DESIGNING OPTIMAL COMPACT OBLIVIOUS ROUTING FOR DATACENTER NETWORKS IN POLYNOMIAL TIME

KANATIP CHITAVISUTTHIVONG

ID 1820402

A THESIS SUBMITTED TO

VIDYASIRIMEDHI INSTITUTE OF SCIENCE AND TECHNOLOGY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF MASTER OF ENGINEERING

IN INFORMATION SCIENCE AND TECHNOLOGY

2024

Designing Optimal Compact Oblivious Routing for Datacenter Networks in Polynomial Time

Abstract

Kanatip Chitavisutthivong

Recent datacenter network topologies are shifting towards heterogeneous and structured topologies for high throughput, low cost, and simple manageability. However, they rely on sub-optimal routing approaches that fail to achieve their designed capacity. This thesis proposes a process for designing optimal oblivious routing that is programmed compactly on programmable switches. The process consists of three steps in tandem. We first transform a robust optimization problem for designing oblivious routing into a linear program, which is solvable in polynomial time but cannot scale for datacenter topologies. We then prove that the repeated structures in a datacenter topology lead to a structured optimal solution. We use this insight to formulate a scalable linear program, so an optimal oblivious routing solution is obtained in polynomial time for large-scale topologies. For real-world deployment, the optimal solution is converted into forwarding rules for programmable switches with stringent memory. With this constraint, we utilize the repeated structures in the optimal solution to group the forwarding rules, resulting in compact forwarding rules with a much smaller memory requirement. Furthermore, one may possess knowledge of the range of traffic demands within a datacenter network, which can be obtained from historical data or through capacity planning. Hence, we also extend the design process to accommodate such scenarios. Extensive evaluations show our process i) obtains optimal solutions faster and more scalable than a state-of-the-art technique and ii) reduces the memory requirement by no less than 90% for most

considered topologies.

Keyword: Oblivious routing, compact routing, datacenter networks,
graph automorphism

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Trend of Designing Datacenter Network Topologies

Nowadays, many private companies have established their own datacenters to operate a range of services with various requirements and applications, such as social networks, search engines, video-on-demand, cloud computing, machine learning, and more. With the rapid expansion of datacenter networks to meet the growing demands of the AI and cloud computing era, the trend of designing network topologies is currently shifting towards heterogeneous, *structured topologies* for high capacity, low cost, and ease of management [1, 2, 3, 4]. This attempts to replace the conventional folded-Clos family, including FatTree [5], Google's Jupiter [6], Facebook's Fabric [7], and Microsoft's VL2 [8]. Therefore, alternative topologies have been proposed, such as Xpander [3], FatClique [2], DRing [9], SlimFly [10], and other server-centric networks where servers have routing capability, such as BCube [11].

Suppose a high-capacity datacenter network, datacenter administrators expect that its network resources will be utilized efficiently. This relies heavily on *traffic routing* in achieve this efficiency [12].

1.2 Routing in Datacenter Networks

Given a network topology and a flow of traffic demand from a source server to a destination server, the routing algorithm is used to determine paths from the source to the destination and allocate the fractions of traffic demand over these paths[1]. In practice, however, the routing of traffic demands for each source and destination occurs simultaneously within the network, where links have limited capacity and are shared among the flows. This can lead to issues such as network congestion, which could

[1]To alleviate the congestion and enhance fault tolerance, network administrators often employ *multi-path routing*, which routes traffic along multiple alternative paths rather than a single path.

degrade network performance (e.g., reduced network throughput). Particularly, when the traffic load on certain links is heavy while others are underutilized[2].

Routing traffic inside datacenter networks can be categorized into two categories: *dynamic routing* and *oblivious routing*. Dynamic routing regularly adjusts routes and fractions of traffic demands over routes based on network states, such as queue occupancy and traffic demand, to ensure that the network resources will be utilized efficiently [10, 11, 13, 14]. In fact, the network state (e.g., traffic demand) in datacenter networks are highly dynamic and difficult-to-predict in nature [8, 15, 16]. Thus, dynamic routing must react immediately to these changes. However, this come at the cost of complex network routing system and may require special hardware. On the other hand, oblivious routing is much simpler. For oblivious routing, traffic demands are routed according to pre-determined routes and fractions of the demands over each route, which are designed to be *robust* to changing traffic demands [17, 18, 19, 20, 20, 21, 22]. In particular, the routing is oblivious to the current traffic demand. Therefore, neither frequent re-configuration nor dynamic routing is required when traffic demand changes. However, this obliviousness may lead to increased network congestion compared to dynamic routing operating under the same network state. Despite this potential drawback, oblivious routing remains widely used in production networks due to its robustness and simplicity [6, 7, 8]. As illustrated in Figure 1.1, for example, the folded-Clos family [5, 6, 7, 8] often employs the Equal-Cost Multi Paths (ECMP) routing approach [17] that distributes traffic demand evenly among all equal-cost paths, regardless of traffic demand. Another oblivious routing approach is Valiant Load Balancing (VLB) [18, 19, 20], also known as two-stage routing, because the routing is performed in two stages, i.e., a traffic is equally split to intermediate nodes, before being forwarded to its final destination.

[2]The network congestion can be considered in two different types: link-level congestion and node-level congestion. To simplify the model, in this thesis, our focus is solely on link-level congestion.

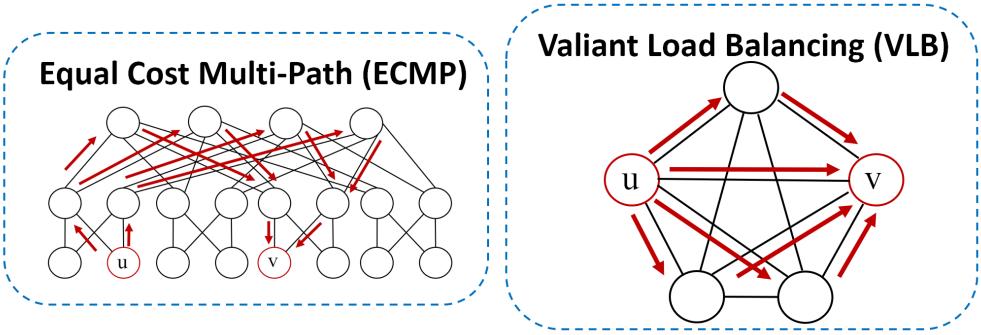

Figure 1.1 The widely used oblivious routing approaches. (Left) ECMP is employed on FatTree topology. (Right) VLB is employed on clique topology.

1.3 Research Motivation

Due to the advantages of oblivious routing, the design of *optimal oblivious routing* that minimizes network congestion has been studied for decades [20, 21, 22]. Hereafter, *network congestion* (or maximum congestion ratio) refers to the congestion ratio of the most congested link. Formally, this is defined as the maximum value of the congestion ratio among all links in the network. The congestion ratio of a link is the proportion of total traffic load on the link to its capacity.

While, the widely used ECMP and VLB approaches have been proven to be optimal oblivious routing methods for specific network topologies—ECMP for the folded-Clos networks [23] and VLB for clique networks [19]—they are sub-optimal for the alternative topologies such as Xpander, FatClique, DRing, SlimFly, and BCube, due to topological differences. This motivates us to pose a research problem: how to design optimal oblivious routing for general datacenter networks, including those alternative topologies to the folded-Clos family.

Generally, designing an optimal oblivious routing for an arbitrary network topology is equivalent to solving a robust multi-commodity flow problem[3] [20, 21, 22]. The works in [20, 21] views this problem as a game[4] and derived a linear program for

[3]The robust multi-commodity flow problem is a variation of the multi-commodity flow problem that optimize under a broad set of possible traffic demands.

[4]The problem is viewed as a two-player zero-sum game: Player 1 seeks routing that minimizes network congestion given Player 2's traffic demand, while Player 2 aims to generate the traffic demand that maximizes network congestion given Player 1's routing strategy.

intra-domain networks. However, the networks they studied, are considered *small-scale* compared to datacenter networks which typically have more than a thousand nodes and tens of thousands of links. As the size of datacenter networks is rapidly expanding to accommodate the increasing demands in the AI and cloud computing era, the recent work [22] exploited the repeated network structures commonly found in *structured topologies*, which are typically used in existing datacenter networks [2, 3, 5, 6, 7, 8, 9, 10, 11]. This approach aims to reduce the complexity of the robust multi-commodity flow problem. Therefore, an optimal oblivious routing solution is obtained for larger network sizes.

In summary, the works [20, 21] can obtain the optimal routing solution in polynomial time[5] (in the size of input instance, which is extremely large for large-scale networks). However, the second work [22] can scale to larger networks but could take non-polynomial time due to the complexity of robust optimization[6]. This leads to a research question:

> **Research question 1:** Could we design optimal oblivious routing in polynomial time that also scales for large datacenter networks to achieve the best of both worlds?

The memory constraint is another important issue for the real-world deployment of oblivious routing. After an optimal routing solution is obtained, it is converted to forwarding rules that determine how traffic is split at each switch in a network. The forwarding rules could be configured into programmable switches with traffic-splitting capability[7] by using these splitting techniques [27, 28, 29, 30]. Table 1.1 illustrates an example of a forwarding rules table at switch i which converted from a routing solution. The limited size of memory allocated for the forwarding rules in a switch becomes a concern, when dealing with a large number of forwarding rules, especially during the deployment of optimal routing for large-scale datacenter networks with thousands of commodities (source-destination pairs). Several techniques to reduce

[5] Since there exists a polynomial-time algorithm for solving linear programming problems [24, 25, 26].

[6] They solve the robust multi-commodity flow problem by introducing an iterative algorithm that alternately solves two linear programs. (For further details, please see Chapter 3.1.2.)

[7] The switches that enable operators to control packet forwarding behavior, include commodity switches (e.g., OpenFlow switches) and P4 data plane programmable switches.

Table 1.1 An example of a forwarding rules table at switch i, which has three links connecting to switches j, k, and l, respectively. There are two commodities (source-destination pairs) whose traffic are traversed via switch i.

Commodity	Next-hop	Split ratio
(u,v)	j	1/3
	k	2/3
(a,b)	j	1/4
	k	1/4
	l	1/2

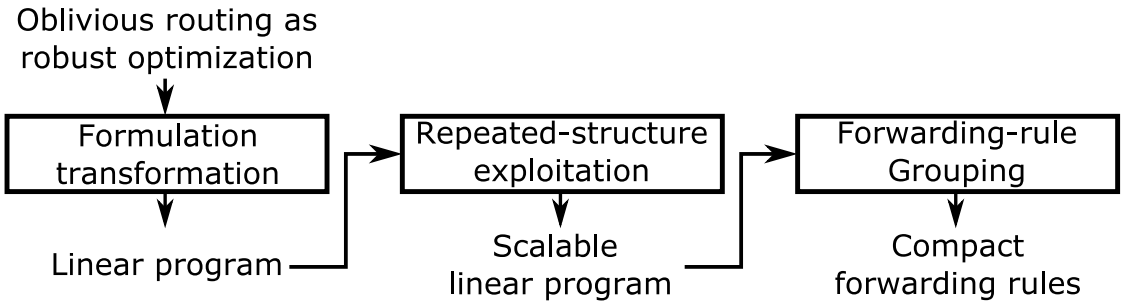

Figure 1.2 The design process of optimal compact oblivious routing.

memory requirements have been studied [27, 28, 31, 32]. For instance, the approximation approaches [27, 28] trade off between optimality and available memory by approximating the routing solution and then truncating it to fewer rules. While techniques in [31, 32] are customized specifically for designing oblivious routing that based on Räcke's hierarchical decomposition tree [33]. Nevertheless, the works that focus on designing optimal oblivious routing by solving optimization problems [20, 21, 22] did not emphasize the reduction of memory requirement. This leads to another research question:

> **Research question 2:** Could we reduce the memory requirement given an optimal oblivious routing solution?

In this thesis, we propose a process for designing optimal compact oblivious routing for datacenter networks to address the two research questions above, as shown in Figure 1.2. We first formulate an oblivious routing problem as robust optimization for general datacenter networks, including those alternative topologies to the folded-Clos family. We then transform the robust optimization into a linear program using decomposition and duality techniques. This transformation reduces the complexity of optimal oblivious routing design to just solving a *polynomial-time* solvable linear program. However, the linear program becomes tractable for small-scale datacenter networks, due to its problem size—in terms of the number of variables and constraints—becoming excessively large in the context of large-scale datacenter networks.

Since datacenter networks usually have repeated structures, we show the existence of an automorphism-invariant optimal solution, whose variables have repeated values. From this insight, we formulate a *scalable* linear program—a linear program with a significantly reduced problem size, allowing it to scale to larger network sizes. As a result, we can design optimal oblivious routing for large-scale datacenter networks with polynomial-time complexity, which addresses the first research question. For the second research question, we exploit the structure of the optimal oblivious routing solution, obtained from the scalable linear program, to group the forwarding rules. This grouping compacts the forwarding rules and significantly reduces memory requirement, which addresses the second research question. In addition, some datacenter administrators might have insights into traffic demand ranges within a datacenter network, such as the range of traffic demand between a source-destination pair. Therefore, we extend the design process to accommodate such situations.

Extensive evaluations of the scalable linear program on various datacenter network topologies and sizes show that the linear program is faster and more scalable than the state-of-the-art technique in [22] and another linear formulation in [21]. Moreover, the forwarding rules grouping significantly reduces forwarding rules' memory requirement by no less than 90% for the majority of considered topologies. We also provide possible applications of the optimal compact oblivious routing.

1.4 Organizations of the Thesis

This thesis is organized as follows:

Chapter 1 introduces the trend of designing datacenter network topologies and background of routing in datacenter networks, as well as the research challenges and motivations.

Chapter 2 presents research objectives, research scopes, and contributions of the thesis.

Chapter 3 provides the literature review on the brief history of the development of oblivious routing studies, which focus on the objective of minimizing network congestion, and related works about reduction of forwarding rules.

Chapter 4 describes the model of datacenter networks and the oblivious routing problem. It discusses the transformation of the problem into a linear program and the exploitation of the repeated network structures to improve scalability. Additionally, the chapter introduces the modification to support customized traffic models, and presents a grouping technique to compact forwarding rules for real-world deployment.

Chapter 5 presents extensive evaluations of our scalable linear program on various datacenter network topologies and sizes. Its scalability is evaluated in terms of computation times and problem sizes. Additionally, the chapter provides an evaluation of the efficiency of our forwarding rules grouping method, and demonstrates how our work is applicable to an existing server-centric topology, as well as how it incorporates traffic demand knowledge.

Chapter 6 concludes the thesis and outlines potential directions for future research.

1.5 Publications Based on This Thesis

- Chitavisutthivong K, So-In C, Supittayapornpong S. **Designing optimal compact oblivious routing for datacenter networks in polynomial time.** IEEE INFOCOM 2023 - IEEE Conference on Computer Communications; 2023. p. 1-10.

- Chitavisutthivong K, So-In C, Supittayapornpong S. Designing optimal compact oblivious routing for datacenter networks in polynomial time. **IEEE/ACM Transactions on Networking.** 2024:(Under Review)

Chapter 2

Objectives, Scopes, and Contributions

2.1 Research Objectives

This thesis aims to develop a polynomial-time process for designing optimal compact oblivious routing for datacenter networks. The main objectives can be summarized as follows:

Objective 1 (Optimal Oblivious Routing): To develop a polynomial-time and scalable approach to design optimal oblivious routing for datacenter networks.

Objective 2 (Compact Routing): To design a technique for reducing the memory requirement given an optimal oblivious routing solution in real-world deployment.

2.2 Scopes of Research

The scope of this thesis is summarized as follows:

1) We focus on designing optimal compact oblivious routing for datacenter networks that minimizes *network congestion*, specifically at the link level, where links in the network have limited capacity. However, network congestion also be considered at the node level or as a combination of both. To simplify the problem, we only focus on *link-level congestion*, assuming that all network nodes, such as switches or servers, have an unlimited buffer size and can process and forward traffic without internal delays.

2) We focus on the *hose traffic variation model* [34] as an underlying assumption for all possible traffic demands within a datacenter network, similar to [20, 22]. This traffic model assumes that the knowledge of the maximum traffic entering and leaving the network at each network node is available. Such knowledge could be inferred using the physical limitations of the devices, such as the provided

bandwidth in Network Interface Cards (NICs) or switch chips. However, this assumption might be *overly pessimistic*, as the normal traffic demand entering and leaving the network through any node often falls within a certain range rather than consistently reaching maximum rates. In practice, some datacenter administrators may possess the knowledge of the traffic demand ranges [21, 35] (i.e., the traffic demand between a source-destination pair is restricted to a given demand range). Therefore, we also focus on the *customized hose model* which is a hose model that incorporates knowledge of traffic demand ranges (For formal definition, please see Section 4.4).

3) We assume that traffic can be *split arbitrarily* at any intermediate routing-capable network node (i.e., switch or routing-capable server), as proposed in [19, 36]. The traffic for each commodity is considered as an aggregation of flows[1] with the same commodity[2].

4) The practical implementation technique of traffic splitting, according to forwarding rules, on switches is not a focus of this thesis. However, the forwarding rules could be configured into programmable switches with traffic-splitting capability[3] by using these splitting techniques [27, 28, 29, 30].

5) We use the mathematical programming solver— i.e, Gurobi [37]— to solve all linear programs in the experiments.

6) The issue of exiting closed loops (i.e., directed cycles) in the routing solution is not a focus of this thesis. However, these loops could be removed during post-processing[4].

[1]Any packets are part of the same flow when their IP packet header fields are identical — a flow is typically defined by 5-tuple (source IP address, destination IP address, source port number, destination port number, transport protocol (e.g., TCP or UDP)).

[2]In practical deployment, a commodity could be defined by a pair of source node addresses and destination node addresses. For instance, the address for a Top-of-Rack switch might be assigned using the IP prefix of the servers connected to it.

[3]The switches that enable operators to control packet forwarding behavior, include commodity switches (e.g., OpenFlow switches) and P4 data plane programmable switches.

[4]For further details, please see Section 3.5 "Flow Decomposition Algorithms" in [38]

7) The issue of out-of-order packet delivery induced by multi-path routing is not a focus of this thesis. Nevertheless, this could be avoided by ensuring that packets belonging to the same TCP or UDP flow are routed along the same path.

2.3 Contributions

In this thesis, we propose a process for designing optimal compact oblivious routing for datacenter networks in polynomial time. The process consists of three steps, which are summarized as follows:

Achievement of Objective 1 (Optimal Oblivious Routing):

1) We transform a robust optimization problem for designing oblivious routing into a linear program. By solving this linear program, which is done in polynomial time, we obtain an optimal oblivious routing solution. However, this linear program is tractable for small-scale datacenter network topologies. (see Section 4.2)

2) We prove that a datacenter network topology with repeated structures has an optimal solution whose variables also have repetition. We use this insight to formulate a scalable linear program that is solvable in polynomial time and is tractable for larger datacenter network topologies. (see Section 4.3)

Achievement of Objective 2 (Compact Routing):

3) We utilize the structure of the optimal solution of the scalable linear program, to compact forwarding rules and reduce memory requirement for real-world deployment. (see Section 4.5)

Additionally, some datacenter administrators might have insights into traffic demands within a datacenter network, specifically, the range of traffic demand for each source-destination pair. These insights could come from historical measurements or capacity planning. We also extend the design process to accommodate such situations. (see Section 4.4)

Chapter 3

Literature Reviews

In this chapter, we provide a brief history of the development of oblivious routing studies which emphasizes on the objective of minimizing network congestion. Additionally, we review the compilation of works that related to the reduction of forwarding rules.

3.1 Oblivious Routing

Oblivious routing has long been studied in two perspectives: algorithmic construction with performance bounds [18, 33, 39, 40] and designing optimal oblivious routing via mathematical optimization [20, 21, 22, 41].

3.1.1 Algorithmic Construction with Performance Bounds

The performance of a routing algorithm is measured by the competitive ratio, defined as the maximum ratio of the network congestion resulting from an oblivious routing algorithm to the minimum possible network congestion across all traffic demands.

In the 1980s, Valiant *et al.* [18, 39] proposed an $O(\log n)$-competitive ratio oblivious routing algorithm for hypercube topology where n is the number of nodes in a network. The key idea of Valiant's routing scheme is that the traffic from a source is equally split to intermediate nodes, and every intermediate node routes the traffic to its destination. Alternatively, an oblivious routing scheme can be efficiently constructed from decomposition tree(s) via the hierarchical decomposition technique. In 2002, Räcke [33] proposed an oblivious routing scheme for undirected graphs with competitive ratio $O(\log^3 n)$ by using a single decomposition tree. Later in 2008, Räcke [40] utilized the convex combination of decomposition trees [42] to obtain an $O(\log n)$-competitive ratio routing scheme.

Nevertheless, the aforementioned methods emphasize the performance

bound in terms of competitive ratio rather than achieving optimal oblivious routing.

3.1.2 Designing Optimal Oblivious Routing via Mathematical Optimization

The studies on designing optimal oblivious routing via mathematical optimization can be categorized into two groups: optimal oblivious routing with polynomial time construction [20, 21, 41] and scalable design technique for optimal oblivious routing [22].

3.1.2.1 Optimal Oblivious Routing with Polynomial Time Construction

The framework for designing optimal oblivious routing in polynomial time typically unfolds in the following manner: Initially, obtaining optimal oblivious routing requires transforming the robust multi-commodity flow problem into a linear program [20, 21, 41]. Subsequently, this linear program can be solved in *polynomial time* (in the size of the problem instance, which is the network size) using these algorithms [24, 25, 26].

In 2003, Azar *et al.* [41] proposed the first polynomial-time approach for designing optimal oblivious routing. Initially, they formulated the optimal oblivious routing problem as a robust multi-commodity flow problem for an arbitrary network. Then, they transformed the robust multi-commodity flow problem into a main linear program with the objective of minimizing network congestion across all possible traffic matrices[1] within a network. They defined a traffic set as a set of all possible traffic matrices that can be routed (using any routing) with network congestion at most 1. However, due to the infinite (continuous) number of possible traffic matrices in the traffic set, they addressed this challenge by formulating a separation oracle linear program. This linear program is utilized to test the feasibility of the candidate routing solution for each link. In other words, given a candidate routing solution and a considered link, the separation oracle linear program aims to maximize the link's congestion ratio by

[1] A traffic matrix is formed by the traffic demands from every source-destination pair.

generating the worst-case traffic matrix that maximizes congestion ratio of the link. The candidate routing is feasible if every link's congestion ratio is less than the objective value specified in the main linear program. To ensure the construction of optimal oblivious routing in a polynomial time, they input the main linear program and the separation oracle into an iterative algorithm for solving the linear optimization problem—the Ellipsoid method [24]. This method is employed to find an optimal routing solution. However, in practice, the Ellipsoid method is known for its relatively slow and inefficient performance.

Motivated by the need for an optimal oblivious routing scheme for intra-domain networks, Applegate *et al.* [21] improved prior work by introducing an efficient polynomial-time technique for designing optimal oblivious routing for intra-domain networks. They formulated an efficient single linear program by combining the dual formulation of the separate oracle with the main linear program. The proposed linear program can be solved using a more efficient and practical optimization method called the Interior-point method [25]. However, the works [21, 41] made a broad assumption about possible traffic matrices within a network, those that can be routed with maximum congestion ratio at most 1. Later, Kodialam *et al.* [20] investigated the optimal oblivious routing problem for intra-domain networks, emphasizing a different assumption about possible traffic matrices. They employed the hose traffic variation model [34] which assumes that the knowledge of the maximum traffic entering and leaving the network at each network node is available. They adapted the previous work [21] to accommodate this traffic model. However, the intra-domain networks they studied[2], are smaller in term of scale compared to datacenter networks.

[2]Both are evaluated on coalesced versions of Rocketfuel topologies [43], specifically geographical PoP-to-PoP ISP networks, where nodes correspond to cities. These topologies have a maximum topology size of approximately 60 nodes and 90 links.

3.1.2.2 Scalable Technique for Designing Optimal Oblivious Routing

As datacenter network sizes are expanding rapidly to accommodate the increasing demand in the the era of AI and cloud computing, recent work proposed by Supittayapornpong *et al*. [22] in 2022 leveraged the repeated network structures, which typically exist in datacenter networks, to reduce the complexity of robust multi-commodity flow problem. This makes it tractable to obtain an optimal oblivious routing solution for larger network sizes. They utilized the hose traffic variation model as a traffic model and introduced an iterative algorithm that alternately solves two linear programs to solve the robust multi-commodity flow problem. To tackle the issue of considering an infinite number of possible traffic matrices in the traffic model, they first transformed the robust optimization problem into a linear program, which considers an initial traffic set with limited number of traffic matrices. They obtained a sub-optimal routing solution by solving this linear program. Subsequently, they generated additional traffic matrices by solving another linear program, using the obtained routing solution as a guide to improve the next routing solution. These generated traffic matrices were then added to the limited traffic set for solving the robust optimization problem again. This process continues until no infeasible traffic matrices were found[3]. While off-the-shelf solvers can efficiently solve a linear program in polynomial time, the challenge arises from the iterative nature of this method, where two linear programs are alternately solved. This is problematic because we have no notion when the algorithm would end.

All of the aforementioned methods either provide a polynomial-time construction bound but are intractable for large-scale topologies [20, 21, 41], or they are scalable for large networks but could take non-polynomial time construction due to the complexity of robust optimization [22]. This leads to a research question.

> **Research question 1:** Could we design optimal oblivious routing in polynomial time that also scales for large datacenter networks to achieve the best of both worlds?

[3] An infeasible traffic matrix is a traffic matrix that that violates the capacity of certain links.

3.2 Compact Oblivious Routing

Recent studies on technique to reduce the memory requirements can be categorized into two groups: 1) Trading-off split accuracy with memory requirements—allowing the rules to fit available memory, by approximating the routing solution and then truncating it to fewer rules [27, 28]. 2) Designing *compact* oblivious routing scheme, which is routing scheme that requires small number of forwarding rules [31, 32].

Most recent works about designing oblivious routing scheme that requires small memory have concentrated on designing algorithm for constructing compact oblivious routing with performance bounds [31, 32]. The performance of the algorithm is measured by the competitive ratio. In 2019, Räcke *et al.* [31] argued that the existing oblivious routing algorithm [33] requires large forwarding rules (i.e., polynomial in the network size). Therefore, they introduced the first *compact* oblivious routing algorithm that minimize congestion with a poly-logarithmic competitive ratio and provides a poly-logarithmic upper bound on size of forwarding rules (in the network size). Their algorithm is based on a hierarchical decomposition tree, as presented in [33], however, they also proposed the technique to reduce the size of forwarding rules. One important drawback of this work is that it only applies to undirected graphs with uniform capacity. Later in 2020, Czerner *et al.* [32] enhanced Räcke's result [31] by extending it to accommodate arbitrary undirected graphs.

These techniques to reduce the number of forwarding rules [31, 32] are customized specifically for use with oblivious routing design that based on Räcke's hierarchical decomposition tree [33]. However, the works that focus on designing optimal oblivious routing by solving optimization problems [20, 21, 22, 41] did not emphasize the reduction of forwarding rules. Meanwhile, the approximation approaches [27, 28] trade-off between optimality and available memory[4]. This leads to another research question:

> **Research question 2:** Could we reduce the memory requirement given an optimal oblivious routing solution?

[4]It is worth mentioning that the approximation approaches could be employed in situations where the reduced rules still exceed the available memory of a switch.

Chapter 4

Methodology

In this chapter, we describe the model of datacenter networks and the oblivious routing problem (Section 4.1). Then, we discuss the transformation of the problem into a linear program (Section 4.2), and the exploitation of the repeated network structures to improve scalability (Section 4.3). We also describe the modification to support customized traffic models (Section 4.4), and provide a grouping technique to compact forwarding rules for real-world deployment (Section 4.5).

4.1 System Model

In this section, we formally model datacenter networks, traffic demands, and routing components before formulating an oblivious routing optimization problem. These models are general and should fit most practical datacenter networks.

4.1.1 Datacenter Network Model

Datacenter network architecture can be divided into the traditional server-switch architecture [1, 2, 3, 9, 10, 44, 45] and the server-centric architecture [11, 46]. The differences between these architectures are that every server in the former has only one direct connection to a switch and has no routing capability, while every server in the latter has the routing capability and may have multiple direct connections to other servers or switches. Our network model capture both the traditional server-switch architecture and the server-centric architecture.

A datacenter network is an interconnection of network devices (e.g., servers or ethernet switches). Each device has a finite number of full-duplex ports for interconnection with other devices. For example, Broadcom's Tomahawk 4 Ethernet switch chip can be configured as 256 ports at 100Gbps [47] or a server which installs one Broadcom's P2100G Ethernet NIC with 2 ports at 100Gbps [48]. Two devices are

Table 4.1 List of key notations

Symbols	Description
Sets	
$\mathcal{S}$	Set of routing-capable nodes
$\mathcal{P}$	Set of routing-incapable nodes
$\mathcal{N}$	Set of all nodes, where $\mathcal{N} = \mathcal{S} \cup \mathcal{P}$
$\mathcal{L}$	Set of all links, where node i and node j are connected by link $(i, j) \in \mathcal{L}$ with link capacity C_{ij}
$\mathcal{H}$	Set of nodes that have positive capacities, where $\mathcal{H} = \{n \in \mathcal{N} : H_n > 0\}$
$\mathcal{C}$	Set of all commodities, where $\mathcal{C} = \{(u,v) \in \mathcal{H}^2 : u \neq v\}$
$\mathcal{T}$	Set of all possible traffic matrices
Parameters	
C_{ij}	The capacity of link (i, j)
H_n	The capacity of node n
$t^{uv}_{\min}$	The minimum traffic demand of commodity (u,v)
$t^{uv}_{\max}$	The maximum traffic demand of commodity (u,v)
Decision Variables	
f^{uv}_{ij}	The share of commodity (u,v) over link (i, j)
β^{u}_{ij}	The dual variable correspond to the node capacity constraint in (4.2.2) of node u over link (i, j)
γ^{u}_{ij}	The dual variable correspond to the node capacity constraint in (4.2.2) of node u over link (i, j)
λ^{uv}_{ij}	The dual variable correspond to the range constraint in (4.4.1) of commodity (u,v) over link (i, j)
μ^{uv}_{ij}	The dual variable correspond to the range constraint in (4.4.1) of commodity (u,v) over link (i, j)

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 aggregated capacity of all physical links between the devices.

In our model, a network device can be viewed as a node. To deal with the difference between two datacenter network architectures, we model a datacenter network by a graph with two types of nodes: routing-capable nodes and routing-incapable nodes. Let $\mathcal{S}$ and $\mathcal{P}$ be the sets of routing-capable nodes and routing-incapable nodes respectively. The set of all nodes is denoted by $\mathcal{N} = \mathcal{S} \cup \mathcal{P}$. Formally, a datacenter network is a directed graph with the set of all nodes $\mathcal{N}$ and the set of logical links $\mathcal{L}$, as shown in Figure 4.1. Every logical link connects two nodes. A directed link (i, j) connects node i to node j with capacity C_{ij}. 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 $(i, j) \in \mathcal{L}$.

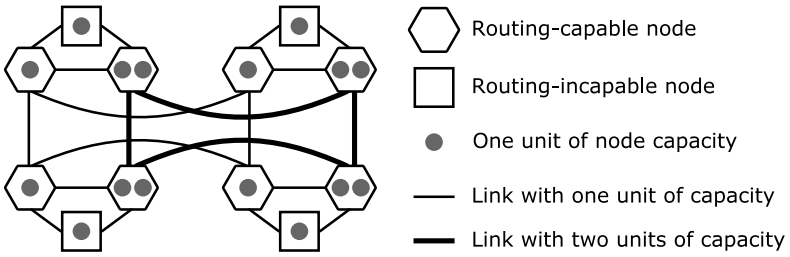

Figure 4.1 A simple network consists of routing-capable nodes, routing-incapable nodes, and links. These nodes and links could have different capacities.

4.1.2 Traffic Model

Traffic inside a datacenter network is a combination of demands generated by servers. In our datacenter network model, each network device (e.g., switch or server) is represented as a node. Each node n has a node capacity, denoted as H_n. Particularly, H_n represents the maximum amount of traffic that can enter and leave the network through that node. The value of H_n could be inferred using the provided bandwidth in Network Interface Cards (NICs) or switch chips. We denote the set of nodes that have positive capacities by $\mathcal{H} = \{n \in \mathcal{N} : H_n > 0\}$.

A commodity (u,v) is a pair of source node u and destination node v where both nodes have positive capacity. We denote the set of all commodities by $\mathcal{C} = \{(u,v) \in \mathcal{H}^2 : u \neq v\}$. Node u sends traffic to node v and creates a traffic demand for commodity (u,v). This traffic demand is denoted by t^{uv}. The traffic demands from every commodity form a traffic matrix $[t^{uv}] \in \mathbb{R}_+^{|\mathcal{C}|}$, where $\mathbb{R}_+$ is a set of non-negative reals.

Since traffic demands inside a datacenter network could be highly dynamic and difficult-to-predict in nature [8, 15, 16], we consider the set of all possible traffic matrices, which is also called the *hose traffic model* [20, 22, 49]:

$$\mathcal{T} = \left\{ \begin{array}{ll} \sum_{v \in \mathcal{H}_{-u}} t^{uv} \leq H_u & ,\forall u \in \mathcal{H} \\ \sum_{u \in \mathcal{H}_{-v}} t^{uv} \leq H_v & ,\forall v \in \mathcal{H} \\ t^{uv} \in \mathbb{R}_+ & ,\forall (u,v) \in \mathcal{C} \end{array} \right\}, \tag{4.1.1}$$

where $\mathcal{A}_{-x}$ contains all members of $\mathcal{A}$ but excludes x, i.e., $\mathcal{A}_{-x} = \mathcal{A} \backslash \{x\}$. The first constraint in (4.1.1) limits the amount of traffic generated from each node by the node's capacity. Similarly, the second constraint bounds the amount of traffic each node can receive by the node's capacity.

4.1.3 Routing Model

We adopt the multi-commodity flow model with the commodity set $\mathcal{C}$. For every commodity $(u,v) \in \mathcal{C}$, the share of commodity's demand over link (i,j) is denoted by f_{ij}^{uv} for every link $(i,j) \in \mathcal{L}$. Notice that, given a traffic demand t^{uv}, the amount of the demand over link (i,j) is $t^{uv} f_{ij}^{uv}$. These shares over all links in the network, $\left\{ f_{ij}^{uv} : (i,j) \in \mathcal{L} \right\}$, form the shares of commodity (u,v). We denote the shares of all commodities by $[f_{ij}^{uv}]$, where $[f_{ij}^{uv}] \in \mathbb{R}_{+}^{|\mathcal{C}| \times |\mathcal{L}|}$.

Recall that there are two node types. A *routing-capable* node can route every traffic independent of the traffic's commodity. A *routing-incapable* node can only route traffic corresponding to the commodities involving the node, i.e., the node is either a source or a destination of the traffic. Define $\mathbb{I}[x]$ as an indicator function that returns 1 if the statement x is true; otherwise 0. We define the set of all possible shares as follows:

$$\mathcal{F} = \left\{ \begin{array}{ll} \sum\limits_{j \in \mathcal{O}(i)} f_{ij}^{uv} - \sum\limits_{j \in \mathcal{I}(i)} f_{ji}^{uv} = \mathbb{I}[i=u] - \mathbb{I}[i=v] & \\ \qquad\qquad , \forall (u,v) \in \mathcal{C}, \forall i \in \mathcal{N} & \\ f_{ij}^{uv} = 0 \qquad , \forall (u,v) \in \mathcal{C}, \forall (i,j) \in \mathcal{N} \times \mathcal{P}_{-v} & \\ f_{iu}^{uv} = f_{vi}^{uv} = 0, \forall (u,v) \in \mathcal{C}, \forall i \in \mathcal{N} & \\ f_{ij}^{uv} \in \mathbb{R}_{+} \qquad , \forall (u,v) \in \mathcal{C}, \forall (i,j) \in \mathcal{L} & \end{array} \right\}, \tag{4.1.2}$$

where $\mathcal{O}(i)$ and $\mathcal{I}(i)$ are respectively the set of nodes corresponding to the outgoing links from node i and the set of nodes corresponding to the incoming links to node i. The first constraint is the share conservation where the total share at every source and every destination is one. The second constraint enforces that every routing-incapable node is not involved in the routes of other commodities unrelated to the node. The third constraint ensures that both the source and destination are the endpoints of their corresponding traffic.

4.1.4 Oblivious Routing Formulation

Given shares $[f_{ij}^{uv}]$ and a traffic matrix $[t^{uv}]$, we define the *congestion ratio* of link (x,y) by the proportion of total traffic demand on the link to the link's capacity [21, 41]:

$$\sum_{(u,v) \in \mathcal{C}} \frac{t^{uv} f_{xy}^{uv}}{C_{xy}}.$$

An oblivious routing problem aims to design the shares $[f_{ij}^{uv}]$ that *minimizes* this congestion ratio across every link in the network and under all possible traffic demands[1]. We formulate this oblivious routing problem as follows[2]. :

$$\min_{[f_{ij}^{uv}]\in\mathcal{F}} \max_{\substack{(x,y)\in\mathcal{L},\\ [t^{uv}]\in\mathcal{T}}} \sum_{(u,v)\in\mathcal{C}} \frac{t^{uv} f_{xy}^{uv}}{C_{xy}}. \tag{4.1.3}$$

This formulation has two challenges: (1) the robust objective with infinite possibilities of traffic demands and (2) the scalability of the formulation for large networks. The first challenge stems from the fact that the congestion ratio is a function of every traffic demand in the traffic set $\mathcal{T}$, so the entire set must be considered. The second challenge is due to the sizes of datacenter networks, resulting in large numbers of decision variables and constraints in the formulation. We tackle the two challenges successively in Section 4.2 and Section 4.3.

4.2 Linear Program Formulation

This section tackles the challenge of the robust objective by transforming the robust formulation in (4.1.3) to a simpler linear program, so an optimal oblivious routing solution can be designed for appropriate network sizes[3]. This transformation is inspired by [21], which considers a different network model and assumes possible traffic matrices in the traffic set differently and uses different proof techniques.

We first observe that each link has its worst-cast traffic matrix, yielding its maximal congestion ratio. Therefore, we introduce traffic matrix $[t_{xy}^{uv}] \in \mathcal{T}$ for every link (x,y) to decouple the inner maximization in (4.1.3) as follows:

$$\max_{\substack{(x,y)\in\mathcal{L},\\ [t^{uv}]\in\mathcal{T}}} \sum_{(u,v)\in\mathcal{C}} \frac{t^{uv} f_{xy}^{uv}}{C_{xy}} = \max_{(x,y)\in\mathcal{L}} \max_{[t_{xy}^{uv}]\in\mathcal{T}} \sum_{(u,v)\in\mathcal{C}} \frac{t_{xy}^{uv} f_{xy}^{uv}}{C_{xy}} \tag{4.2.1}$$

[1]This is equivalent to the performance ratio in [21, 41] when the minimum possible maximum link utilization is ignored.

[2]While the formulation may introduce cycles in a routing solution, those cycles can be easily removed in post-processing (For further details, please see Section 3.5 "Flow Decomposition Algorithms" in [38])

[3]While, *inappropriate* network sizes refer to those that cause the problem size of the linear program—in terms of the number of variables and constraints—to exceed the computation resources (e.g., the time for solving or the memory of optimization solvers), making the task of obtaining the optimal oblivious routing solution intractable.

The above decomposition allows us to consider the sub-problem, $\max_{[t_{xy}^{uv}]\in\mathcal{T}} \sum_{(u,v)\in\mathcal{C}} \frac{t_{xy}^{uv} f_{xy}^{uv}}{C_{xy}}$, which finds the worst-case congestion ratio of link (x,y), in isolation. Given a considered link (x,y) and a share solution $[f_{ij}^{uv}]$, this sub-problem is rewritten as the following formulation:

$$\begin{aligned}
\text{Maximize} \quad & \sum_{(u,v)\in\mathcal{C}} \frac{t_{xy}^{uv} f_{xy}^{uv}}{C_{xy}} \\
\text{Subject to} \quad & \sum_{v\in\mathcal{H}_{-u}} t_{xy}^{uv} \leq H_u \quad ,\forall u \in \mathcal{H} \\
& \sum_{u\in\mathcal{H}_{-v}} t_{xy}^{uv} \leq H_v \quad ,\forall v \in \mathcal{H} \\
& t_{xy}^{uv} \in \mathbb{R}_+ \qquad\quad ,\forall (u,v) \in \mathcal{C}.
\end{aligned} \tag{4.2.2}$$

Instead of solving this sub-problem directly, we consider its dual problem with dual variables $[\beta_{xy}^u]_{u\in\mathcal{H}}$ and $[\gamma_{xy}^u]_{u\in\mathcal{H}}$:

$$\begin{aligned}
\text{Minimize} \quad & \sum_{u\in\mathcal{H}} H_u \left(\beta_{xy}^u + \gamma_{xy}^u\right) \\
\text{Subject to} \quad & \frac{f_{xy}^{uv}}{C_{xy}} - \beta_{xy}^u - \gamma_{xy}^v \leq 0 \quad ,\forall (u,v) \in \mathcal{C} \\
& \beta_{xy}^u \in \mathbb{R}_+,\ \gamma_{xy}^u \in \mathbb{R}_+ \quad ,\forall u \in \mathcal{H}.
\end{aligned} \tag{4.2.3}$$

Lemma 1. *For a given link (x,y) and a share solution $[f_{ij}^{uv}]$, the formulation in* (4.2.3) *is the dual of the problem in* (4.2.2).

Proof. Fix a link (x,y) and a share solution $[f_{ij}^{uv}]$. We derive a dual problem by the duality technique [50]. Define non-negative dual variables $\left[\beta_{xy}^u\right]_{u\in\mathcal{H}}, \left[\gamma_{xy}^u\right]_{u\in\mathcal{H}}$ respectively for the first two inequality constraints of the sub-problem in (4.2.2). We define the Lagrangian associated with this problem as

$$\begin{aligned}
L\left(\left[t_{xy}^{uv}\right], \left[\beta_{xy}^u\right], \left[\gamma_{xy}^v\right]\right) = & \sum_{(u,v)\in\mathcal{C}} \frac{t_{xy}^{uv} f_{xy}^{uv}}{C_{xy}} \\
& - \sum_{u\in\mathcal{H}} \beta_{xy}^u \left(\sum_{v\in\mathcal{H}_{-u}} t_{xy}^{uv} - H_u \right) - \sum_{v\in\mathcal{H}} \gamma_{xy}^v \left(\sum_{u\in\mathcal{H}_{-v}} t_{xy}^{uv} - H_v \right) \\
= & \sum_{(u,v)\in\mathcal{C}} t_{xy}^{uv} \left(\frac{f_{xy}^{uv}}{C_{xy}} - \beta_{xy}^u - \gamma_{xy}^v \right) + \sum_{u\in\mathcal{H}} H_u \left(\beta_{xy}^u + \gamma_{xy}^u\right).
\end{aligned}$$

The dual function is defined as

$$D\left(\left[\beta_{xy}^{u}\right],\left[\gamma_{xy}^{v}\right]\right)=\max_{\left[t_{xy}^{uv}\right]\in\mathbb{R}_{+}^{|\mathcal{C}|}} L\left(\left[t_{xy}^{uv}\right],\left[\beta_{xy}^{u}\right],\left[\gamma_{xy}^{v}\right]\right).$$

Note that this dual function is undefined, $D\left(\left[\beta_{xy}^{u}\right],\left[\gamma_{xy}^{v}\right]\right)\rightarrow\infty$ when $\frac{f_{xy}^{uv}}{C_{xy}}-\beta_{xy}^{u}-\gamma_{xy}^{v}>0$ for some commodity $(u,v)\in\mathcal{C}$. We only consider the well-defined dual function $D\left(\left[\beta_{xy}^{u}\right],\left[\gamma_{xy}^{v}\right]\right)=\sum_{u\in\mathcal{H}}H_u\left(\beta_{xy}^{u}+\gamma_{xy}^{u}\right)$ when $\frac{f_{xy}^{uv}}{C_{xy}}-\beta_{xy}^{u}-\gamma_{xy}^{v}\leq 0$ for every commodity $(u,v)\in\mathcal{C}$. This is because a dual problem minimizes this dual function, i.e., $\min_{\left[\beta_{xy}^{u}\right]\in\mathbb{R}_{+}^{|\mathcal{H}|},\left[\gamma_{xy}^{v}\right]\in\mathbb{R}_{+}^{|\mathcal{H}|}} D\left(\left[\beta_{xy}^{u}\right],\left[\gamma_{xy}^{v}\right]\right)$, resulting in the dual problem in (4.2.3), which proves the lemma.

□

We use the dual problem in (4.2.3) to transform the robust optimization problem in (4.1.3) into the following linear program:

$$\begin{aligned}
\text{Minimize}\quad & \eta \\
\text{Subject to}\quad & \sum_{u\in\mathcal{H}} H_u\left(\beta_{ij}^{u}+\gamma_{ij}^{u}\right)\leq\eta\ ,\forall(i,j)\in\mathcal{L} \\
& \frac{f_{ij}^{uv}}{C_{ij}}-\beta_{ij}^{u}-\gamma_{ij}^{v}\leq 0\ ,\forall(u,v)\in\mathcal{C},\forall(i,j)\in\mathcal{L} \\
& \beta_{ij}^{u}\in\mathbb{R}_{+},\ \gamma_{ij}^{u}\in\mathbb{R}_{+}\ ,\forall u\in\mathcal{H},\forall(i,j)\in\mathcal{L} \\
& [f_{ij}^{uv}]\in\mathcal{F}.
\end{aligned} \tag{4.2.4}$$

Theorem 1. *The robust oblivious routing formulation in* (4.1.3) *and the linear program in* (4.2.4) *are equivalent.*

Proof. The proof replaces the sub-problem in (4.2.2) with the dual problem in (4.2.3) on the right-hand side of equation (4.2.1). Let $g_{xy}([f_{ij}^{uv}])$ represent the dual problem in (4.2.3) of link (x,y) given routing variables $[f_{ij}^{uv}]$. The replacement yields:

$$\max_{\substack{(i,j)\in\mathcal{L},\\ [t^{uv}]\in\mathcal{T}}}\sum_{(u,v)\in\mathcal{C}}\frac{t^{uv}f_{ij}^{uv}}{C_{ij}}=\max_{(x,y)\in\mathcal{L}} g_{xy}([f_{ij}^{uv}]).$$

Since the left-hand side of the above equation is the inner maximization of the oblivious routing formulation in (4.1.3), it follows that the formulation is equivalent to

$$\min_{[f_{ij}^{uv}]\in\mathcal{F}}\max_{(x,y)\in\mathcal{L}} g_{xy}([f_{ij}^{uv}]).$$

The above min-max problem can be transformed into a linear program with an auxiliary variable η as follows:

$$\begin{aligned} \text{Minimize} \quad & \eta \\ \text{Subject to} \quad & g_{xy}([f_{ij}^{uv}]) \leq \eta \quad ,\forall (x,y) \in \mathcal{L} \\ & [f_{ij}^{uv}] \in \mathcal{F}. \end{aligned}$$

Finally, replacing the dual problem $g_{xy}([f_{ij}^{uv}])$ with its objective function and constraints in (4.2.3) gives the linear program in (4.2.4) and proves the theorem. □

The implication of Theorem 1 is that an optimal oblivious routing solution of the robust formulation in (4.1.3) can be obtained by solving the linear program in (4.2.4). Since linear programming is solvable in polynomial time [25, 26], we can design optimal oblivious routing in polynomial time for appropriate network sizes, in terms of variables and constraints. However, the scales of datacenter networks could render the linear program in (4.2.4) intractable in practice. The next section tackles this scalability issue.

4.3 Scalable Linear Program Formulation

This section tackles the scalability of the linear formulation in (4.2.4) for large-scale datacenter networks. We exploit the repeated structures in a network topology to scale the formulation. This idea is inspired by the work in [22], which is non-polynomial time and has no routing-incapable nodes.

4.3.1 Automorphism in Datacenter Networks

The repeated structures in a datacenter network are captured by graph automorphism. Graph automorphism is a mapping of nodes in a graph to the same set of nodes in the same graph such that the new graph with the mapped nodes is similar to the original graph [51]. In particular, the described automorphism preserves node adjacency. We extend this automorphism to our network model by the definition of our automorphism.

Definition 1. *A mapping function $\phi : \mathcal{N} \rightarrow \mathcal{N}$ is an automorphism if the following conditions hold:*

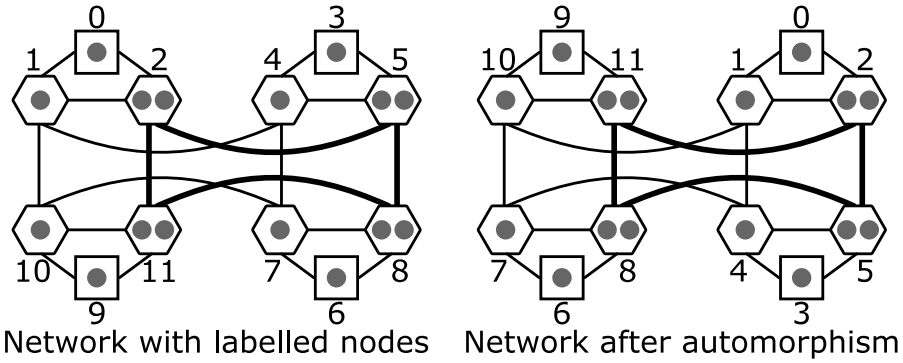

Figure 4.2 Automorphism example from the simple network in Figure 4.1. Nodes in the right network are nodes in the left network relabeled by the automorphism $\phi(n) = n - 3 \mod 12$. Both networks have the same adjacency, link capacity distribution, node types, and node capacity distribution.

1. *Adjacency:* $(\phi(i), \phi(j)) \in \mathcal{L}$, $\forall (i,j) \in \mathcal{L}$.
2. *Link capacity:* $C_{\phi(i)\phi(j)} = C_{ij}$, $\forall (i,j) \in \mathcal{L}$.
3. *Type:* $\phi(n) \in \mathcal{S}$, $\forall n \in \mathcal{S}$ *and* $\phi(n) \in \mathcal{P}$, $\forall n \in \mathcal{P}$.
4. *Node capacity:* $H_{\phi(n)} = H_n$, $\forall n \in \mathcal{N}$.

The automorphism in Definition 1 is illustrated in Figure 4.2. The first condition ensures that the adjacency between nodes is preserved under the automorphism. The second condition keeps the capacity of every link the same after the automorphism mapping. The type and capacity of every node also stay the same under the automorphism from the last two conditions.

Let Φ be the set of all automorphisms satisfying Definition 1. Since the numbers of nodes and links in a network are finite, there are finite permutations, and the cardinality of the set is finite, $|\Phi| < \infty$. In addition, we define $\hat{\Phi}$ as the set of generators that entirely generate the automorphism set Φ. This generator set can be obtained from off-the-shelf software, such as `nauty` [52].

To obtain the generator set $\hat{\Phi}$, we encode the four properties in Definition 1 on a network topology as an undirected graph with colored vertices, the input format for `nauty` [52]. Algorithm 1 summarizes the encoding process, and Figure 4.3 shows the graph constructed by the algorithm from the network in Figure 4.1. In the

Algorithm 1: Graph construction for generator set $\hat{\Phi}$

Initialize a graph with a vertex set $\mathcal{V}$ and an edge set $\mathcal{E}$

Initialize dictionaries Γ, W, W', M, M' with values as colors

Initialize variables D, D' with value as color

for $n \in \mathcal{N}$ **do**
- $\mathcal{V} \leftarrow \mathcal{V} \cup \{n\}$
- **if** $(\mathbb{I}[n \in \mathcal{S}], H_n) \notin W$ **then**
  - $W[(\mathbb{I}[n \in \mathcal{S}], H_n)] \leftarrow$ new color
- $\Gamma[n] \leftarrow W[(\mathbb{I}[n \in \mathcal{S}], H_n)]$

for $(i, j) \in \mathcal{L}$ **do**
- Let n be a new vertex
- $\mathcal{V} \leftarrow \mathcal{V} \cup \{n\}$
- $\mathcal{E} \leftarrow \mathcal{E} \cup \{(i,n), (n, j)\}$
- **if** $C_{ij} \notin W'$ **then**
  - $W'[C_{ij}] \leftarrow$ new color
- $\Gamma[n] \leftarrow W'[C_{ij}]$

return a graph with sets $\mathcal{V}, \mathcal{E}$ and dictionary Γ

algorithm, vertices associated with nodes having the same type and node capacity are assigned the same color. Any two vertices associated with nodes having different types or different capacities are given different colors. This construction ensures the preservation of node type and node capacity in the generator set. For links in the network, each link yields a new vertex with two edges, each connecting to the vertex associated with each side of the link. Vertices associated with links having the same capacity are assigned the same color. Any two vertices associated with links having different capacities are given different colors. This construction ensures the properties on adjacency and link capacity are enforced. The time complexity of Algorithm 1 is $O(|\mathcal{N}| + |\mathcal{L}|)$. Finally, the constructed graph is input to `nauty` [52] to produce the generator set.

Next, we use the automorphism set Φ and its generator set $\hat{\Phi}$ to identify a special optimal oblivious routing solution.

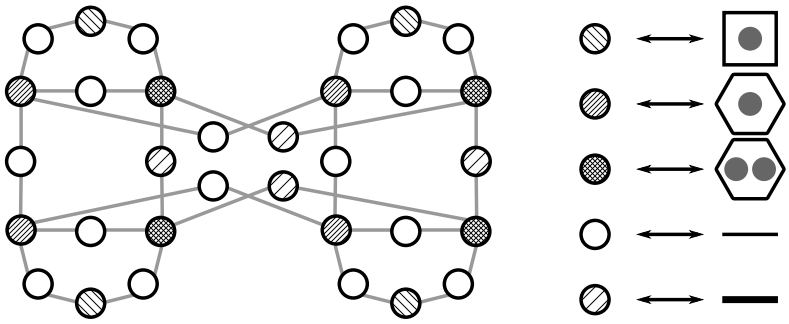

Figure 4.3 A graph with colored vertices is constructed by Algorithm 1 from the simple network in Figure 4.1.

4.3.2 Automorphism-invariant Optimal Solution

In this section, we show that the repeated structures in a datacenter network lead to an optimal oblivious routing solution that also has repeated structures.

We first argue that the linear program in (4.2.4) has an optimal solution. We know that the robust optimization in (4.1.3) aims to minimize the congestion ratio over the traffic set, which is assumed to be nonempty, and the sum of all link capacities in the network is finite. Therefore, the minimal congestion ratio exists. Since the linear program in (4.2.4) is equivalent to the robust optimization in (4.1.3) by Theorem 1, it must have an optimal solution. Let $\left(\eta^*, [\beta_{ij}^{u*}], [\gamma_{ij}^{u*}], [f_{ij}^{uv*}]\right)$ be an optimal solution of the linear program in (4.2.4). We show that another optimal solution can be constructed from an automorphism.

Lemma 2. *For any $\phi \in \Phi$, a solution $\left(\eta, [\beta_{ij}^{u}], [\gamma_{ij}^{u}], [f_{ij}^{uv}]\right)$ is an optimal solution of the linear program in* (4.2.4) *when*

$$\eta = \eta^*,\ \ \beta_{ij}^{u} = \beta_{\phi(i)\phi(j)}^{\phi(u)*},\ \ \gamma_{ij}^{u} = \gamma_{\phi(i)\phi(j)}^{\phi(u)*},\ \ f_{ij}^{uv} = f_{\phi(i)\phi(j)}^{\phi(u)\phi(v)*}.$$

Proof. Fix an automorphism $\phi \in \Phi$. Since $\eta = \eta^*$, the objective cost under the new solution equals the optimal objective cost. We then need to show that the solution, constructed from the automorphism, is feasible and satisfies all constraints in (4.2.4) and (4.1.2). We begin with the domains of variables β_{ij}^{u} and γ_{ij}^{u} in (4.2.4) and variables f_{ij}^{uv} in (4.1.2). These domains are non-negative reals, so the variables of the constructed solution are in the same domains. Next, we consider the constraints in (4.2.4).

Considering the first constraint in (4.2.4) with link (i,j), we have that

$$\sum_{u\in\mathcal{H}} H_u\left(\beta_{ij}^u+\gamma_{ij}^u\right)=\sum_{u\in\mathcal{H}} H_{\phi(u)}\left(\beta_{\phi(i)\phi(j)}^{\phi(u)*}+\gamma_{\phi(i)\phi(j)}^{\phi(u)*}\right)$$
$$=\sum_{u\in\mathcal{H}} H_u\left(\beta_{\phi(i)\phi(j)}^{u*}+\gamma_{\phi(i)\phi(j)}^{u*}\right)\leq\eta^*=\eta.$$

The first equality substitutes the solution and uses the node capacity property in Definition 1. Reindexing $u\in\mathcal{H}$ leads to the second equality. The last inequality holds because the constraint associated with link $(\phi(i),\phi(j))$ holds under the optimal solution. Thus, the first constraint in (4.2.4) is satisfied.

Considering the second constraint in (4.2.4) with commodity (u,v) and link (i,j), we have that

$$\frac{f_{ij}^{uv}}{C_{ij}}-\beta_{ij}^u-\gamma_{ij}^v=\frac{f_{\phi(i)\phi(j)}^{\phi(u)\phi(v)*}}{C_{\phi(i)\phi(j)}}-\beta_{\phi(i)\phi(j)}^{\phi(u)*}-\gamma_{\phi(i)\phi(j)}^{\phi(v)*}\leq 0.$$

The first equality substitutes the solution and uses the link capacity property in Definition 1. The last inequality holds because the constraint associated with commodity $(\phi(u),\phi(v))$ and link $(\phi(i),\phi(j))$ holds under the optimal solution. Thus, the second constraint in (4.2.4) is satisfied.

We then consider the constraints in (4.1.2). Considering the second constraint with commodity $(u,v)\in\mathcal{C}$ and link $(i,j)\in\mathcal{N}\times\mathcal{P}_{-v}$, we have that $f_{ij}^{uv}=f_{\phi(i)\phi(j)}^{\phi(u)\phi(v)*}$. Since node j is routing-incapable and $j\neq v$, node $\phi(j)$ is also routing-incapable and $\phi(j)\neq\phi(v)$ from Definition 1. It follows that $f_{\phi(i)\phi(j)}^{\phi(u)\phi(v)*}=0$ because the optimal share of commodity $(\phi(u),\phi(v))$ never routes traffic over link $(\phi(i),\phi(j))$ as $\phi(j)\in\mathcal{P}_{-\phi(v)}$. Therefore, we have $f_{ij}^{uv}=0$, and the second constraint in (4.1.2) is satisfied.

Considering the third constraint in (4.1.2) with commodity $(u,v)\in\mathcal{C}$ and node $i\in\mathcal{N}$, we have that $f_{iu}^{uv}=f_{\phi(i)\phi(u)}^{\phi(u)\phi(v)*}=0$ because the optimal share of commodity $(\phi(u),\phi(v))$ never routes traffic back to the traffic's source node. A similar argument can be applied to the constraint $f_{vi}^{uv}=0$ as the optimal share of commodity $(\phi(u),\phi(v))$ never routes traffic away from the traffic's destination node. Therefore, The third constraint in (4.1.2) is satisfied. For the first constraint in (4.1.2), the proof that the constraint is satisfied is similar to the proof in [22] and is omitted for brevity. Thus, it holds that $[f_{ij}^{uv}]\in\mathcal{F}$.

Altogether, the solution achieves the same optimal objective cost and is feasible. It must be an optimal solution. □

The implication of Lemma 2 is that we can construct multiple optimal solutions from an optimal solution and the automorphism set. Next, we use these optimal solutions to identify an optimal solution with repetitive variables.

Theorem 2. *There exists an automorphism-invariant solution* $\left(\hat{\eta}, [\hat{\beta}^u_{ij}], [\hat{\gamma}^u_{ij}], [\hat{f}^{uv}_{ij}]\right)$ *that is optimal for the linear program in* (4.2.4) *and satisfies:*

$$\hat{f}^{uv}_{ij} = \hat{f}^{\phi(u)\phi(v)}_{\phi(i)\phi(j)}, \;\; \hat{\beta}^u_{ij} = \hat{\beta}^{\phi(u)}_{\phi(i)\phi(j)}, \;\; \hat{\gamma}^u_{ij} = \hat{\gamma}^{\phi(u)}_{\phi(i)\phi(j)}, \;\; \forall \phi \in \Phi.$$

Proof. We first construct a solution before showing that it is optimal and satisfies the automorphism-invariant property. Let $\left(\hat{\eta}, [\hat{\beta}^u_{ij}], [\hat{\gamma}^u_{ij}], [\hat{f}^{uv}_{ij}]\right)$ be a solution such that

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

$$\hat{\beta}^u_{ij} = \frac{1}{|\Phi|} \sum_{\phi \in \Phi} \beta^{\phi(u)*}_{\phi(i)\phi(j)}, \qquad \hat{\gamma}^u_{ij} = \frac{1}{|\Phi|} \sum_{\phi \in \Phi} \gamma^{\phi(u)*}_{\phi(i)\phi(j)},$$

where $\left(\eta^*, [\beta^{u*}_{ij}], [\gamma^{u*}_{ij}], [f^{uv*}_{ij}]\right)$ is an existing optimal solution.

Since $\hat{\eta} = \eta^*$, the objective cost under the solution equals to the optimal cost. Next, we show that the solution is feasible.

The linearity in the construction of the solution implies that i) the domains of variables β^u_{ij} and γ^u_{ij} in (4.2.4), ii) the domains of variables f^{uv}_{ij} in (4.1.2), and iii) the second and third constraints in (4.1.2) are satisfied. The proof that the first constraint in (4.1.2) is satisfied is similar to the proof in [22] and is omitted for brevity. We next consider the other constraints in (4.2.4).

The first constraint in (4.2.4) with link (i, j) gives

$$\sum_{u \in \mathcal{H}} H_u \left[\frac{1}{|\Phi|} \sum_{\phi \in \Phi} \beta^{\phi(u)*}_{\phi(i)\phi(j)} + \frac{1}{|\Phi|} \sum_{\phi \in \Phi} \gamma^{\phi(u)*}_{\phi(i)\phi(j)} \right]$$
$$= \frac{1}{|\Phi|} \sum_{\phi \in \Phi} \sum_{u \in \mathcal{H}} H_{\phi(u)} \left(\beta^{\phi(u)*}_{\phi(i)\phi(j)} + \gamma^{\phi(u)*}_{\phi(i)\phi(j)} \right) \leq \eta^*.$$

The first equality uses the node capacity property in Definition 1. The last inequality uses that the inner summation is at most η^* since the optimal solution satisfies the constraint with link $(\phi(i), \phi(j))$. Therefore, the constraint is satisfied.

The second constraint in (4.2.4) with commodity (u,v) and link (i,j) gives

$$\frac{1}{C_{ij}|\Phi|}\sum_{\phi\in\Phi} f^{\phi(u)\phi(v)*}_{\phi(i)\phi(j)} - \frac{1}{|\Phi|}\sum_{\phi\in\Phi}\beta^{\phi(u)*}_{\phi(i)\phi(j)} - \frac{1}{|\Phi|}\sum_{\phi\in\Phi}\gamma^{\phi(v)*}_{\phi(i)\phi(j)}$$
$$= \frac{1}{|\Phi|}\sum_{\phi\in\Phi}\left[\frac{f^{\phi(u)\phi(v)*}_{\phi(i)\phi(j)}}{C_{\phi(i)\phi(j)}} - \beta^{\phi(u)*}_{\phi(i)\phi(j)} - \gamma^{\phi(v)*}_{\phi(i)\phi(j)}\right] \leq 0.$$

The first equality uses the link capacity property in Definition 1. The last inequality uses the fact that the expression in the summation is non-positive since the optimal solution satisfies the constraint with commodity $(\phi(u),\phi(v))$ and link $(\phi(i),\phi(j))$. Therefore, the constraint is satisfied. Altogether, the solution is feasible and optimal.

Finally, we show the solution satisfies the automorphism-invariant property. We first consider $\hat{\beta}^u_{ij}$ and any $\phi\in\Phi$:

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

The second equality used the fact that the set of all automorphism mapping Φ is a group under the operation of function composition[4] and the summation is over the entire set. Therefore, every variable $\hat{\beta}^u_{ij}$ is automorphism-invariant. Similar arguments prove the automorphism-invariant property of variables $\hat{f}^{uv}_{ij}$ and $\hat{\gamma}^u_{ij}$ and are omitted. Thus, the solution is automorphism-invariant. This proves the theorem. □

The implication of Theorem 2 is that the linear program in (4.2.4) always has an optimal solution whose variables form groups. Every variable in each group takes the same value, i.e., $\hat{\beta}^u_{ij} = \beta^{\phi(u)}_{\phi(i)\phi(j)}$ for all $\phi\in\Phi$. We use this insight to formulate a new linear program targeting the automorphism-invariant optimal solution to improve scalability. This linear program has a significantly reduced problem size, allowing it to scale to larger network sizes.

[4]The composition of two automorphism mapping functions gives another automorphism mapping in the same set Φ.

4.3.3 Representative Variables

We first identify how variables $[\beta_{ij}^u], [\gamma_{ij}^u], [f_{ij}^{uv}]$ of the linear program in (4.2.4) form groups, so the variables in each group can be represented by a representative variable. The challenge of this identification is its combinatorial nature.

We observe from Theorem 2 that the variables $[\beta_{ij}^u]$ and $[\gamma_{ij}^u]$ share the same group relations as they share the same index set, $\mathcal{H} \times \mathcal{L}$. We therefore develop an efficient algorithm to group the variables based on the generator set $\hat{\Phi}$ as summarized in Algorithm 2. The algorithm searches over the index set. It takes an index from the set and utilizes the generator set to find all indices sharing the same group. Once a group is formed, the algorithm takes an unvisited index and continues the search process until all groups are formed. The time complexity of Algorithm 2 is $O(|\mathcal{H}|\,|\mathcal{L}|\,|\hat{\Phi}|)$, since every index is visited once and exactly $|\hat{\Phi}|$ indices are searched over per visited index. Notice that using the generator set is more efficient than the entire automorphism set, which is exponentially large.

The Algorithm 2 outputs the representative index set $\hat{\mathcal{G}}$ and the dictionary ω containing automorphisms that can map each variable to its representative. In particular, let $\hat{\beta}_{ij}^u$ and $\hat{\gamma}_{ij}^u$ be representative variables for every $(u,i,j) \in \hat{\mathcal{G}}$. They represent variables $[\beta_{ij}^u]$ and $[\gamma_{ij}^u]$ of the linear program in (4.2.4) as follows:

$$\begin{aligned} \beta_{ij}^u &\xrightarrow{\text{represented by}} \varphi\left[\beta_{ij}^u\right] = \hat{\beta}_{\phi(i)\phi(j)}^{\phi(u)} \quad \text{where } \phi = \omega[u,i,j] \\ \gamma_{ij}^u &\xrightarrow{\text{represented by}} \varphi\left[\gamma_{ij}^u\right] = \hat{\gamma}_{\phi(i)\phi(j)}^{\phi(u)} \quad \text{where } \phi = \omega[u,i,j] \end{aligned} \tag{4.3.1}$$

for every $(u,i,j) \in \mathcal{H} \times \mathcal{L}$. We use $\varphi[x]$ to denote the representative of x.

For the share variables $[f_{ij}^{uv}]$, their groups can be obtained from Algorithms 2 and 3 in [22], whose multi-commodity formulation includes a related conservation constraint similar to ours in (4.1.2). Algorithm 2 in [22] outputs the set of representative commodities, $\hat{\mathcal{C}}$, and a dictionary π containing automorphisms that can map each commodity to its representative commodity. For each representative commodity $(u,v) \in \hat{\mathcal{C}}$, Algorithm 3 in [22] outputs the set of representative links of share variables, $\hat{\mathcal{L}}^{uv}$, and a dictionary σ^{uv} containing automorphisms that can map each share variable to its representative. In particular, let $\hat{f}_{ij}^{uv}$ be a representative share variable for every $(u,v) \in \hat{\mathcal{C}}$ and every $(i,j) \in \hat{\mathcal{L}}^{uv}$. They represent shares $[f_{ij}^{uv}]$ of the linear program in

Algorithm 2: Identification of $[\hat{\beta}_{ij}^{u}]$ and $[\hat{\gamma}_{ij}^{u}]$

Initialize empty sets $\mathcal{Q}, \mathcal{Z}, \hat{\mathcal{G}}$

Initialize dictionaries D, ω

for $(u,i,j) \in \mathcal{H} \times \mathcal{L}$ **do**

  if $(u,i,j) \in \mathcal{Z}$ **then**

    continue

  $\hat{\mathcal{G}} \leftarrow \hat{\mathcal{G}} \cup \{(u,i,j)\}$

  $D[u,i,j] \leftarrow \phi_{\text{identity}}$

  $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(u,i,j)\}$

  while $\mathcal{Q}$ *is not empty* **do**

    Pop (a,b,c) from $\mathcal{Q}$

    $\mathcal{Z} \leftarrow \mathcal{Z} \cup \{(a,b,c)\}$

    for $\phi \in \hat{\Phi}$ **do**

      if $(\phi(a), \phi(b), \phi(c)) \notin \mathcal{Z}$ **then**

        $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(\phi(a), \phi(b), \phi(c))\}$

        $D[\phi(a), \phi(b), \phi(c)] \leftarrow \phi(D[a,b,c])$

for $(u,i,j) \in \mathcal{H} \times \mathcal{L}$ **do**

  $\omega[u,i,j] \leftarrow (D[u,i,j])^{-1}$

return representative index set $\hat{\mathcal{G}}$ and dictionary ω

Algorithm 3: Construction of index set $\hat{\mathcal{M}}$

Initialize empty sets $\mathcal{Z}, \hat{\mathcal{M}}$

for $(i,j) \in \mathcal{L}$ **do**

  Let $e = (a_{uij}, b_{uij})_{(u,i,j) \in \hat{\mathcal{G}}}$ where a_{uij} and b_{uij} are respectively the counts of representative $\hat{\beta}_{ij}^{u}$ and $\hat{\gamma}_{ij}^{u}$ in $\sum_{n \in \mathcal{H}} H_n \left(\varphi\left[\beta_{ij}^{n}\right] + \varphi\left[\gamma_{ij}^{n}\right] \right)$

  if $e \in \mathcal{Z}$ **then**

    continue

  $\hat{\mathcal{M}} \leftarrow \hat{\mathcal{M}} \cup \{(i,j)\}$

  $\mathcal{Z} \leftarrow \mathcal{Z} \cup \{e\}$

return representative constraint indices $\hat{\mathcal{M}}$

(4.2.4) as follows:

$$f_{ij}^{uv} \xrightarrow{\text{represented by}} \varphi\left[f_{ij}^{uv}\right] = \hat{f}_{\phi'(\phi(i))\phi'(\phi(j))}^{\phi(u)\phi(v)}, \tag{4.3.2}$$

where $\phi = \pi[u,v]$, and $\phi' = \sigma^{\phi(u)\phi(v)}[i,j]$ for every $(u,v,i,j) \in \mathcal{C} \times \mathcal{L}$.

These representatives $[\hat{\beta}_{ij}^{u}], [\hat{\gamma}_{ij}^{u}], [\hat{f}_{ij}^{uv}]$ significantly reduce the variables in (4.2.4), resulting in a scalable linear program.

4.3.4 Automorphism-invariant Formulation

Finally, we formulate the scalable linear program designed to scale to larger network sizes as follows:

$$\begin{aligned}
\text{Minimize} \quad & \eta \\
\text{Subject to} \quad & \sum_{u \in \mathcal{H}} H_u \left(\varphi\left[\beta_{ij}^{u}\right] + \varphi\left[\gamma_{ij}^{u}\right]\right) \leq \eta \ , \forall (i,j) \in \hat{\mathcal{M}} \\
& \frac{\hat{f}_{ij}^{uv}}{C_{ij}} - \varphi\left[\beta_{ij}^{u}\right] - \varphi\left[\gamma_{ij}^{v}\right] \leq 0 \\
& \qquad\qquad\qquad\qquad , \forall (u,v) \in \hat{\mathcal{C}}, \forall (i,j) \in \hat{\mathcal{L}}^{uv} \\
& \hat{\beta}_{ij}^{u} \in \mathbb{R}_+, \ \hat{\gamma}_{ij}^{u} \in \mathbb{R}_+, \forall (u,i,j) \in \hat{\mathcal{G}} \\
& [\hat{f}_{ij}^{uv}] \in \hat{\mathcal{F}},
\end{aligned} \tag{4.3.3}$$

where the set $\hat{\mathcal{F}}$ is defined in (4.3.4).

$$\hat{\mathcal{F}} = \left\{ \begin{array}{ll}
\sum_{j \in \mathcal{O}(i)} \hat{f}_{ij}^{uv} - \sum_{j \in \mathcal{I}(i)} \hat{f}_{ji}^{uv} = \mathbb{I}[i=u] - \mathbb{I}[i=v] & \\
\qquad\qquad , \forall (u,v) \in \hat{\mathcal{C}}, \forall i \in \mathcal{N} & \\
\hat{f}_{ij}^{uv} = 0 \qquad , \forall (u,v) \in \hat{\mathcal{C}}, \forall (i,j) \in \hat{\mathcal{L}}^{uv} \cap (\mathcal{N} \times \mathcal{P}_{-v}) & \\
\hat{f}_{iu}^{uv} = \hat{f}_{vi}^{uv} = 0, \forall (u,v) \in \hat{\mathcal{C}}, \forall (i,u),(v,i) \in \hat{\mathcal{L}}^{uv} & \\
\hat{f}_{ij}^{uv} \in \mathbb{R}_+ \qquad , \forall (u,v) \in \hat{\mathcal{C}}, \forall (i,j) \in \hat{\mathcal{L}}^{uv} &
\end{array} \right\} \tag{4.3.4}$$

The set $\hat{\mathcal{M}}$ in (4.3.3) is constructed by Algorithm 3 to remove unnecessary constraints, causing intractability for large networks.

The scalable linear program in (4.3.3) designs an optimal oblivious routing solution in polynomial time for large networks. As shown in Chapter 5, it is faster and more scalable than the state-of-the-art technique [22]. While this result achieves

the best of both worlds given the set of all traffic matrices, one may want to design the optimal oblivious routing for other traffic sets. The next section generalizes our approach to support such situations.

4.4 Generalized Traffic Model

Some datacenter administrators have the knowledge of the traffic demand ranges or may want to tread commodities differently [21, 35]. In this section, we extend our approach to support customized traffic models such that the traffic demand of commodity (u,v) is restricted to the range $[t^{uv}_{\min}, t^{uv}_{\max}]$ where $t^{uv}_{\min}$ and $t^{uv}_{\max}$ denote respective the minimum and maximum for every commodity $(u,v) \in \mathcal{C}$. Given the ranges, we subject the oblivious routing optimization in (4.1.3) to the following generalized traffic set $\mathcal{T}'$ (instead of $\mathcal{T}$):

$$\mathcal{T}' = \left\{ \begin{array}{ll} \sum_{v \in \mathcal{H}_{-u}} t^{uv} \leq H_u & ,\forall u \in \mathcal{H} \\ \sum_{u \in \mathcal{H}_{-v}} t^{uv} \leq H_v & ,\forall v \in \mathcal{H} \\ t^{uv}_{\min} \leq t^{uv} \leq t^{uv}_{\max} & ,\forall (u,v) \in \mathcal{C} \\ t^{uv} \in \mathbb{R}_+ & ,\forall (u,v) \in \mathcal{C} \end{array} \right\}. \tag{4.4.1}$$

Following the transformation technique in Section 4.2 yields the linear program in (4.4.2) with dual variables $[\beta^u_{ij}], [\gamma^u_{ij}], [\lambda^{uv}_{ij}]$, and $[\mu^{uv}_{ij}]$, where the last two groups of variables correspond to the range constraint, $t^{uv}_{\min} \leq t^{uv} \leq t^{uv}_{\max}$ introduced in (4.4.1).

$$\begin{array}{ll} \text{Minimize} & \eta \\ \text{Subject to} & \displaystyle\sum_{u \in \mathcal{H}} H_u \left(\beta^u_{ij} + \gamma^u_{ij}\right) + \sum_{(u,v) \in \mathcal{C}} t^{uv}_{\max} \lambda^{uv}_{ij} \\ & \qquad \displaystyle - \sum_{(u,v) \in \mathcal{C}} t^{uv}_{\min} \mu^{uv}_{ij} \leq \eta \qquad ,\forall (i,j) \in \mathcal{L} \\ & \displaystyle\frac{f^{uv}_{ij}}{C_{ij}} - \beta^u_{ij} - \gamma^v_{ij} - \lambda^{uv}_{ij} + \mu^{uv}_{ij} \leq 0 \\ & \qquad\qquad ,\forall (u,v) \in \mathcal{C}, \forall (i,j) \in \mathcal{L} \\ & \beta^u_{ij} \in \mathbb{R}_+,\ \gamma^u_{ij} \in \mathbb{R}_+, \forall u \in \mathcal{H}, \forall (i,j) \in \mathcal{L} \\ & \lambda^{uv}_{ij} \in \mathbb{R}_+,\ \mu^{uv}_{ij} \in \mathbb{R}_+, \forall (u,v) \in \mathcal{C}, \forall (i,j) \in \mathcal{L} \\ & [f^{uv}_{ij}] \in \mathcal{F}. \end{array} \tag{4.4.2}$$

Similar to Section 4.3, the scalability of the above linear program is further

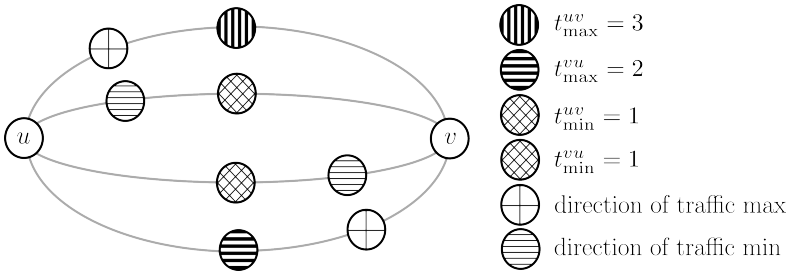

Figure 4.4 How vertices and edges are added between commodities (u,v) and (v,u) to accommodate their range constraints.

improved by exploiting the repeated structures in the network and formulation. The automorphism in Definition 1 is added with 1) preservation of the minimum demand: $t^{\phi(u)\phi(v)}_{\min} = t^{uv}_{\min}, \forall (u,v) \in \mathcal{C}$ and 2) preservation of the maximum demand: $t^{\phi(u)\phi(v)}_{\max} = t^{uv}_{\max}, \forall (u,v) \in \mathcal{C}$, to accommodate the range constraint in (4.4.1). The generators of the modified definition can be obtained by adding colored vertices and edges for each commodity to Algorithm 1 as shown in Figure 4.4. In particular, we introduce two new vertices and three edges for the maximum demand preservation of commodity (u,v). The two vertices represent the maximum demand and its direction. The first edge connects source vertex u to the directional vertex, the second edge connects both vertices, and the third edge connects the demand vertex to destination vertex v. Any two maximum demand vertices having the same demand value are assigned the same color, and different colors are assigned to vertices having different demand values. All directional vertices are assigned an identical color for every commodity. Vertices and edges for minimum demand preservation are added in a similar manner. Notice that, the time complexity of modified Algorithm 1 is $O(|\mathcal{N}| + |\mathcal{L}| + |\mathcal{C}|)$.

It is worth mentioning that when the maximum and minimum demands are symmetry, such that $t^{uv}_{\max} = t^{vu}_{\max}$ and $t^{uv}_{\min} = t^{vu}_{\min}, \forall (u,v) \in \mathcal{C}$, a generator set can be obtained by a graph constructed from Algorithm 1 without any modification.

With the above definition of automorphism, let Φ' and $\hat{\Phi}'$ be the sets of automorphism and generators. It is possible to show that some optimal solution to the linear program in (4.4.2) has repeated structures. We state the result below, which can be proved by techniques used in Lemma 2 and Theorem 2.

Theorem 3. *There exists an automorphism-invariant solution*

$\left(\hat{\eta}, [\hat{\beta}_{ij}^{u}], [\hat{\gamma}_{ij}^{u}], [\hat{\lambda}_{ij}^{uv}], [\hat{\mu}_{ij}^{uv}], [\hat{f}_{ij}^{uv}]\right)$ *that is optimal for the linear program in* (4.4.2) *and satisfies:*

$$\begin{aligned}
&\hat{f}_{ij}^{uv} = \hat{f}_{\phi(i)\phi(j)}^{\phi(u)\phi(v)}, \quad \hat{\beta}_{ij}^{u} = \hat{\beta}_{\phi(i)\phi(j)}^{\phi(u)}, \quad \hat{\gamma}_{ij}^{u} = \hat{\gamma}_{\phi(i)\phi(j)}^{\phi(u)}, \\
&\hat{\lambda}_{ij}^{uv} = \hat{\lambda}_{\phi(i)\phi(j)}^{\phi(u)\phi(v)}, \quad \hat{\mu}_{ij}^{uv} = \hat{\mu}_{\phi(i)\phi(j)}^{\phi(u)\phi(v)}, \quad \forall \phi \in \Phi'.
\end{aligned}$$

Theorem 3 implies that variables in an automorphism-invariant solution form groups, and every variable in each group takes the same value. Comparing Theorem 3 with Theorem 2, we first observe that variables $[\hat{\beta}_{ij}^{u}]$, $[\hat{\gamma}_{ij}^{u}]$ and $[\hat{f}_{ij}^{uv}]$ in both theorems form groups using the same formation. Therefore, they represent variables $[\beta_{ij}^{u}]$, $[\gamma_{ij}^{u}]$ and $[f_{ij}^{uv}]$ of the linear program in (4.4.2) as the mappings in (4.4.3) and (4.3.2) respectively. Secondly, we observe that the variables $[\lambda_{ij}^{uv}]$, $[\mu_{ij}^{uv}]$ share the same group relations as they share the same index set, $\mathcal{C} \times \mathcal{L}$. Therefore, we adapt Algorithm 2 for grouping the variables $[\lambda_{ij}^{uv}]$, $[\mu_{ij}^{uv}]$ by searching over the index set, $\mathcal{C} \times \mathcal{L}$, instead. Notice that, the time complexity of grouping the variables $[\lambda_{ij}^{uv}]$ and $[\mu_{ij}^{uv}]$ is $O(|\mathcal{C}|\,|\mathcal{L}|\,|\hat{\Phi}|)$. Let $\hat{\lambda}_{ij}^{uv}$ and $\hat{\mu}_{ij}^{uv}$ be representative variables for every $(u,v,i,j) \in \mathcal{C} \times \mathcal{L}$. These representatives represent variables $[\lambda_{ij}^{uv}]$ and $[\mu_{ij}^{uv}]$ of the linear program in (4.4.2) as follows:

$$\begin{aligned}
&\lambda_{ij}^{uv} \xrightarrow{\text{represented by}} \varphi\left[\lambda_{ij}^{uv}\right] = \hat{\lambda}_{\phi(i)\phi(j)}^{\phi(u)\phi(v)} \quad \text{where } \phi = \omega\,[u,v,i,j] \\
&\mu_{ij}^{uv} \xrightarrow{\text{represented by}} \varphi\left[\mu_{ij}^{uv}\right] = \hat{\mu}_{\phi(i)\phi(j)}^{\phi(u)\phi(v)} \quad \text{where } \phi = \omega\,[u,v,i,j]
\end{aligned} \tag{4.4.3}$$

for every $(u,v,i,j) \in \mathcal{C} \times \mathcal{L}$. We use $\varphi[x]$ to denote the representative of x.

Finally, we formulate the scalable linear program that considers a generalized traffic set $\mathcal{T}'$ in (4.4.1) as follows:

$$
\begin{aligned}
&\text{Minimize } \eta \\
&\text{Subject to } \sum_{u\in\mathcal{H}} H_u\left(\varphi\left[\beta_{ij}^u\right]+\varphi\left[\gamma_{ij}^u\right]\right)+\sum_{(u,v)\in\mathcal{C}} t_{\max}^{uv}\,\varphi\left[\lambda_{ij}^{uv}\right] \\
&\qquad -\sum_{(u,v)\in\mathcal{C}} t_{\min}^{uv}\,\varphi\left[\mu_{ij}^{uv}\right]\le\eta \qquad ,\forall(i,j)\in\hat{\mathcal{M}}' \\
&\frac{\hat{f}_{ij}^{uv}}{C_{ij}}-\varphi\left[\beta_{ij}^u\right]-\varphi\left[\gamma_{ij}^v\right]-\hat{\lambda}_{ij}^{uv}+\hat{\mu}_{ij}^{uv}\le 0 \qquad (4.4.4) \\
&\qquad\qquad ,\forall(u,v)\in\hat{\mathcal{C}},\forall(i,j)\in\hat{\mathcal{L}}^{uv} \\
&\hat{\beta}_{ij}^u\in\mathbb{R}_+,\ \hat{\gamma}_{ij}^u\in\mathbb{R}_+,\forall(u,i,j)\in\hat{\mathcal{G}} \\
&\hat{\lambda}_{ij}^{uv}\in\mathbb{R}_+,\ \hat{\mu}_{ij}^{uv}\in\mathbb{R}_+,\forall(u,v)\in\hat{\mathcal{C}},\forall(i,j)\in\hat{\mathcal{L}}^{uv} \\
&[\hat{f}_{ij}^{uv}]\in\hat{\mathcal{F}},
\end{aligned}
$$

where the sets $\hat{\mathcal{F}}$ is defined in (4.3.4). The set $\hat{\mathcal{M}}'$ in (4.4.4) is constructed by adjusting Algorithm 3 to include only one link from a group of links that have identical counts of representatives $([\hat{\beta}_{ij}^u],[\hat{\gamma}_{ij}^u],[\hat{\lambda}_{ij}^{uv}],[\hat{\mu}_{ij}^{uv}])$ in $\sum_{n\in\mathcal{H}} H_n(\varphi\left[\beta_{ij}^n\right]+\varphi\left[\gamma_{ij}^n\right])+\sum_{(u,v)\in\mathcal{C}} t_{\max}^{uv}\varphi\left[\lambda_{ij}^{uv}\right]-\sum_{(u,v)\in\mathcal{C}} t_{\min}^{uv}\varphi\left[\mu_{ij}^{uv}\right]$.

To conclude this section, when we have the knowledge of demand ranges, the scalable linear program in (4.4.4) gives an optimal oblivious routing solution with respect to the demand ranges. This optimal solution is no worse than a solution from (4.3.3), which ignores the demand ranges. It is worth noting that when the demand range is loosened to $[0,\infty)$ for every commodity, the optimal solution from the linear program in (4.4.4) becomes identical to the optimal solution from (4.3.3). The empirical result in Section 5.7 also confirms this insight. The next section describes how an optimal solution is compacted for practicality.

4.5 Compact Forwarding Rules

An optimal oblivious routing solution, obtained from the scalable formulation either in (4.3.3) or (4.4.4), could be distributively deployed on switches in a datacenter network. Every switch decides how traffic is split over to next-hop switches for each

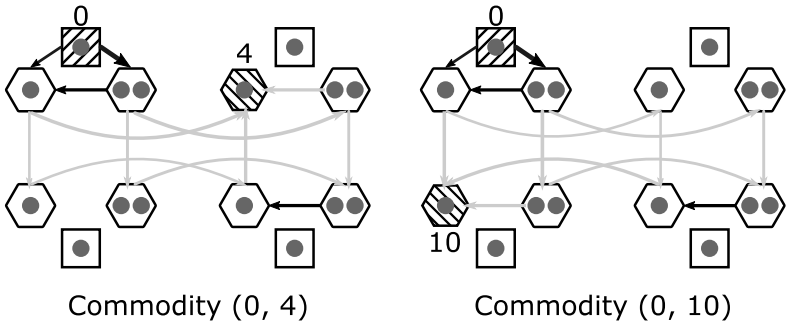

Figure 4.5 Optimal oblivious routing solution of commodities $(0,4)$ and $(0,10)$, which have the same representative, is presented. The black arrows represent similar split weights that can be grouped at each corresponding node.

commodity[5]. In particular, switch i splits the traffic of commodity (u,v) to next-hop switch j over link (i,j) with weight

$$w_{ij}^{uv} = \frac{f_{ij}^{uv}}{\sum_{k\in\mathcal{O}(i)} f_{ik}^{uv}} \quad ,\forall(u,v)\in\mathcal{C},\forall i\in\mathcal{N},\forall j\in\mathcal{O}(i). \tag{4.5.1}$$

These weights could be configured into programmable switches with flow splitting capability in the form of forwarding rules [27, 29, 30]. However, a switch has limited memory for storing the rules from multiple commodities in a large datacenter network. This practical limitation could hinder the deployment of the optimal oblivious routing solution.

We observe that some rules, derived from the commodities having the same representative, are similar as shown in Figure 4.5. We use this insight to group the forwarding rules in order to reduce memory requirement. Algorithm 4 summarized our grouping method. For each node in a network, the commodities having the same representative are grouped by the similarity of split weights, i.e., $(i,j,\varphi\left[f_{ij}^{xy}\right])_{(i,j)\in\mathcal{O}(n)}$ (ignoring the common denominator in (4.5.1)). The commodities having similar split weights form a group, so one collection of forwarding rules is sufficient for these commodities. The algorithm outputs the collection of grouped rules, $\{W_n\}_{n\in\mathcal{N}}$. For node n, the grouped rules are stored in the dictionary W_n whose key e represents a collection of split weights and the value $W_n[e]$ is the set of commodities using the weights. The time complexity of Algorithm 4 is $O(|\mathcal{N}|\,|\mathcal{C}|)$. Note that Algorithm 4 is highly parallelizable as the grouping

[5]TCP reordering effect could be avoided by ensuring that packets belonging to the same TCP or UDP flow are routed along the same path, which is out of scope for this thesis [53, 54].

process can be parallelized for each node.

Algorithm 4: Forward-rule grouping

Initialize empty set $\mathcal{Q}$

Initialize dictionary $\{W_n\}_{n\in\mathcal{N}}$

for $n \in \mathcal{N}$ **do**

  $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{n\}$

  for $(u,v) \in \hat{\mathcal{C}}$ **do**

    for (x,y) *represented by* (u,v) **do**

      Let $e = (i, j, \varphi\left[f_{ij}^{xy}\right])_{(i,j)\in\mathcal{O}(n)}$

      if $e \notin W_n$ **then**

        $W_n[e] \leftarrow \emptyset$

      $W_n[e] \leftarrow W_n[e] \cup \{(x,y)\}$

return Grouped forwarding rules $\{W_n\}_{n\in\mathcal{N}}$

In short, Algorithm 4 utilizes the repeated structures in the optimal oblivious routing solution, which is automorphism invariant, to reduce memory requirement for the deployment of forwarding rules in real-world switches. In addition, if the reduced requirement exceeds the available memory of a switch, an approximation technique, such as [27], could be applied to our grouped rules to trade-off between optimality and available memory. In other words, our grouping method circumvents the unnecessary trade-off when it is avoidable.

Chapter 5

Experimental Results

In this chapter, our scalable linear program in (4.3.3) is evaluated over various datacenter network topologies and sizes. Its scalability is evaluated in terms of computation times (Section 5.3) and problem sizes (Section 5.4). The efficiency of our grouping method in Algorithm 4 is evaluated for the same topologies (Section 5.5). Ultimately, we demonstrate our work's applicability to an existing server-centric topology (Section 5.6) and the presence of traffic demand knowledge (Section 5.7).

Every evaluation is executed on a commodity computer with an Intel Core i9-12900K processor and 128GB memory. All linear programs are solved by `Gurobi` [37]. We use off-the-shelf software, `nauty` [52] to compute generator sets.

5.1 Topology Setting and Brief Background

We motivate the topologies used in our evaluation as follows. FatClique is designed for low-cost manageability, regarding topology deployment and expansion [2]. SlimFly focuses on throughput performance by the use of low-diameter graphs with high-radix switches [10]. BCube aims for shipping-container-based datacenters [11]. FatClique and SlimFly follow the server-switch architecture, while BCube belongs to the server-centric architecture.

For topology setting, we intend to provide the example demonstrating ability of this work to design optimal routing for larger networks, comparing it with existing methods. These topologies are constructed as follows. For FatClique, we set the numbers of switches in a sub-block, sub-blocks in a block, and blocks to an identical value. We vary this value from 2 to 12. For SlimFly, we use its provided topologies with the number of network radices ranging from 5 to 43. For BCube, each topology is built from 4-port switches, and we vary the number of levels of switches from 2 to 5. It is worth noting that the size of the network increases as the parameter value increases. The maximum

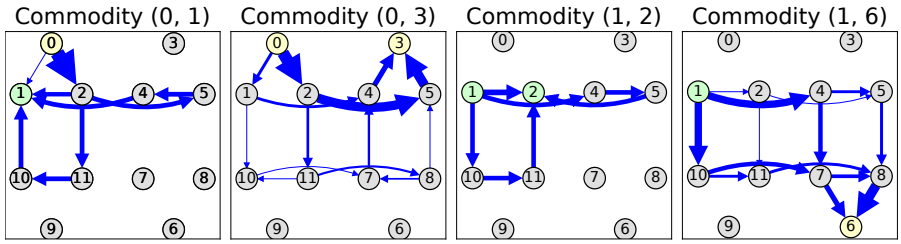

Figure 5.1 The optimal oblivious routing solution of the network in Figure 4.1. The four commodities correspond to all combinations of node types.

number of nodes in the topologies generated by these settings is 2304.

5.2 Correctness of Optimal Solutions

In every evaluation, we confirm the correctness of the optimal solutions obtained from (4.3.3) by comparing them with the solutions from the state-of-the-art technique in [22]. When the technique fails to compute solutions due to computational resource limitations, we adopt another technique in [55] to validate our solutions.

Moreover, as shown in Figure 5.1, the scalable linear program in (4.3.3) handles routing-incapable nodes correctly for the simple network in Figure 4.1. Every routing-incapable node unrelated to a considered commodity is not involved in the commodity's routing.

5.3 Scalability in Terms of Computation Time

We vary the sizes of FatClique, SlimFly, and BCube. We record the optimization times of our scalable formulation in (4.3.3), the state-of-the-art technique in [22], and the other linear program in [21]. The limit of the optimization time is set to 24 hours. The results are plotted in Figure 5.2. Our work takes much less time to find optimal routing solutions than the other techniques for all topologies and all sizes. The work in [22] scales well for FatClique and BCube but fails to obtain optimal routing solutions within the time limit for SlimFly beyond 98 nodes. Due to insufficient computing memory, the linear program in [21] cannot scale beyond 112 nodes for all considered topologies.

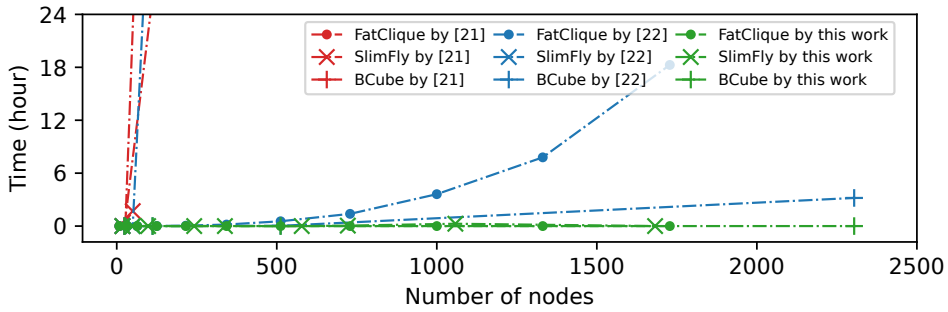

Figure 5.2 The optimization time at different sizes of FatClique, SlimFly, and BCube. The maximum times of the scalable formulation are 0.008 sec. for FatClique, 2.97 min. for SlimFly, and 0.026 sec. for BCube.

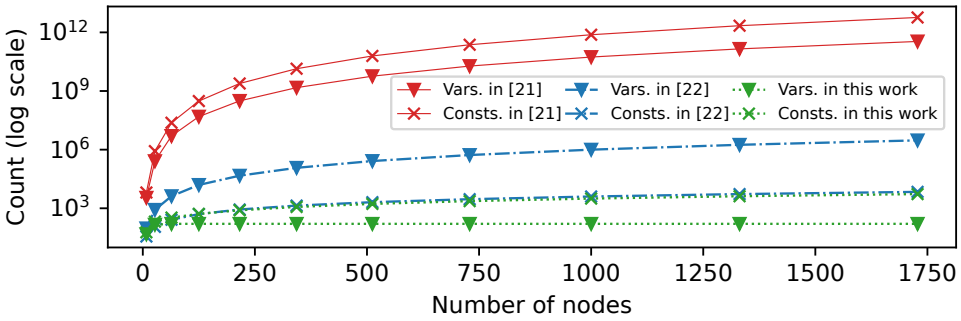

Figure 5.3 The numbers of variables and constraints at various sizes of FatClique. For the largest FatClique, the numbers of variables and constraints in the scalable formulation are 160 and 5326 respectively.

5.4 Scalability in Terms of Variables and Constraints

Figure 5.3 shows the numbers of variables and constraints at different sizes of FatClique from 8 to 1728 nodes, which follows the same evaluation in [22]. Our scalable formulation in (4.3.3) is much leaner than the other techniques regarding the numbers of variables and constraints. While the linear program in [21] can obtain an optimal routing solution in polynomial time, its formulation size grows exponentially as the topology size increases, resulting in the insufficient memory issue in Section 5.3. Because the state-of-the-art technique in [22] iteratively solves two linear programs, we only count the total numbers of variables and constraints from the two programs at the first iteration without additional constraints from later iterations. Nevertheless, their formulation size (i.e., the numbers of variables and constraints) is still larger than ours.

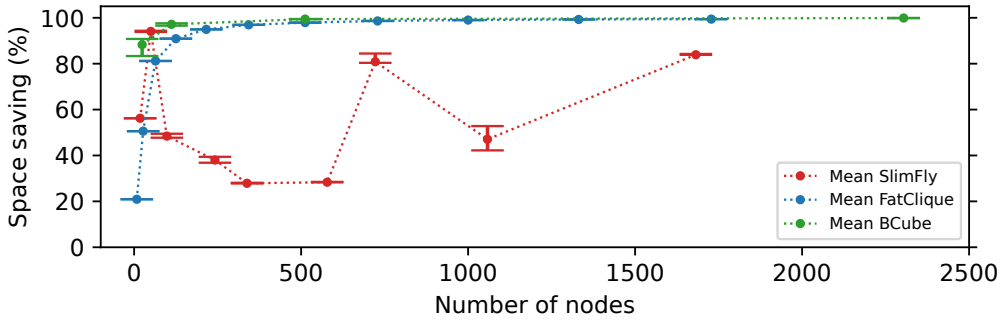

Figure 5.4 The space-saving percentage at different sizes of FatClique, SlimFly, and BCube. Each mean value is computed from all nodes in a topology. The maximum and minimum values are represented by horizontal bars.

5.5 Reduction of Forwarding Rules

The efficiency of the forwarding rule grouping from Algorithm 4 is measured by the percentage of space saving, which is the proportion of rule reduction to non-grouped rules. Figure 5.4 shows the space saving at various sizes of FatClique, SlimFly, and BCube. Our grouping method reduces more than 90% of the non-grouped forwarding rules under FatClique with no less than 216 nodes and under BCube with no less than 112 nodes. The space saving for SlimFly highly depends on topology configuration.

5.6 Possible Application for BCube

BCube employs dynamic source routing to utilize multi-path capacity. Each source selects the best-quality path from its candidate paths, based on the current maximum available bandwidth. This approach constantly measures available bandwidth, introduces complexity, and may need specialized hardware and network stack. Instead, one could employ the oblivious routing approach, such as ECMP routing which is easy to implement, on BCube for simpler production.

To illustrate the network performance improvement Table 5.1 shows the results of our optimal oblivious routing solution and ECMP routing, which is an oblivious routing with equal split. The optimal solution achieves a performance improvement of 1.8× to 6.7× in terms of the maximum congestion ratio over the equal split.

Table 5.1 The maximum congestion ratio in BCube with oblivious routing

BCube	Maximum congestion ratio		
#Nodes	Our work	Equal split	Improvement
24	2.50	4.50	1.80x
112	4.00	11.49	2.87x
512	5.50	21.50	3.91x
2304	6.97	47.19	6.77x

5.7 Performance Gain with Insights about Traffic Demand

In practice, some datacenter administrator team has an insight into traffic demands occurring in a datacenter network. This insight could come from historical measurements or capacity planning. For example, Google observes that the actual traffic demands in their datacenter networks can be well approximated by the Gravity model [35]. Therefore, one could deploy an optimal routing solution from the method in Section 4.4 with the knowledge of demand ranges.

In this experiment, we assume that traffic in a datacenter network follows the Gravity model where the base demand of commodity (u,v) is $t^{uv}_{\text{base}} = H_u H_v / \sum_{k\in\mathcal{H}} H_k$. Let δ be a non-negative margin parameter. The demand range of commodity (u,v) in the generalized traffic set $\mathcal{T}'$ in (4.4.1) is set to $[t^{uv}_{\text{base}}/\delta, t^{uv}_{\text{base}}\delta]$ for every commodity.

When the margin parameter δ increases from 1 to 10, where a larger margin parameter corresponds to a more relax demand range, the maximum congestion ratio as the results of the routing solutions from the scalable linear programs in (4.4.4) and (4.3.3) (with and without the knowledge of traffic ranges) is observed. Figures 5.5 and 5.6 show the results for the toy topology and BCube respectively. Note that lower maximum congestion ratio values indicate more efficient resource utilization, and therefore are preferred. We first observe that an optimal solution from the scalable linear program without the knowledge of traffic ranges underperforms the other solution obtained with the knowledge. Besides, the differences between the two routing solutions get smaller when the range is more relaxed, as the margin parameter becomes larger. In short, the

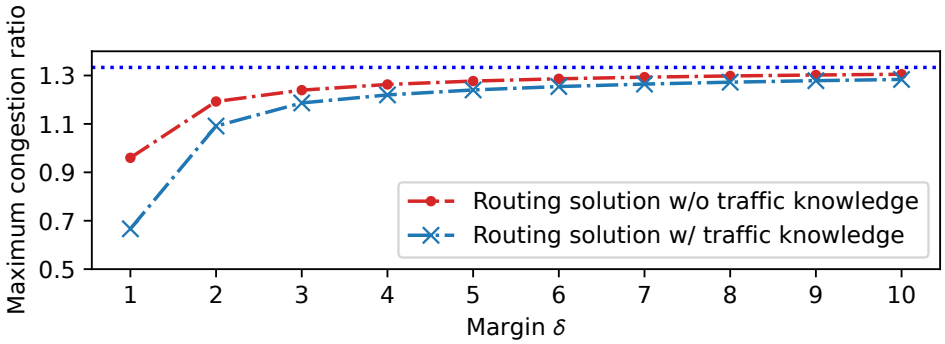

Figure 5.5 The maximum congestion ratio obtained from the routing solutions designed with and without the knowledge of traffic ranges under the Toy topology in Figure 4.1.

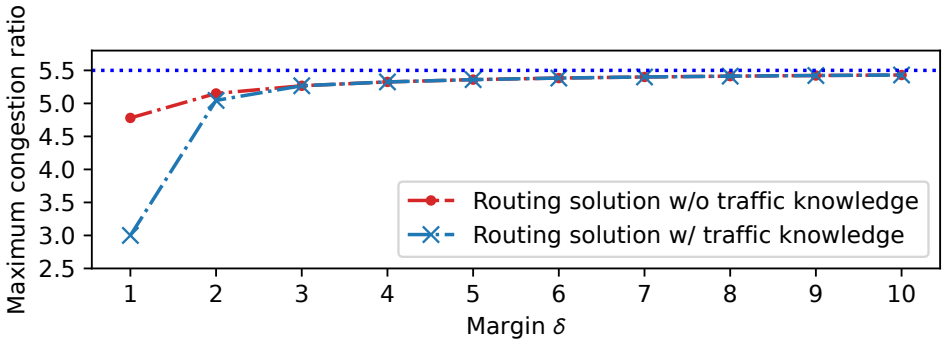

Figure 5.6 The maximum congestion ratio obtained from the routing solutions designed with and without the knowledge of traffic ranges under BCube topology with 512 nodes.

knowledge of traffic ranges can improve the performance of the optimal oblivious routing solution designed by our method in (4.4.4).

Chapter 6

Conclusion and Future Work

6.1 Conclusion

This thesis proposes a polynomial-time process for designing optimal compact oblivious routing for datacenter networks. In this process, an optimal oblivious routing solution is designed by solving a scalable linear program, derived from the transformation of a robust optimization problem and the exploitation of repeated network structures. After obtaining an optimal routing solution, the process compacts the forwarding rules converted from the optimal solution to reduce memory requirements for real-world deployment. Additionally, the design process is extended to accommodate a more generalized traffic model when the knowledge of the traffic demand ranges is available.

6.2 Potential Future Works

Topology Asymmetry: Topology asymmetry may arise, either being innate by design (e.g., Jellyfish [4], Scafida [56]) or due to several reasons, such as heterogeneous link/node capacity and network failure (specifically, link or switch failure). This asymmetry diminishes the advantage gained from exploiting repeated structures for scalability and compactness. Hence, alternative techniques to achieve scalability and compactness in cases of topological asymmetry can be explored in future studies.

Network Failure: When network failures occur in the network, the topology changes. The current routing solution may no longer be optimal, so the entire design process needs to be recomputed, which takes a considerable amount of time. Therefore, it would be a valuable opportunity to further study designing optimal oblivious routing that is also robust against network failure scenarios. In addition, the changed topology may lead to asymmetry, diminishing the advantage gained from exploiting repeated structures.

Computation speed-up: Even using the scalable linear program can yield promising solve times. However, solving this linear program may take longer than expected due to various factors, such as topology asymmetry and the huge scale of networks. This poses a crucial challenge, as it may exceed acceptable computation times (e.g., longer than one month). One possible solution is to trade off between optimality and computation speed-up [57].

Traffic Model From Real-world Measurement: In practice, datacenter network administrators frequently measure and record historical traffic demands for further analysis and management purposes. With this historical data, one can gain insights into the expected traffic demands, which are represented by the set of all convex combinations of historical traffic demands [58]. This would be a valuable opportunity to further study on how to utilize these insights in designing optimal oblivious routing.

Reference

1. Singla A, Godfrey PB, Kolla A. **High throughput data center topology design**. Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation. Seattle, WA: USENIX Association; 2014. p. 29-41.

2. Zhang M, Mysore RN, Supittayapornpong S, Govindan R. **Understanding lifecycle management complexity of datacenter topologies**. 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19). Boston, MA: USENIX Association; 2019. p. 235-54.

3. Valadarsky A, Shahaf G, Dinitz M, Schapira M. **Xpander: towards optimal-performance datacenters**. Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies. Irvine, California, USA: Association for Computing Machinery; 2016. p. 205-19.

4. Singla A, Hong CY, Popa L, Godfrey PB. **Jellyfish: Networking data centers randomly**. 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12). San Jose, CA: USENIX Association; 2012. p. 225-38.

5. Al-Fares M, Loukissas A, Vahdat A. A scalable, commodity data center network architecture. **SIGCOMM Comput Commun Rev**. 2008;38(4):63-74.

6. Singh A, Ong J, Agarwal A, Anderson G, Armistead A, Bannon R, et al. Jupiter rising: a decade of clos topologies and centralized control in google's datacenter network. **SIGCOMM Comput Commun Rev**. 2015;45(4):183-97.

7. Andreyev A. **Reinventing facebook's data center network**; 2019. Available from: `https://engineering.fb.com/2019/03/14/data-center-engineering/f16-minipack/`.

8. Greenberg A, Hamilton JR, Jain N, Kandula S, Kim C, Lahiri P, et al. Vl2: A scalable and flexible data center network. **SIGCOMM Comput Commun Rev**. 2009 Aug;39(4):51-62.

9. Harsh V, Jyothi SA, Godfrey PB. **Spineless data centers**. Proceedings of the 19th ACM Workshop on Hot Topics in Networks. Virtual Event, USA: Association for Computing Machinery; 2020. p. 67-73.

10. Besta M, Hoefler T. **Slim fly: a cost effective low-diameter network topology**. SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis; 2014. p. 348-59.

11. Guo C, Lu G, Li D, Wu H, Zhang X, Shi Y, et al. Bcube: a high performance, server-centric network architecture for modular data centers. **SIGCOMM Comput Commun Rev**. 2009;39(4):63-74.

12. Chen SS, He K, Wang R, Seshan S, Steenkiste P. **Precise data center traffic engineering with constrained hardware resources**. 21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24). Santa Clara, CA: USENIX Association; 2024. p. 669-90.

13. Al-Fares M, Radhakrishnan S, Raghavan B, Huang N, Vahdat A. **Hedera: Dynamic flow scheduling for data center networks**. Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation. USA: USENIX Association; 2010. p. 19.

14. Raiciu C, Barre S, Pluntke C, Greenhalgh A, Wischik D, Handley M. **Improving datacenter performance and robustness with multipath tcp**. Proceedings of the ACM SIGCOMM 2011 Conference. New York, NY, USA: Association for Computing Machinery; 2011. p. 266-77.

15. Kandula S, Sengupta S, Greenberg A, Patel P, Chaiken R. **The nature of data center traffic: measurements & analysis**. Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement. New York, NY, USA: Association for Computing Machinery; 2009. p. 202-8.

16. Benson T, Akella A, Maltz DA. **Network traffic characteristics of data centers in the wild**. Proceedings of the 10th ACM SIGCOMM Conference on Internet

Measurement. New York, NY, USA: Association for Computing Machinery; 2010. p. 267-80.

17. Hopps C. **Rfc2992: Analysis of an equal-cost multi-path algorithm**. USA: RFC Editor; 2000.

18. Valiant LG. A scheme for fast parallel communication. **SIAM Journal on Computing**. 1982;11(2):350-61.

19. Zhang-Shen R, McKeown N. **Designing a predictable internet backbone with valiant load-balancing**. Proceedings of the 13th International Conference on Quality of Service. Passau, Germany: Springer-Verlag; 2005. p. 178-92.

20. Kodialam M, Lakshman TV, Sengupta S. Traffic-oblivious routing in the hose model. **IEEE/ACM Transactions on Networking**. 2011;19(3):774-87.

21. Applegate D, Cohen E. **Making intra-domain routing robust to changing and uncertain traffic demands: understanding fundamental tradeoffs**. Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications. Karlsruhe, Germany: Association for Computing Machinery; 2003. p. 313-24.

22. Supittayapornpong S, Namyar P, Zhang M, Yu M, Govindan R. **Optimal oblivious routing for structured networks**. IEEE INFOCOM 2022 - IEEE Conference on Computer Communications; 2022. p. 1988-97.

23. Chiesa M, Kindler G, Schapira M. **Traffic engineering with equal-cost-multipath: An algorithmic perspective**. IEEE INFOCOM 2014 - IEEE Conference on Computer Communications; 2014. p. 1590-8.

24. Bland RG, Goldfarb D, Todd MJ. Feature article-the ellipsoid method: a survey. **Operations Research**. 1981;29(6):1039-91.

25. Karmarkar N. **A new polynomial-time algorithm for linear programming**. Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing. New York, NY, USA: Association for Computing Machinery; 1984. p. 302-11.

26. Cohen MB, Lee YT, Song Z. **Solving linear programs in the current matrix multiplication time**. Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing. Phoenix, AZ, USA: Association for Computing Machinery; 2019. p. 938-42.

27. Kang N, Ghobadi M, Reumann J, Shraer A, Rexford J. **Efficient traffic splitting on commodity switches**. Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies. Heidelberg, Germany: Association for Computing Machinery; 2015. .

28. Sadeh Y, Rottenstreich O, Kaplan H. Load balancing with minimal deviation in switch memories. **IEEE Transactions on Network and Service Management**. 2023;20(4):4283-96.

29. McKeown N, Anderson T, Balakrishnan H, Parulkar G, Peterson L, Rexford J, et al. Openflow: enabling innovation in campus networks. **SIGCOMM Comput Commun Rev**. 2008;38(2):69-74.

30. Zhou J, Tewari M, Zhu M, Kabbani A, Poutievski L, Singh A, et al. **Wcmp: weighted cost multipathing for improved fairness in data centers**. Proceedings of the Ninth European Conference on Computer Systems. Amsterdam, The Netherlands: Association for Computing Machinery; 2014. .

31. Räcke H, Schmid S. **Compact oblivious routing**. Bender MA, Svensson O, Herman G, editors. 27th Annual European Symposium on Algorithms (ESA 2019). vol. 144 of Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik; 2019. p. 75:1-75:14.

32. Czerner P, Räcke H. **Compact oblivious routing in weighted graphs**. Grandoni F, Herman G, Sanders P, editors. 28th Annual European Symposium on Algorithms (ESA 2020). vol. 173 of Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik; 2020. p. 36:1-36:23.

33. Räcke H. **Minimizing congestion in general networks**. The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.; 2002. p. 43-52.

34. Duffield NG, Goyal P, Greenberg A, Mishra P, Ramakrishnan KK, van der Merive JE. **A flexible model for resource management in virtual private networks**. Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication. Cambridge, Massachusetts, USA: Association for Computing Machinery; 1999. p. 95-108.

35. Poutievski L, Mashayekhi O, Ong J, Singh A, Tariq M, Wang R, et al. **Jupiter evolving: transforming google's datacenter network via optical circuit switches and software-defined networking**. Proceedings of the ACM SIGCOMM 2022 Conference. Amsterdam, Netherlands: Association for Computing Machinery; 2022. p. 66-85.

36. Kodialam M, Lakshman TV, Sengupta S. **Maximum throughput routing of traffic in the hose model**. Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications. Barcelona, Spain; 2006. p. 1-11.

37. Gurobi Optimization, LLC. **Gurobi Optimizer Reference Manual**; 2023. Available from: `https://www.gurobi.com`.

38. Ahuja RK, Magnanti TL, Orlin JB. **Network flows: theory, algorithms, and applications**. USA: Prentice-Hall, Inc.; 1993.

39. Valiant LG, Brebner GJ. **Universal schemes for parallel communication**. Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing. Milwaukee, Wisconsin, USA: Association for Computing Machinery; 1981. p. 263-77.

40. Räcke H. **Optimal hierarchical decompositions for congestion minimization in networks**. Proceedings of the Fortieth Annual ACM Symposium on Theory

of Computing. Victoria, British Columbia, Canada: Association for Computing Machinery; 2008. p. 255-64.

41. Azar Y, Cohen E, Fiat A, Kaplan H, Räcke H. **Optimal oblivious routing in polynomial time**. Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing. San Diego, CA, USA: Association for Computing Machinery; 2003. p. 383-8.

42. Fakcharoenphol J, Rao S, Talwar K. **A tight bound on approximating arbitrary metrics by tree metrics**. Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing. San Diego, CA, USA: Association for Computing Machinery; 2003. p. 448-55.

43. Spring N, Mahajan R, Wetherall D, Anderson T. Measuring isp topologies with rocketfuel. **IEEE/ACM Transactions on Networking**. 2004;12(1):2-16.

44. Ahn JH, Binkert N, Davis A, McLaren M, Schreiber RS. **Hyperx: topology, routing, and packaging of efficient large-scale networks**. Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis. Portland, Oregon: Association for Computing Machinery; 2009. .

45. Kim J, Dally WJ, Scott S, Abts D. **Technology-driven, highly-scalable dragonfly topology**. 2008 International Symposium on Computer Architecture; 2008. p. 77-88.

46. Guo C, Wu H, Tan K, Shi L, Zhang Y, Lu S. Dcell: a scalable and fault-tolerant network structure for data centers. **SIGCOMM Comput Commun Rev**. 2008;38(4):75-86.

47. Broadcom. **Tomahawk4 / BCM56990 Series**; 2023. Available from: `https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm56990-series`.

48. Broadcom. **Broadcom / 2 x 100GbE PCIe NIC**; 2023. Available from: `https://www.broadcom.com/products/ethernet-connectivity/network-adapters/p2100g`.

49. Supittayapornpong S, Raghavan B, Govindan R. **Towards highly available clos-based wan routers**. Proceedings of the ACM Special Interest Group on Data Communication. Beijing, China: Association for Computing Machinery; 2019. p. 424-40.

50. Boyd S, Vandenberghe L. **Convex Optimization**. Cambridge University Press; 2004.

51. Godsil C, Royle GF. **Algebraic Graph Theory**. No. Book 207 in Graduate Texts in Mathematics. Springer; 2001.

52. McKay BD, Piperno A. Practical graph isomorphism, II. **Journal of Symbolic Computation**. 2014;60:94-112.

53. Kandula S, Katabi D, Sinha S, Berger A. Dynamic load balancing without packet reordering. **SIGCOMM Comput Commun Rev**. 2007 mar;37(2):51-62.

54. Vanini E, Pan R, Alizadeh M, Taheri P, Edsall T. **Let it flow: Resilient asymmetric load balancing with flowlet switching**. 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17). Boston, MA: USENIX Association; 2017. p. 407-20.

55. Towles B, Dally WJ. Worst-case traffic for oblivious routing functions. **IEEE Computer Architecture Letters**. 2002;1(1):4-4.

56. Gyarmati L, Trinh TA. Scafida: a scale-free network inspired data center architecture. **SIGCOMM Comput Commun Rev**. 2010 oct;40(5):4-12.

57. Narayanan D, Kazhamiaka F, Abuzaid F, Kraft P, Agrawal A, Kandula S, et al. **Solving large-scale granular resource allocation problems efficiently with pop**. Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles. New York, NY, USA: Association for Computing Machinery; 2021. p. 521-37.

58. Wang H, Xie H, Qiu L, Yang YR, Zhang Y, Greenberg A. Cope: traffic engineering in dynamic networks. **SIGCOMM Comput Commun Rev**. 2006 aug;36(4):99-110.