the dictionary D keeps track of automorphisms that map representatives to their represented commodities. The last step inverts each automorphism in D and constructs the dictionary π storing automorphisms that map represented commodities to their representatives.

C. Representative share

For each representative commodity (u, v), the set of representative shares $\hat{\mathcal{L}}^{uv}$ and the mapping functions σ^{uv} defined in Section IV-A are constructed efficiently by Algorithm 3.

Given a representative commodity (u, v), the algorithm first constructs a commodity-preserved generator set $\hat{\Phi}^{uv}$ containing all generators that have no effect on the given commodity. The algorithm then searches over the links set using these commodity-preserved generators. It picks a link as

Algorithm 3: Representative share construction

Initialize empty sets $\mathcal{Q}, \mathcal{P}, \hat{\mathcal{L}}^{uv}$ and dictionary D, σ^{uv}
$\hat{\Phi}^{uv} \leftarrow \left\{\phi \in \hat{\Phi} : (u,v) = (\phi(u), \phi(v))\right\}$
for $(i,j) \in \mathcal{L}$ **do**
  if $(i,j) \in \mathcal{P}$ **then**
    continue
  $\hat{\mathcal{L}}^{uv} \leftarrow \hat{\mathcal{L}}^{uv} \cup \{(i,j)\}$
  $D(i,j) \leftarrow \phi_{\text{identity}}$
  $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(i,j)\}$
  while $\mathcal{Q}$ *is not empty* **do**
    Pop (a,b) from $\mathcal{Q}$
    $\mathcal{P} \leftarrow \mathcal{P} \cup \{(a,b)\}$
    for $\phi \in \hat{\Phi}^{uv}$ **do**
      if $(\phi(a), \phi(b)) \notin \mathcal{P}$ **then**
        $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(\phi(a), \phi(b))\}$
        $D(\phi(a), \phi(b)) \leftarrow \phi(D(a,b))$
for $(i,j) \in \mathcal{L}$ **do**
  $\sigma^{uv}(i,j) \leftarrow (D(i,j))^{-1}$
return Representative set $\hat{\mathcal{L}}^{uv}$ and function σ^{uv}

a representative link and finds all corresponding represented links. This process utilizes the generator set $\hat{\Phi}^{uv}$ to search for represented links. The algorithm visits each link once, so its complexity is $O\left(|\mathcal{L}|\left|\hat{\Phi}^{uv}\right|\right)$. Once all represented links of the picked representative are found, the algorithm picks an unpicked link and continues the process. The dictionary D keeps track of automorphisms that map representatives to their represented links. The last step inverts each automorphism in D and constructs the dictionary σ^{uv} storing automorphisms that map represented links to their representatives.

D. Representative link capacity constraint

The set of representative links $\hat{\mathcal{L}}$ for the capacity constraints in (6) can be constructed by removing redundant constraints under automorphisms. More precisely, two constraints of links (x,y) and (g,h) are redundant when 1) their link capacities are identical, $C_{xy} = C_{gh}$, and 2) their traffic loads in (4) are identical under the traffic set $\mathcal{T}$. The former is easy to identify, but the latter is challenging as the comparison is over the whole traffic set. Theorem 2 simplifies this comparison.

Theorem 2. *Given links (x,y) and (g,h), their capacity constraints in terms of representative shares are*

$$\sum_{(u,v)\in\hat{\mathcal{C}}} \sum_{(i,j)\in\hat{\mathcal{L}}^{uv}} \hat{f}_{ij}^{uv} \sum_{(a,b)\in\mathcal{C}:\varphi\left[f_{xy}^{ab}\right]=\hat{f}_{ij}^{uv}} t^{ab} \le C_{xy} \quad \forall [t^{uv}] \in \mathcal{T},$$

$$\sum_{(u,v)\in\hat{\mathcal{C}}} \sum_{(i,j)\in\hat{\mathcal{L}}^{uv}} \hat{f}_{ij}^{uv} \sum_{(a,b)\in\mathcal{C}:\varphi\left[f_{gh}^{ab}\right]=\hat{f}_{ij}^{uv}} t^{ab} \le C_{gh} \quad \forall [t^{uv}] \in \mathcal{T}.$$

The two constraints are identical when 1) their link capacities are identical, i.e., $C_{xy} = C_{gh}$, and 2) their numbers of traffic demands associated with a representative share is identical for every representative share such that the following holds for every $(u,v) \in \hat{\mathcal{C}}$ and every $(i,j) \in \hat{\mathcal{L}}^{uv}$:

$$\left|\left\{(a,b) \in \mathcal{C} : \varphi\left[f_{xy}^{ab}\right] = \hat{f}_{ij}^{uv}\right\}\right| = \left|\left\{(a,b) \in \mathcal{C} : \varphi\left[f_{gh}^{ab}\right] = \hat{f}_{ij}^{uv}\right\}\right|. \quad (8)$$

Proof. The first condition is straightforward. We prove the second condition as follows. Considering link (i',j'), we observe that every commodity in $\left\{(a,b) \in \mathcal{C} : \varphi\left[f_{i'j'}^{ab}\right] = \hat{f}_{ij}^{uv}\right\}$ is represented by a common commodity (u,v) as they share the same representative share variable $\hat{f}_{ij}^{uv}$. It follows that these commodities generate an identical set of traffic demands, so the set of aggregated traffic demands associated with the representative share variable are identical if the numbers of members in both sets are the same, i.e., when (8) holds, then

$$\left\{\sum_{(a,b)\in\mathcal{C}:\varphi\left[f_{xy}^{ab}\right]=\hat{f}_{ij}^{uv}} t^{ab} : \forall [t^{ab}] \in \mathcal{T}\right\} = \left\{\sum_{(a,b)\in\mathcal{C}:\varphi\left[f_{gh}^{ab}\right]=\hat{f}_{ij}^{uv}} t^{ab} : \forall [t^{ab}] \in \mathcal{T}\right\}.$$

Therefore, comparing the number of traffic demands associated with each representative share is sufficient. □

The implication of Theorem 2 is that we only need to compare the number of traffic demand terms associated with each representative share to determine whether two constraints are identical under automorphism. This leads to a simpler construction of the representative capacity constraints.

The set of representative capacity constraints is constructed by computing the distribution of the numbers of traffic demand terms associated with representative shares. Then, for every group of links having the same distribution, we take one link from the group and use it as the representative of the group.

VI. Evaluation

The performance, scalability, and generality of Algorithm 1 is evaluated over three potential applications. The algorithm is implemented in Python 3. The generators of automorphisms is obtained from `nauty` [26]. All linear programs are solved by Gurobi [27]. Further, all optimal routing solutions are double checked for capacity violation using the method in [24].

A. Throughput performance under partially deployed FatTree

A fully deployed FatTree topology [1], constructed from 32-port switches, can accommodate 8192 servers. In practice, this topology is incrementally deployed in blocks when additional servers are needed. However, the routing for the fully deployed topology is not optimal for the partially deployed one due to the imbalance at the core switches, i.e., each core switch in Fig. 1 has two links to one block and a link to another block. Therefore, we apply our algorithm to find the optimal oblivious routing for partially deployed FatTree with different numbers of aggregation blocks. Note that the fully deployed

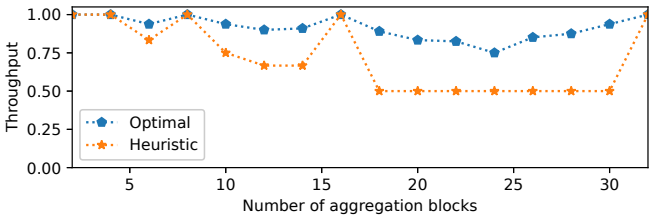

Fig. 8. Throughput values under partially deployed FatTree. At 30 aggregation blocks, the throughput improvement is 87.5%.

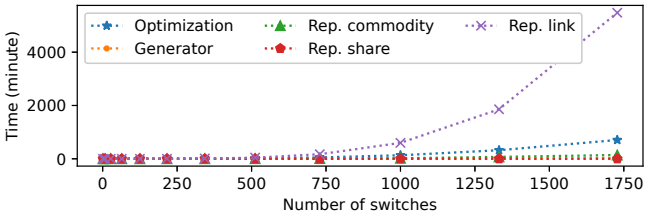

Fig. 9. The overall computation times at different FatClique's sizes are breakdown into computing automorphism generators (Generator), finding representative commodities by Algorithm 2 (Rep. commodity), finding representative shares by Algorithm 3 (Rep. share), finding representative links utilizing Theorem 2 (Rep. link), and the execution time of Algorithm 1 (Optimization).

topology has 32 aggregation blocks. The result is shown in Fig. 8 and is compared to a heuristic algorithm in [28], which tries to balance the imbalance. Our algorithm yields $12.5\% - 87.5\%$ throughput improvement over the heuristic algorithm when their throughput values are different.

B. Scalability under FatClique

The FatClique topology in [11] has been proposed for high manageability datacenter networks without routing. Our algorithm could provide this missing routing, which is optimal according to the formulation in (1). The computation time of Algorithm 1 at different sizes of FatClique is shown in Fig. 9. We vary the sizes of FatClique from 8 to 1728 switches. While finding the representative links is the most time consuming part, it can be pre-computed before executing Algorithm 1.

Fig. 10 shows the sizes of the strawman formulation in (1) and the automorphism-invariant formulation in (6) in terms of numbers of variables and constraints assuming only one traffic matrix is considered, $|\mathcal{T}| = 1$. It is easy to see that the latter formulation is much smaller than the former one. In addition, Fig. 11 shows the numbers of traffic matrices required for finding the optimal oblivious routing solutions in formulations (1) and (6). Again, the numbers of required traffic matrices in Algorithm 1 is much smaller than the set of all extreme points.

C. Structured topology with non-uniformity

Our algorithm is applicable for a more general structured topology with non-uniform link capacities and server distribution as illustrated in Fig. 12. The top-left plot shows the network with 4 groups of switches, $\{0,1\}, \{2,3,4\}, \{5,6,7,8\}, \{9,10,11\}$, where the first three groups have different numbers of servers per switch and the last group has none. Logical links with different thickness have different capacities. The other plots show the optimal oblivious routing for different

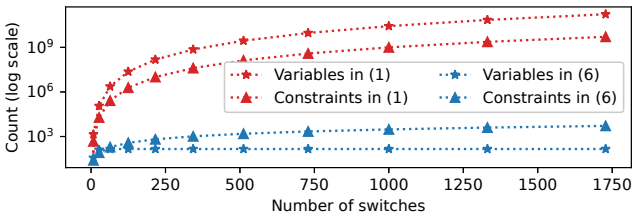

Fig. 10. The numbers of variables and constraints, assuming $|\mathcal{T}| = 1$, in formulations (1) and (6) are plotted at different sizes of FatClique. The maximum numbers for formulation (6) are respectively 144 and 5185 when topology has 1728 switches.

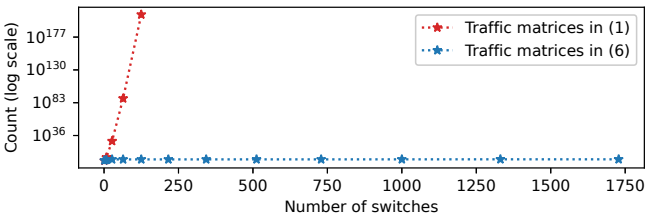

Fig. 11. The numbers of traffic matrices per link capacity constraint in formulations (1) and (6) are plotted at different sizes of FatClique. The maximum number for formulation (6) is 81 when topology has 1331 switches, while the numbers beyond 10^{308} are not plotted for formulation (1) with more than 125 switches.

representative commodities. It is worth mentioning that the optimal routing solution for each representative commodity can be different. For example, commodities $(0,1)$ only uses one-hop intermediate switches, while commodity $(0,2)$ disperses traffic to all switches. This behavior differs from [15] where traffic demands from every commodity are dispersed to the same set of intermediate switches.

VII. Conclusion

This paper presents an iterative algorithm to find the optimal oblivious routing for structured topologies. The algorithm utilizes the insight that the optimal routing solution is automorphism invarient in order to reduce the solution space and complexity. This algorithm is applicable for designing oblivious routing for existing datacenter network topologies and future structured topologies with non-uniform link capacities and server distribution.

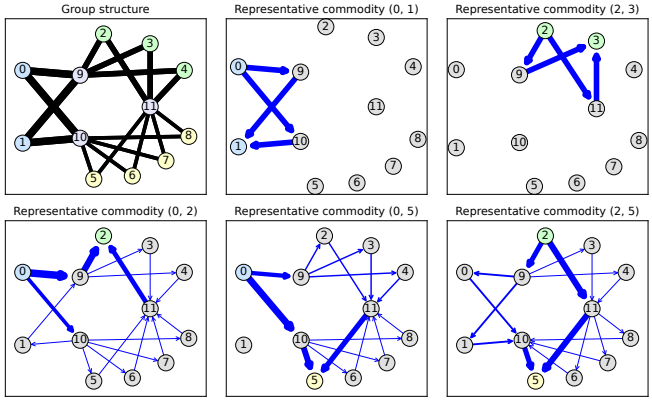

Fig. 12. Optimal oblivious routing for non-uniform and structured topology

REFERENCES

[1] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 63–74, Aug. 2008. [Online]. Available: https://doi.org/10.1145/1402946.1402967

[2] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Vl2: A scalable and flexible data center network," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 51–62, Aug. 2009. [Online]. Available: https://doi.org/10.1145/1594977.1592576

[3] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano, A. Kanagala, J. Provost, J. Simmons, E. Tanda, J. Wanderer, U. Hölzle, S. Stuart, and A. Vahdat, "Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 183–197, Aug. 2015. [Online]. Available: https://doi.org/10.1145/2829988.2787508

[4] A. Andreyev. Introducing data center fabric, the next-generation facebook data center network. [Online]. Available: https://engineering.fb.com/2014/11/14/production-engineering/introducing-data-center-fabric-the-next-generation-facebook-data-center-network/

[5] J. H. Ahn, N. Binkert, A. Davis, M. McLaren, and R. S. Schreiber, "Hyperx: Topology, routing, and packaging of efficient large-scale networks," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, ser. SC '09. New York, NY, USA: Association for Computing Machinery, 2009. [Online]. Available: https://doi.org/10.1145/1654059.1654101

[6] J. Kim, W. J. Dally, S. Scott, and D. Abts, "Technology-driven, highly-scalable dragonfly topology," in *2008 International Symposium on Computer Architecture*, 2008, pp. 77–88.

[7] M. Besta and T. Hoefler, "Slim fly: A cost effective low-diameter network topology," in *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2014, pp. 348–359.

[8] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey, "Jellyfish: Networking data centers randomly," in *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. San Jose, CA: USENIX Association, Apr. 2012, pp. 225–238. [Online]. Available: https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/singla

[9] A. Valadarsky, G. Shahaf, M. Dinitz, and M. Schapira, "Xpander: Towards optimal-performance datacenters," in *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 205–219. [Online]. Available: https://doi.org/10.1145/2999572.2999580

[10] A. Singla, P. B. Godfrey, and A. Kolla, "High throughput data center topology design," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. Seattle, WA: USENIX Association, Arp. 2014, pp. 29–41. [Online]. Available: https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/singla

[11] M. Zhang, R. N. Mysore, S. Supittayapornpong, and R. Govindan, "Understanding lifecycle management complexity of datacenter topologies," in *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. Boston, MA: USENIX Association, Feb. 2019, pp. 235–254. [Online]. Available: https://www.usenix.org/conference/nsdi19/presentation/zhang

[12] V. Harsh, S. A. Jyothi, and P. B. Godfrey, "Spineless data centers," in *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*, ser. HotNets '20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 67–73. [Online]. Available: https://doi.org/10.1145/3422604.3425945

[13] D. Thaler and C. Hopps. Rfc2991: Multipath issues in unicast and multicast next-hop selection. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc2991

[14] R. Zhang-Shen and N. McKeown, "Guaranteeing quality of service to peering traffic," in *IEEE INFOCOM 2008 - The 27th Conference on Computer Communications*, 2008, pp. 1472–1480.

[15] M. Kodialam, T. V. Lakshman, J. B. Orlin, and S. Sengupta, "Preconfiguring ip-over-optical networks to handle router failures and unpredictable traffic," *IEEE Journal on Selected Areas in Communications*, vol. 25, no. 5, pp. 934–948, 2007.

[16] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'10. USA: USENIX Association, 2010, p. 19.

[17] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 123–137, Aug. 2015. [Online]. Available: https://doi.org/10.1145/2829988.2787472

[18] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "Dcell: A scalable and fault-tolerant network structure for data centers," in *SIGCOMM08*. Association for Computing Machinery, Inc., August 2008. [Online]. Available: https://www.microsoft.com/en-us/research/publication/dcell-a-scalable-and-fault-tolerant-network-structure-for-data-centers/

[19] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, S. Lu, and G. Lv, "Bcube: A high performance, server-centric network architecture for modular data centers," in *ACM SIGCOMM*. Association for Computing Machinery, Inc., August 2009. [Online]. Available: https://www.microsoft.com/en-us/research/publication/bcube-a-high-performance-server-centric-network-architecture-for-modular-data-centers/

[20] Broadcom. Tomahawk4 / bcm56990 series. [Online]. Available: https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm56990-series

[21] P. Namyar, S. Supittayapornpong, M. Zhang, M. Yu, and R. Govindan, "A throughput-centric view of the performance of datacenter topologies," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, ser. SIGCOMM '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 349–369. [Online]. Available: https://doi.org/10.1145/3452296.3472913

[22] S. Supittayapornpong, B. Raghavan, and R. Govindan, "Towards highly available clos-based wan routers," in *Proceedings of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 424–440. [Online]. Available: https://doi.org/10.1145/3341302.3342086

[23] D. Applegate and E. Cohen, "Making intra-domain routing robust to changing and uncertain traffic demands: Understanding fundamental tradeoffs," in *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '03. New York, NY, USA: Association for Computing Machinery, 2003, p. 313–324. [Online]. Available: https://doi.org/10.1145/863955.863991

[24] B. Towles and W. Dally, "Worst-case traffic for oblivious routing functions," *IEEE Computer Architecture Letters*, vol. 1, no. 1, pp. 4–4, 2002.

[25] S. A. Jyothi, A. Singla, P. B. Godfrey, and A. Kolla, "Measuring and understanding throughput of network topologies," in *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2016, pp. 761–772.

[26] B. D. McKay and A. Piperno, "Practical graph isomorphism, ii," *Journal of Symbolic Computation*, vol. 60, pp. 94–112, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0747717113001193

[27] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2021. [Online]. Available: https://www.gurobi.com

[28] J. Zhou, M. Tewari, M. Zhu, A. Kabbani, L. Poutievski, A. Singh, and A. Vahdat, "Wcmp: Weighted cost multipathing for improved fairness in data centers," in *Proceedings of the Ninth European Conference on Computer Systems*, ser. EuroSys '14. New York, NY, USA: Association for Computing Machinery, 2014. [Online]. Available: https://doi.org/10.1145/2592798.2592803