NAMP: Network-Aware Multipathing in Software-Defined Data Center Networks

Yingying Cheng and Xiaohua Jia, *Fellow, IEEE*

***Abstract*— Data center networks employ parallel paths to perform load balancing. Existing traffic splitting schemes propose weighted traffic distribution across multiple paths via a centralized view. An SDN controller computes the traffic splitting ratio of a flow group among all the paths, and implements the ratio by creating multiple rules in the flow table of OpenFlow switches. However, since the number of rules in TCAM-based flow table is limited, it is not scalable to implement the ideal splitting ratio for every flow group. Existing solutions, WCMP and Niagara, aim at reducing the maximum oversubscription of all egress ports and reducing traffic imbalance, respectively. However, the transmission time of flow groups, which measures the quality of cloud services, is sub-optimal in existing solutions that ignore heterogeneous network bandwidth. We propose and implement NAMP, a multipathing scheme considering the network heterogeneity, to efficiently optimize the transmission time of flow groups. Experimental results show that NAMP reduces the transmission time by up to 45.4% than Niagara, up to 50% than WCMP, and up to 60% than ECMP.**

***Index Terms*— Data center networks, software-defined networks, multipath routing, rule allocation.**

I. Introduction

Today's data center networks employ symmetric tree-based topologies, e.g, Fat-Tree and Clos topology, among tens of thousands of servers to provide a large bisection bandwidth. There are often multiple paths between each pair of servers. It is crucial to efficiently balance traffic load across the multiple paths between any two servers to fully utilize the bisection bandwidth. Canonical load balancing mechanism, e.g., equal-cost multipath routing (ECMP), uniformly splits traffic across multiple paths between two servers. However, ECMP leads to traffic imbalance because it assumes the network is regular and homogeneous.

In order to overcome ECMP's limitations, existing solutions employ weighted traffic splitting across multiple paths based on a global view provided by Software-Defined Networks (SDN). For a flow group, which is identified by the destination prefix, SDN enables each switch to perform the weighted traffic distribution among multiple next-hops by crafting a set of wildcard rules in the flow table. However, the number of flow entries in a flow table is very much limited. Flow tables in today's OpenFlow switches are implemented using Ternary Content Addressable Memory (TCAM), which is an expensive hardware resource [1]. A commodity switch typically supports up to a few thousands of entries [2]. In the weighted multipath routing, the traffic of a flow group, which belongs to one cloud service, typically takes 74 routing rules on average and 288 rules in the worst case. In other words, only tens of cloud services can be supported by a data center network in the worst case. The weighted traffic splitting scheme is not scalable due to too many routing rules required for one flow group and the limited rule entries in off-the-shelf switches.

In order to provide scalable multipath routing in data center networks, it is necessary to solve the table fitting problem that allocates routing rules in flow tables to flow groups. A flow group is a group of flows that have the same source edge switch and the same destination edge switch. A flow group usually belongs to one cloud service, such as a social networking service. Table fitting problem for one flow group concerns how many routing rules should be allocated to each egress port in each switch. For multiple flow groups, the table fitting problem is allocating the rules to the flow groups. The number of routing rules allocated to a flow group determines the traffic distribution across multiple paths. Subsequently, both bandwidth utilization and quality of service are affected.

There has been significant interest on the scalable multipath routing in existing literatures. WCMP [3] and Niagara [4] are two state-of-the-art solutions considering the limited size of flow tables. Both of the solutions are based on a set of ideal traffic splitting weights initially computed by an SDN controller. WCMP and Niagara then re-compute a set of new weights for each flow group under the constraint of flow table capacity to optimize some objectives. In particular, WCMP [3] focuses on minimizing the maximum oversubscription of all egress ports in one switch under the limited number of routing rules. The oversubscription of each egress port is defined as its new weight divided by its ideal weight. While Niagara aims at minimizing the traffic imbalance, i.e., the fraction of traffic sent to the wrong next-hops based on the ideal splitting weights, under the limited entries in the flow table.

Most of data center traffic is generated by online applications such as web searching, social networking and instant messaging, which are time-critical services. An online service usually generates a group of flows between multiple racks in a data center. In order to guarantee quality of service, minimizing the transmission time of flow groups is a top

Manuscript received March 18, 2018; revised November 05, 2018; accepted January 31, 2020; approved by IEEE/ACM Transactions on Networking Editor Y. Guan. Date of publication February 19, 2020; date of current version April 16, 2020. This work was supported in part by the grant from the Research Grants Council of Hong Kong under Project CityU 11214316. *(Corresponding author: Yingying Cheng.)*

The authors are with the Department of Computer Science, City University of Hong Kong, Hong Kong (e-mail: yychengcs@gmail.com; csjia@cityu.edu.hk).

Digital Object Identifier 10.1109/TNET.2020.2971587

priority for cloud operators. However, existing studies on multipath forwarding do not consider the minimization of flow group transmission time. We propose a network-aware multipathing (NAMP) scheme to minimize the flow group transmission time by considering the heterogeneous bandwidth in data center networks.

It is challenging to allocate the limited routing rules among multiple flow groups to minimize their transmission time. First, it is difficult to jointly allocate routing rules in the flow tables of multiple switches to one flow group. The reason is that the rule allocation in an edge switch will affect the rule allocation in the following aggregation switch along the same path. The table fitting problems for multiple switches are correlated to each other. Second, it is difficult to jointly allocate routing rules to multiple flow groups. Since the flow groups share one flow table inside a switch, the rule allocation of one flow group will affect the rule allocation of the other flow groups. The table fitting problems for multiple flow groups are also correlated even inside one single switch. Third, it is hard to efficiently achieve the minimal transmission time for all flow groups simultaneously, because the rule allocation problem for multiple flow groups is a nonlinear integer program which usually does not have optimal solutions.

In this paper, we first formulated the network-aware multipathing problem for one flow group as a biconvex integer programming problem. The objective is minimizing the transmission time of the flow group under the constraint of flow table capacity. Then, we proposed an efficient LP-based alternate convex search solution by discovering the special structure of the problem after linear transformations. We further extended the solution to the case of multiple flow groups and formulate the multipathing problem as an integer programming problem. An LP-based solution is proposed through linear transformations to an LP problem. The proposed scheme, called NAMP, is efficient and easy to implement in real-world SDN controllers. Since NAMP requires only performing linear transformations and solving LPs, the running time is still linear to the problem size. We prototyped NAMP as an application in RYU controller using OpenFlow APIs and existing LP solvers.

We evaluated NAMP against three state-of-the-art schemes: Niagara, WCMP and ECMP in data center networks simulated by NS3 and RYU. Extensive experiments are performed for one-to-one communication, all-to-all communication and real-world data center traffic in data center networks with different size. Experimental results show that NAMP outperforms Niagara, WCMP and ECMP in terms of flow group completion time. For one-to-one traffic, NAMP reduces the flow group transmission time by up to 33%, 36%, 44.5% shorter than Niagara, WCMP and ECMP respectively. For all-to-all communication, NAMP reduces the transmission time by up to 18%, 25.6%, 45.4% shorter than Niagara, WCMP and ECMP respectively. For real-world data center traffic, NAMP reduces the transmission time by up to 45.4%, 50%, 60% shorter than Niagara, WCMP and ECMP respectively.

The rest of this paper is organized as follows. Section II describes the related work and Section III discusses the background and motivation of our problem. Section IV presents our system model and problem formulation. Section V shows the table fitting problem for one flow group, while Section VI discusses the solution for multiple flow groups. Sections VII and VIII describe the system implementation and performance evaluation. Finally we concluded the paper in Section IX.

II. Related Work

There have been extensive studies in load balancing and traffic engineering for multipath routing in data center networks by considering the limited flow table size.

DomainFlow [2] proposes a flow management model composed of two domains inside every OpenFlow switch to save flow table space. In Domain 1, a flow is matched with wild-carded matching table (WMT). In Domain 2, a flow is matched with exact matching table (EMT). The Domain 2 is used to match unicast, broadcast and multicast packets. If there is a lookup failure for a packet, it is handled by Domain 1. However, the flow management model does not discuss optimizing flow table usage for wild card rules.

Some follow-up studies aim at optimizing the network throughput by considering the constraint on flow table size. OFFICER [5] proposes a linear optimization model of the rule allocation problem in OpenFlow switches with relaxing routing policy. The objective is to maximize the throughput subject to the flow table constraints and endpoint policy constraints. The max-flow problem is proven to be NP-hard and a polynomial time heuristic is proposed as the solution. With the same objective, [6] models the limited flow table size by bounding the number of paths that can pass through each node in the network, which is called path-degree. Then, a bounded path-degree max-flow problem is formulated to maximize the network throughput, while satisfying the bandwidth capacities and the path-degree constraints. Reference [7] maximizes the network throughput by satisfying the flow table constraint and the controller processing capacity constraint. Reference [8] maximizes the network throughput by satisfying both the flow table constraint and the group table constraint. Reference [9] proposes to trade throughput with the memory constraints. The objective is to maximize the amount of traffic matching to the policies. On the other hand, [10] minimizes the rule space occupation for multiple unicast sessions while subjecting to the throughput constraint. Reference [11] minimizes energy consumption for a backbone network considering the throughput constraint and the flow table constraints.

A scalable load balancer based on multipath forwarding is proposed in [12] to distribute user requests to server replicas for online services. Each server replica has a weight, which represents the share of requests the replica should process. The load balancer distributes client traffic to the servers according to their weights. A set of wild card rules is generated in the load balancer to match the incoming requests based on the client IP addresses. The objective is to generate the minimal set of wild card rules that divide the entire client IP address space according to the load balancing weights. The partitioning algorithm computes the closest power of 2 to the sum of all replicas' weights and re-normalizes the weights accordingly. However, the load balancer can not be applied to split network flows in a data center with a large amount of switches.

WCMP [3] and Niagara [4] are two state-of-the-art studies on multipath routing subject to the flow table capacity. For a flow group, the controller firstly computes a set of ideal traffic splitting weights among multiple egress ports without concerning the limited multipath table size, by running max-flow min-cut algorithms. Then, both WCMP and Niagara propose to re-compute a set of new weights for the flow group by optimizing different objectives. WCMP optimizes two objectives: (1) maximize the possible reduction on the group size subject to a maximum oversubscription limit as the constraint; (2) minimize the maximum oversubscription with a constraint on the group size. The group size means the total number of routing rules allocated to the flow group in the multipath table. The maximum oversubscription is the maximal oversubscription value for all egress ports. The oversubscription value for a port is equal to its new weight divided by its ideal weight. On the other hand, Niagara aims at minimizing the traffic imbalance (i.e., the fraction of traffic sent to the "wrong" next-hops based on the ideal splitting weights) subject to the multipath table capacity.

Existing studies achieve the scalability of multipath routing by (i) maximizing the throughput subject to the flow table capacity or (ii) minimizing the usage of routing rules through reducing the multipathing weights. Our work differs from the previous works as we address the quality of service for flow groups in multipath routing and propose an efficient and implementable algorithm.

III. Background and Motivation

A. Multipath Forwarding in Switches

Today's data center network (DCN) employs multiple paths between thousands of servers to increase throughput. Equal Cost Multipath (ECMP) is a conventional load balancing solution that uniformly spreads the traffic of a flow group among multiple paths. The bandwidth among the multiple paths is uneven in DCNs. ECMP leads to substantial performance degradation because it ignores the bandwidth heterogeneity. To overcome ECMP's limitations, recent efforts focus on weighted multipathing to dynamically balance traffic in DCNs based on the changing network state. The weighted multipathing spreads the traffic of a flow group among multiple next-hops in a weighted distribution configured by the SDN controller.

In WCMP [3], switches implement multipath forwarding by creating *WCMP groups* in a multipath table. Each WCMP group is associated with a set of egress ports and their weights. The multipath table describes the egress ports for each WCMP group and replicates a port entry in proportion to its weight. For example in the right hand side of Fig. 1, the multipath table includes two WCMP groups. Suppose that there are totally 4 egress ports in the switch. The WCMP Group-2 has 12 entries starting from the 4th entry to the 15th entry. Among the 12 entries, the number of entries allocated to the 4 ports is 2, 2, 3 and 5 respectively. Thus, the egress port 1, 2, 3, 4 has weight 2, 2, 3 and 5 respectively. The traffic of WCMP Group-2 will be distributed to the 4 ports in proportion to their weights.

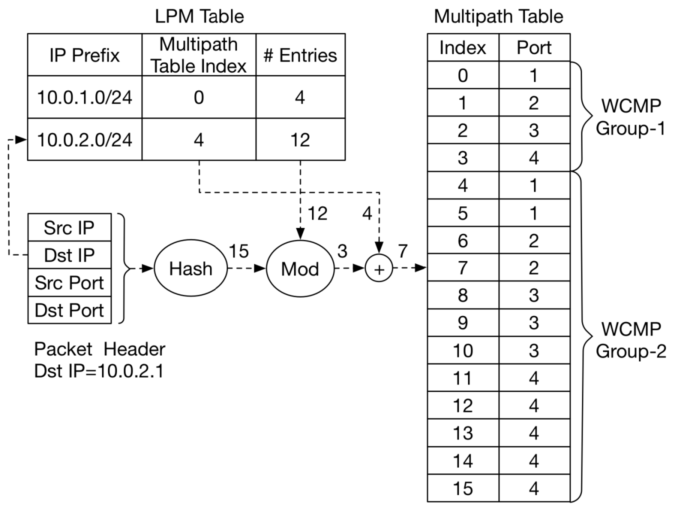


Fig. 1. Multipath forwarding example.

A flow is defined as a stream of packets that have the same source IP and the same destination IP. A flow group consists of flows that have the same source edge switch and the same destination edge switch. For instance, a flow group can be a set of flows that originate from the same subnet to another subnet. All the flow groups are matched against a Longest Prefix Match (LPM) table in switches, as it is shown in the left hand side of Fig. 1. Each row in the LPM table defines a flow group in the first column and the WCMP group information in the second and third column. For example, there are two rows in the LPM table in the left hand side of Fig. 1. The second row defines the flow group with destination IP prefix 10.0.2.0/24, which is associated with the WCMP group that starts from the 4th entry and includes the following 12 entries in the multipath table. In other words, the traffic matched with the second LPM rule is splitted among the 4 egress ports based on the weights defined by WCMP Group-2.

Fig. 1 shows the pipeline of forwarding a packet based on the LPM table and the multipath table. The left corner of Fig. 1 shows the header of an ingress packet. It is matched to the second rule in the LPM table. The packet will be mapped to the WCMP Group-2 that ranges from the 4th entry to the 15th entry in the multipath table. Next, the switch performs hashing over the packet header and generates a result of 15. The hashing result modulos the number of entries for the WCMP group, i.e., 15 mod 12, and obtains a result of 3. Then, the switch adds the modulo result to the group's base index 4 and obtains 7, which is the index of the table entry whose egress port the ingress packet will be forwarded to. The hash function is the total number of packets that have been matched with a routing rule defined in the LPM table. If a new ingress packet with a different header that also matches with the second LPM rule, the hashing result will be incremented to 16. Through this process, this new packet will be mapped to the 8th entry in the multipath table. Eventually, all the packets matched to WCMP group-2 will be evenly splitted to the 12 entries. Since the number of entries for each egress port is equal to its weight, traffic matched with WCMP group-2 is splitted among the port 1, 2, 3, 4 in the distribution of 2:2:3:5.

Implementing the above weighted traffic distribution can consume hundreds of routing rules for a single flow group. Since the overall number of routing rules in today's on-chip multipath table is very much limited, efficiently allocating the routing rules among competing flow groups is critical to the scalability of load balancing in DCNs.

B. Motivation

WCMP [3] and Niagara [4] are two state-of-the-art solutions considering the limited number of routing rules. Both of the solutions re-compute a set of new traffic splitting weights for flow groups based on a set of ideal weights that are initially computed by the SDN controller. WCMP [3] focuses on minimizing the maximum oversubscription of all egress ports in a switch, subject to the limited size of the multipath table in the switch. The oversubscription value of a port is defined as its new weight divided by its ideal weight. While Niagara aims at minimizing the overall traffic imbalance, which is defined as the fraction of traffic sent to the "wrong" next-hops based on the ideal splitting weights of multiple flow groups, subject to the limited number of routing rules.

WCMP focuses on minimizing the maximum oversubscription of all egress ports in a switch without violating the capacity of its multipath table. Niagara aims at minimizing the traffic shifted from the ideal distribution. The flow group transmission time, which indicates the quality of cloud services, can be sub-optimal under the two schemes. The reasons are in three aspects. 1) The flow group transmission time is determined by the traffic demand of the flow group and the link bandwidth. Both WCMP and Niagara assume a homogeneous bandwidth for all the links. However, it is common to have bandwidth heterogeneity among the egress links in one switch, because organizations often own multiple generations of hardware and virtualization techniques are getting popular in data centers [13]. It is possible that WCMP and Niagara distribute most of the traffic in a flow group to an egress link with small bandwidth, which becomes the bottleneck of the flow group transmission time. 2) The flow group transmission time consists of the transmission time of multiple consecutive hops on the same path. However, both WCMP and Niagara only consider allocating routing rules in a single hop/switch, which may not minimize the flow group transmission time. 3) The transmission time of a flow group is also determined by the other flow groups if multiple flow groups share the same switches. WCMP and Niagara do not consider simultaneously minimizing the transmission time of each flow group if there are multiple competing flow groups.

Next, we provide examples to show that WCMP and Niagara can lead to a performance degradation on the flow group transmission time. Let's consider a 2-stage Clos topology example illustrated in Fig. 2. There are three switches in Stage-1 that connect end hosts, and three switches in Stage-2 that connect the Stage-1 switches. Each Stage-1 switch connects to all the three Stage-2 switches. For example, S0, an Stage-1 switch, has three egress links that connect to S1, S2 and S3 respectively. There are three paths for every flow. Suppose that there is a group of flows with the destination

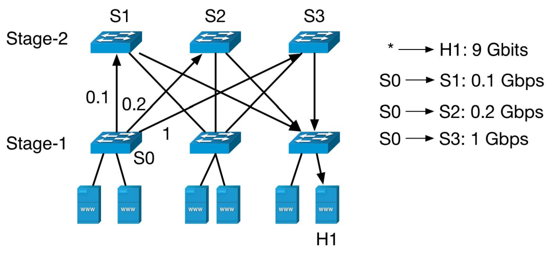


Fig. 2. 2-Stage clos network.

Index	Port
0	S1
1	S2
2	S2
3	S3
4	S3
5	S3

WCMP

Index	Port
0	S1
1	S2
2	S3
3	S3
4	S3
5	S3

Niagara

Index	Port
0	S2
1	S3
2	S3
3	S3
4	S3
5	S3

Our Method

Fig. 3. Table fitting results.

H1 arriving at S0. The total traffic amount for this flow group is 9 Gbits. Thus, S0 needs to distribute the 9 Gbits traffic to the next-hops S1, S2 and S3. Assume that the available bandwidth is 0.1 Gbps from S0 to S1, 0.2 Gbps from S0 to S2, 1 Gbps from S0 to S3. Obviously the ideal traffic splitting ratio among S1, S2 and S3 is 1:2:10, which requires totally 13 entries for this flow group.

Assume that there are only 6 entries available to this flow group. Each entry is associated with one egress port. The question is how to assign weights to the port S1, S2 and S3, with guarantee that the number of consumed entries does not exceed 6. The problem becomes computing the traffic splitting ratio for the flow group among S1, S2 and S3 to fit in the multipath table of S0. WCMP minimizes the maximum oversubscription, which ensures every port is not receiving too much extra traffic than it can serve. WCMP allocates 1, 2, 3 entries to S1, S2 and S3 respectively, and generates a splitting ratio of 1:2:3, as shown in the left table of Fig. 3. Thus, the amount of traffic distributed to each of the three paths from S0 to H1 is: 1.5 Gbits, 3 Gbits, 4.5 Gbits. Suppose the bandwidth of all other links other than (S0, S1), (S0, S2) and (S0, S3) is 10 Gbps. Since the transmission time for the flow group is equal to the maximal transmission time among the three paths, the transmission time for the flow group is 15.6 Sec.

Niagara minimizes the amount of traffic imbalance, which is the amount of traffic shifted away from the ideal distribution. The algorithm in Niagara allocates 1 entry to S1, 1 entry to S2, and 4 entries to S3, as shown in the middle of Fig. 3. The transmission time for the flow group under Niagara is equal to 15.3 Sec. However, if we consider minimizing the maximal path-level transmission time, the number of entries allocated to port S1, S2, S3 will be 0, 1, 5, which is described in the right hand side of Fig. 3. Subsequently, the transmission time for the flow group is equal to 9 Sec, which is much shorter than WCMP and Niagara.

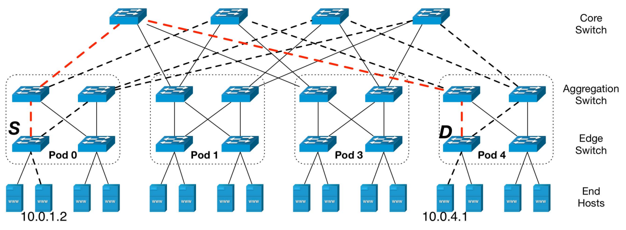


Fig. 4. Fat-tree topology with $N = 2$.

We will formulate the problem of network-aware table fitting problem in a fat-tree topology, which is a more representative topology than the 2-stage Clos topology. Our problem formulation is also applicable to other types of root-based data center network topologies. Note that we use the term “flow group" for the rest of the paper, rather than the WCMP group mentioned in the background section.

IV. System Model and Problem Formulation

Cloud data centers employ 3-tier architectures [14] to interconnect racks of servers. A canonical architecture is the fat-tree topology [15]. In a $(2N)$-ary fat-tree topology, the servers are grouped into $2N$ pods, where N is the number of edge/aggregation switches in each pod. Every edge switch is connected to all the N aggregation switches in the same pod. Each aggregation switch is connected to N different core switches in the network, where there are totally N^2 core switches. Note that no two aggregation switches within the same pod connect to the same core switch. Thus, there are N^2 equal-length paths available for every flow in the network. For example, Fig. 4 shows a fat-tree topology with 4 pods with $N = 2$. Assume that a flow is identified by the tuple (Src-IP, Dst-IP), where Src-IP represents the IP address of the source host, and Dst-IP represents the IP of the destination host.

A flow group consists of flows that traverse through the same source edge switch and the same destination edge switch. The source edge switch of a flow is the edge switch that connects to the source host. Similarly, the destination edge switch of a flow is the edge switch that connects to the destination host. For instance, a flow group can be a set of flows that their Src-IPs are in one subnet and their Dst-IPs are in another subnet. DCNs employ multiple equal-length paths and spread the traffic of one flow group among them to keep a high throughput.

Suppose that there is a flow group represented by (S, D, λ), where S/D represents the source/destination edge switch. Due to the prior work of traffic matrix estimation on data center networks, it is reasonable to assume that the controller knows the traffic demand of each flow group [16]. Let λ denote the traffic demand. As shown in Fig. 4, S is the first edge switch in Pod 0, and D is the first edge switch in Pod 4. λ is the aggregate traffic demand from the hosts connected with S to the hosts connected with D. The traffic of the flow group is split among the 4 paths from S to D. The controller determines the traffic splitting ratio among the egress ports in each switch. Let $P = \{p_{ij}, i = 1, \ldots, N; j = 1, \ldots, N\}$ denote the set of N^2 paths. Each path p_{ij} consists of 4 links. The 1st link is the source edge switch S connecting to the ith aggregation switches. The 2nd link is the ith aggregation switch connecting to its jth core switches. The 3rd link is the core switch connecting to an aggregation switch, and the 4th link is the aggregation switch connecting to the destination switch D. As it is shown in Fig. 4, the path in red shows the 4 links from S to D. Let $b_{ij}^1, b_{ij}^2, b_{ij}^3, b_{ij}^4$ respectively denote the bandwidth of the four links on the path p_{ij}.

In order to determine the traffic distribution over the N^2 paths, the controller must solve the table fitting problem in the source edge switch S and its following N aggregation switches. We need to fit the flow group into the limited table by re-calculating the weights of all egress ports. This is achieved by allocating the multipath table entries to each egress port. Let R_s denote the number of available entries in the source edge switch S for this flow group. R_i denotes the number of available entries in the ith aggregation switch next to S. R_s and R_i are given constants. Let x_i $(i = 1, \ldots, N)$ denote the number of entries allocated to the ith egress port in S, $\sum_i x_i = R_s$, and y_{ij} $(i, j = 1, \ldots, N)$ denotes the number of entries allocated to the jth egress ports in the ith aggregation switch, $\sum_j y_{ij} = R_i$. x_i and y_{ij} are variables in our table fitting problem.

The amount of traffic distributed to the 1st link in the path p_{ij}, denoted by λ_{ij}^1, is expressed as follows: $\lambda_{ij}^1 = \lambda \frac{x_i}{R_s}$, where x_i/R_s means the percentage of traffic distributed to the ith egress port in S. The amount of traffic distributed to the 2nd link at the path p_{ij}, denoted by λ_{ij}^2, is computed as follows: $\lambda_{ij}^2 = \lambda_{ij}^1 \frac{y_{ij}}{R_i} = \lambda \frac{x_i}{R_s} \frac{y_{ij}}{R_i}$, where y_{ij}/R_i is the percentage of traffic distributed to the jth egress port in the ith aggregation switch. On the path p_{ij}, the amount of traffic on the 3rd and 4th link, denoted by λ_{ij}^3 and λ_{ij}^4 respectively, is equal to that on the 2nd link. That is $\lambda_{ij}^2 = \lambda_{ij}^3 = \lambda_{ij}^4$. Since the bandwidth on each of the 4 links on the path p_{ij} is denoted by $\{b_{ij}^1, b_{ij}^2, b_{ij}^3, b_{ij}^4\}$, we can compute the transmission time of the flow group on the path p_{ij}, which is denoted by t_{ij}. t_{ij} is equal to the sum of the transmission time on the 4 links as follows.

$$\begin{aligned} t_{ij} &= \frac{\lambda_{ij}^1}{b_{ij}^1} + \frac{\lambda_{ij}^2}{b_{ij}^2} + \frac{\lambda_{ij}^3}{b_{ij}^3} + \frac{\lambda_{ij}^4}{b_{ij}^4} \\ &= \frac{\lambda}{R_s b_{ij}^1} x_i + \frac{\lambda}{R_s R_i} (\frac{1}{b_{ij}^2} + \frac{1}{b_{ij}^3} + \frac{1}{b_{ij}^4}) x_i y_{ij}. \end{aligned} \tag{1}$$

Let t denote the transmission time for the flow group among all the N^2 paths. The flow group transmission time is determined by the longest time among all the paths. The transmission time for the flow group t is expressed as follows:

$$t = \max_{ij} t_{ij} = \max_{ij} (\frac{\lambda}{R_s b_{ij}^1} x_i + \frac{\lambda}{R_s R_i} (\frac{1}{b_{ij}^2} + \frac{1}{b_{ij}^3} + \frac{1}{b_{ij}^4}) x_i y_{ij}). \tag{2}$$

Our formulation of the flow group transmission time t can also be applied to the other data center topologies apart from the fat-tree. Today’s data center topologies are all in tree-based hierarchical structure with three tiers including the edge-tier, aggregation tier and core-tier switches [14]. Typical examples are VL2 [17] and 3-stage Clos network [18]. Our formulation of path-level transmission time, i.e., t_{ij} in Eq. (1), is applicable to all the three-tier data center topologies. The key reason is that our variable x_i and y_{ij} can represent the rule allocation for every edge switch and every aggregation switch respectively for all the three-tier data center networks. Notice that since our problem formulation can be generalized to all data center topologies, our solution of minimizing t is also applicable to all the topologies. Next, we describe our algorithm of minimzing t for a single flow group under the limited multipath table size. Then, we discuss the table fitting problem for multiple flow groups.

V. Table Fitting for A Flow Group

In this section, the table fitting problem for a single flow group is formally defined as a bi-convex integer programming problem. Then, we propose an LP-based Alternate Convex Search (LACS) framework to obtain the optimum. The LACS employs a convex search outside two linear programs, which is efficient and easy to implement. We elaborate the problem definition and the LACS in the following subsections.

A. Problem Definition

The table fitting problem focuses on determining the rule allocation variable x_i and y_{ij} such that the flow group transmission time t is minimized. The table fitting problem for a single flow group can be formulated as follows:

$$\min \max_{ij} \left(\frac{\lambda}{R_s b^1_{ij}} x_i + \frac{\lambda}{R_s R_i} (\frac{1}{b^2_{ij}} + \frac{1}{b^3_{ij}} + \frac{1}{b^4_{ij}}) x_i y_{ij} \right) \quad (3)$$

$$\text{s.t.} \sum\nolimits_i x_i = R_s, \quad (4)$$

$$\sum\nolimits_j y_{ij} = R_i, \quad \forall i, \quad (5)$$

$$x_i, y_{ij} \in \mathbb{Z}_{\geq 0}, \quad \forall i, j. \quad (6)$$

This is a bi-convex integer programming problem, which minimizes the sum of a convex function in integer variable $\mathbf{x}$, a convex function in integer variable $\mathbf{y}$ and a bilinear term in $\mathbf{x}$ and $\mathbf{y}$ over a closed set [19]. Since all the R_s, λ and b_{ij} are constants, the objective function in (3) is a sum of a linear function in x_i and a bilinear term in x_i and y_{ij}, which is matched with the definition of bi-convex integer programming proble We propose an LP-based Alternate Convex Search (LACS) method to obtain the optimum. The basic idea of LACS is alternatively fixing one type of variables and solving an LP regarding the other type of variables. We first fix the values of $\mathbf{y} = \{y_{ij}\}$ in (3). As it is shown in line-1 and line-4 of Algorithm 1, The initial values of $\mathbf{y}$ are randomly selected. Let e_{ij} denote the coefficient of x_i in (3). Let c_{ij} denote the constant term in (3). e_{ij} and c_{ij} are expressed as follows:

$$e_{ij} = \frac{\lambda}{R_s b^1_{ij}} + \frac{\lambda y_{ij}}{R_s R_i} (\frac{1}{b^2_{ij}} + \frac{1}{b^3_{ij}} + \frac{1}{b^4_{ij}}), \quad c_{ij} = 0. \quad (7)$$

We solve the following problem to obtain $\mathbf{x} = \{x_i\}$.

$$\min \max\nolimits_{ij} (e_{ij} x_i + c_{ij}) \quad (8)$$

$$\text{s.t.} \sum\nolimits_i x_i = R_s, \quad (9)$$

$$x_i \in \mathbb{Z}_{\geq 0}, \quad \forall i, \quad (10)$$

where only x_i is the variable, e_{ij} and c_{ij} are constants.

Algorithm 1 LP-based Alternate Convex Search Framework

```
Input: $\lambda$, $\{b_{ij}\}$, $R_s$, $\{R_i\}$.
Output: The optimum solution of $x_i$ and $y_{ij}$.
1: Choose an arbitrary starting point $z = (\mathbf{x}_0, \mathbf{y}_0)$;
2: Set $r \leftarrow 0$;
3: while $t$ in (2) does not reach minimum do
4:     Fix the values of $\mathbf{y}$ to $\mathbf{y}_r$, solve (8) to obtain $\mathbf{x}^*$;
5:     Fix the values of $\mathbf{x}$ to $\mathbf{x}^*$, solve (12) to obtain $\mathbf{y}^*$;
6:     Set $r \leftarrow r + 1$, $\mathbf{x}_r \leftarrow \mathbf{x}^*$, $\mathbf{y}_r \leftarrow \mathbf{y}^*$, $z_r = (\mathbf{x}_r, \mathbf{y}_r)$;
7:     Compute $t$ based on $\mathbf{x}_r, \mathbf{y}_r$;
8: end while
```

After we obtain the value of $\mathbf{x} = \{x_i\}$, we fix the value of $\mathbf{x}$ in the original problem (3), as it is shown in line-5 of Algorithm 1. Let e_{ij} denote the coefficient of y_{ij} and c_{ij} denotes the constant term in (3). We have

$$e_{ij} = \frac{\lambda x_i}{R_s R_i} (\frac{1}{b^2_{ij}} + \frac{1}{b^3_{ij}} + \frac{1}{b^4_{ij}}), \quad c_{ij} = \frac{\lambda x_i}{R_s b^1_{ij}}. \quad (11)$$

The values of the variables $\mathbf{y}$ can be obtained by solving the following optimization problem:

$$\min \max\nolimits_{ij} e_{ij} y_{ij} + c_{ij} \quad (12)$$

$$\text{s.t.} \sum\nolimits_j y_{ij} = R_i, \quad \forall i, \quad (13)$$

$$y_{ij} \in \mathbb{Z}_{\geq 0} \quad \forall i. \quad (14)$$

B. LP-Based Alternate Convex Search Framework

The LP-based Alternate Convex Search (LACS) framework is described in Algorithm 1. We first set initial values for $\mathbf{x}$ and $\mathbf{y}$ in line-1. As it is described in line-3 of Algorithm 1, the LACS search terminates when t defined in (2) reaches a minimum point. By minimum, we means that the value of t is smaller than that both the value in the last iteration and in the next iteration. In other words, we stop the search if $t_r < t_{r-1}$ and $t_r < t_{r+1}$, where r is the number of iteration.

The problems in (8) and (12) are the same form of integer programming. Thus, both of them can be solved by the same optimization technique. Next, we describe our solution of solving the problem in (12) with variables $y_{ij}, \forall i, j = 1, \ldots, N$. Problem (8) can be solved using the same method. We will show that the problem (12) can be transformed into an equivalent lexicographical minimization problem. The optimum of the lexicographical problem can be obtained by solving a special type of nonlinear program with a separable convex objective function and a totally unimodular coefficient matrix. The optimum of the nonlinear program can be further obtained by solving an equivalent linear programming (LP), which can be solved using existing LP solvers such as GLPK.

C. LP-Based Solution

Let t_{ij} denote the objective function in (12):

$$t_{ij} = e_{ij} y_{ij} + c_{ij}. \quad (15)$$

The objective in (12) is equivalent to $\min \max_{ij} t_{ij}$, which can be transformed to a lexicographical minimization problem.

We first introduce some terminologies on the lexicographical order. Suppose $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are N-dimentional vectors with their elements are sorted in non-increasing order. We say that $\boldsymbol{\alpha}$ is lexicographically smaller than $\boldsymbol{\beta}$, denoted by $\boldsymbol{\alpha} \prec \boldsymbol{\beta}$, if the first non-zero item of $\boldsymbol{\alpha} - \boldsymbol{\beta}$ is negative. Similarly, $\boldsymbol{\alpha}$ is lexicographically no greater than $\boldsymbol{\beta}$, denoted by $\boldsymbol{\alpha} \preceq \boldsymbol{\beta}$, if $\boldsymbol{\alpha} \prec \boldsymbol{\beta}$ or $\boldsymbol{\alpha} = \boldsymbol{\beta}$.

Let $\mathbf{t}$ denote the vector including the transmission time on all the N^2 paths, i.e., $\mathbf{t} = (t_1, \ldots, t_{ij}, \ldots, t_{NN})$. We sort the elements in $\mathbf{t}$ in non-increasing order. Minimizing the maximal element in $\mathbf{t}$, i.e., $\min \max_{ij} t_{ij}$, is equivalent to a lexicographical minimization problem that finds a vector that is lexicographically minimal over all the feasible vectors of $\mathbf{t}$. The lexicographical minimization problem is defined as follows:

$$\begin{aligned} &\text{lexmin}_{\mathbf{y}}\ \mathbf{t} = (t_1, \ldots, t_{ij}, \ldots, t_{NN}) \\ &\text{s.t. Constraints (13) and (14),} \end{aligned} \tag{16}$$

where t_{ij} is given in (15). It is difficult to solve the lexicographical minimization problem in (16), due to its non-convex nature. We propose to minimize a convex function that can produce the same optimal solution in (16). We first collect the set of all constants in the problem (16), which is $\{e_{ij}, c_{ij} | i, j = 1, \ldots, N\}$. Notice that e_{ij} and c_{ij} are non-negative non-integer numbers. Let M denote a number such that $M \times e_{ij}$ and $M \times c_{ij}$ are integers for $i = 1, \ldots, N$ and $j = 1, \ldots, N$. The way to compute M is described as follow. 1) We convert all the e_{ij} and c_{ij} to their fractional form. For example, the fractional form of 0.5 is $\frac{1}{2}$. 2) M is equal to the least common multiple of all the denominators in the fractional forms of e_{ij} and c_{ij}. For instance, 30 is the least common multiple of all the denominators in $\{\frac{1}{2}, \frac{2}{3}, \frac{3}{5}\}$ and hence M is equal to 30. We then define a convex function as follows:

$$g(\mathbf{t}) = \sum_{i=1}^{N} \sum_{j=1}^{N} N^{2Mt_{ij}}, \tag{17}$$

where $\mathbf{t}$ is a vector with elements sorted in non-decreasing order. Notice that $g(\mathbf{t})$ is an exponential function, which is convex. Minimizing the function $g(\mathbf{t})$ can generate the lexicographically minimal vector of $\mathbf{t}$, which is the optimal solution to problem (16). Thus, we obtain the following theorem.

Theorem 1: Minimizing $g(\mathbf{t})$ generates the lexicographical minimum of $\mathbf{t}$.

Proof: The proof is presented in the Appendix. □

Our target becomes minimizing $g(\mathbf{t})$ while subjecting to the constraints (13) and (14), which is a nonlinear integer program with a separable convex objective function. There is a special class of nonlinear integer program whose solution can be obtained by solving a single linear program [20]. The nonlinear integer program must satisfy two conditions: a separable convex objective function and a totally unimodular constraint matrix. We then explain that the coefficient matrix of the constraint (13) is totally unimodular. A matrix is totally unimodular if every square submatrix has a determinant equal to 1, -1, or 0 [21]. This implies that all entries in the matrix are 1, -1, or 0. We can see that all the coefficients of y_{ij} in the constraint (13) are equal to 1. Thus, the coefficient matrix of the constraint (13) is totally unimodular.

The optimum of minimizing $g(\mathbf{t})$ is equal to the optimum of a linear program. The linear program is obtained through the following series of transformations. Let

$$h(y_{ij}) = N^{2Mt_{ij}} = N^{2M(e_{ij}y_{ij} + c_{ij})}. \tag{18}$$

Note that $g(\mathbf{t}) = \sum_{i=1}^{N} \sum_{j=1}^{N} h(y_{ij})$. $h(y_{ij})$ can be transformed into the following linear programs by introducing another variable γ_{ij}^k, which is a positive continuous variable and $k = 1, \ldots, R_i$.

$$h(y_{ij}) = \sum_{k=1}^{R_i} h(k)\gamma_{ij}^k \tag{19}$$

$$\text{s.t.} \sum_k k\gamma_{ij}^k = y_{ij}, \quad \forall i, j, \tag{20}$$

$$\sum_k \gamma_{ij}^k = 1, \quad \forall i, j, \tag{21}$$

$$\gamma_{ij}^k \in \mathbb{R}_{\geq 0}, \quad \forall i, j, k, \tag{22}$$

where R_i is the number of rules available in the ith aggregation switch. γ_{ij}^k is a continuous variable that ≥ 0, $i = 1, \ldots, N$ and $j = 1, \ldots, R$.

The original integer programming problem in (12) is solved by minimizing $g(\mathbf{t})$, which is equivalent to the following linear mix-integer program:

$$\begin{aligned} &\min \sum_{i=1}^{N} \sum_{j=1}^{N} h(y_{ij}) = \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{k=1}^{R_i} h(k)\gamma_{ij}^k \\ &\text{s.t. Constraints (13), (14), (20), (21), (22),} \end{aligned} \tag{23}$$

where γ_{ij}^k is a continuous variable and y_{ij} is an integer variable. If we relax the integer constraint of $y_{ij} \in \mathbb{Z}_{\geq 0}$ into $y_{ij} \in \mathbb{R}_{\geq 0}$, the above mix-integer program (23) becomes a linear program (LP), which is described as follows:

$$\begin{aligned} &\min \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{k=1}^{R_i} N^{2M(e_{ij}k + c_{ij})}\gamma_{ij}^k \\ &\text{s.t. Constraints (13), (20), (21),} \end{aligned} \tag{24}$$

$$y_{ij}, \gamma_{ij}^k \in \mathbb{R}_{\geq 0}, \quad \forall i, j, k. \tag{25}$$

It has been proven in [20] that the optimal values of y_{ij} in the above LP (24) are integral, and they are overlapped with the optimal solutions in the original problem (12). Note that there are totally $N \sum_i R_i + N^2$ variables in the LP (24). We can obtain the optimum of $\mathbf{y}$ efficiently by solving the LP using existing LP solvers such as GLPK.

The proposed optimization solution above for solving the problem (12) can also be applied to the problem (8) to obtain the optimum of $\mathbf{x}$. Computing the optimum of $\mathbf{x}$ in the problem (8) is equivalent to solving the following LP problem.

$$\min \sum_{i=1}^{N} \sum_{k=1}^{R_s} N^{M(e_i k)}\gamma_i^k \tag{26}$$

$$\text{s.t.} \sum_k k\gamma_i^k = x_i, \quad \forall i, \tag{27}$$

$$\sum_k \gamma_i^k = 1, \quad \forall i, \tag{28}$$

$$\sum_i x_i = R_s, \tag{29}$$

$$x_i, \gamma_i^k \in \mathbb{R}_{\geq 0}, \quad \forall i, k. \tag{30}$$

In the above formulation, x_i and λ_i^k are continuous variables that are greater and equal to 0. $e_i = \max_j e_{ij}$ for each $i = 1, \ldots, N$, where e_{ij} is defined in (7). M is the minimal number that Me_i is an integer for every $i = 1, \ldots, N$.

VI. Table Fitting for Multiple Flow Groups

The table fitting problem for one flow group requires the number of entries, i.e., R_s and R_i, are given constants. In this section, we discuss how do we obtain the number of entries for each flow group. For multiple flow groups, they share the multipath table in the same set of switches. The table fitting problem aims at allocating the table entries in the switches to the flow groups. We solve the table fitting problem for each individual switch separately. Next, we describe the table fitting for multiple flow groups in one switch. The key is to compute the number of multipath table entries allocated to each flow group in the switch. We present the problem formulation and propose an LP-based solution, which apply the same technique proposed in the last section.

Suppose that there are F flow groups denoted by the set $\{(S_f, D_f, d_f), f = 1, \ldots, F\}$, where S_f represents the source edge switch of the fth flow group. D_f represents the destination edge switch of the fth flow group. λ_f denotes the traffic demand of the fth flow group. Assume that these F flows groups compete for the limited routing rules in the multipath table inside a switch. Let R denote the total number of table entries available in the switch. Let x_f denote the number of entries allocated to the fth flow group, where $f = 1, \ldots, F$ and $\sum_f x_f = R$. We target at computing the variable x_f in this section.

We propose to allocate table entries to the flow groups in proportion to their traffic demand in the switch, which can be formulated as follows:

$$\min \max_f \frac{x_f}{\lambda_f} \tag{31}$$

$$\text{s.t.} \sum_k x_f = R, \quad \forall f = 1, \ldots, F, \tag{32}$$

$$x_f \in \mathbb{Z}_{\geq 1}. \quad \forall f. \tag{33}$$

Since each flow group must occupy at least one entry in the multipath table, $x_f \geq 1$. Our goal is to determine x_f in proportion to the traffic demand λ_f, which means the ratio of $\frac{x_f}{\lambda_f}$ should be equal to 1 for every f. However, x_f is a discrete integer whereas λ_f is a non-integer number, which means the ratio of $\frac{x_f}{\lambda_f}$ cannot be equal to 1. As a compromise, we make the ratio of $\frac{x_f}{\lambda_f}$ as close as possible to each other for all flow groups. Thus, the objective becomes minimizing the maximal ratio of $\frac{x_f}{\lambda_f}$, which is presented in (32).

The problem in (32) is an integer programming problem, whose optimum cannot be solved directly. Notice that it can be solved by the same optimization technique proposed in the last section for the problem (12). We first transform the problem in (32) into a lexicographical minimization problem. Then, the lexicographical minimization problem is transformed into an equivalent LP problem, which can be solved efficiently.

Let w_f denote the objective function of (32), i.e., $w_f = \frac{x_f}{\lambda_f}$. The integer program in (32) can be transformed into the following lexicographical minimization problem:

$$\begin{aligned} &\text{lexmin}_{x_f} \mathbf{w} = (w_1, \ldots, w_F)_{sorted} \\ &\text{s.t. Constraints (32) and (33).} \end{aligned} \tag{34}$$

Since the constraint (32) and (33) satisfy the totally unimodular property, the lexicographical minimization problem in (34) can be transformed into a nonlinear integer program defined as follows:

$$\begin{aligned} &\min \sum_{f=1}^{F} F^{Mw_f} \\ &\text{s.t. Constraints (32) and (33),} \end{aligned} \tag{35}$$

where M is the minimal number such that M/λ_f is an integer for every $f = 1, \ldots, F$. The nonlinear program in (35) can be further transformed to an LP problem by introducing another variable γ_{fl}. The γ-transformation derives the following LP problem:

$$\min \sum_{f=1}^{F} \sum_{l=1}^{R} F^{Ml/\lambda_f} \gamma_{fl} \tag{36}$$

$$\text{s.t.} \sum_l^R l\gamma_{fl} = x_f, \quad \forall f, \tag{37}$$

$$\sum_l \gamma_{fl} = 1, \quad \forall f, \tag{38}$$

$$\sum_f x_f = R, \tag{39}$$

$$x_f \in \mathbb{R}_{\geq 1}, \quad \gamma_{fl} \in \mathbb{R}_{\geq 0}, \quad \forall f, l, \tag{40}$$

where the constraint (37) and (38) are the γ-related constraints in the γ-transformation. γ_{fl} is an continuous variable that ≥ 0. The constraint (39) is the original constraint on the variable x_f, which is a continuous variable that ≥ 1 by relaxing the integer constraint in the original problem (33). The optimum solution of x_f in (36) is integral and is also equal to the optimum solution of the original problem in (32). Through these transformations, the optimum solution to the table fitting problem for multiple flow groups can be obtained by solving an LP efficiently using existing LP solvers.

In summary, for multiple flow groups, we first allocate multipath table entries in switches to the flow groups using the solution in this section. The table entry allocation results derives the value of R_s and R_i for each flow group. Then, we can further allocate R_s and R_i entries to egress ports in each switch for each flow group. This is achieved by using the proposed solution in Section V, which optimizes the flow group transmission time.

VII. System Implementation

We implemented our proposed multipathing solution, called NAMP, as an application in the RYU OpenFlow

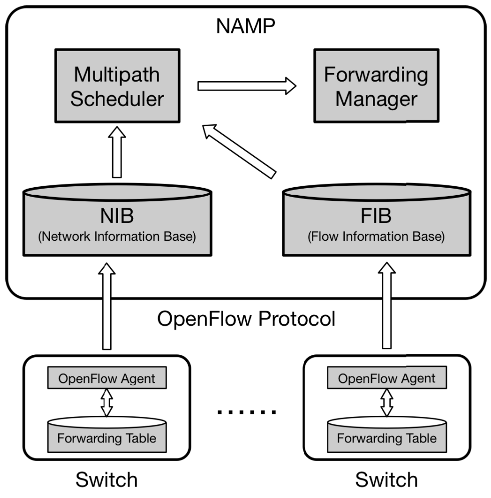


Fig. 5. NAMP architecture.

Controller [22]. We assume a reliable control channel between the network controller and the switches, e.g., a dedicated physical or logical control network. NAMP works in an event-triggered online basis. The controller does not modify the rule allocation of existing flow groups when handling new flow groups. The system architecture of NAMP is shown in Fig. 5. NAMP consists of four modules: NIB (Network Information Base), FIB (Flow Information Base), Multipath Scheduler and Forwarding Manager. Next, we describe the function of each module and their relations.

NIB: The Network Information Base discovers the network topology, link bandwidth and multipath tables in the switches. The OpenFlow switches periodically send liveness packets to the controller, so that the NIB keeps the node information. Each switch also sends adjacency updates to the controller, so that the NIB obtains the link information. The NIB obtains the access information of all the hosts by receiving the ARP packets from edge switches. For the link bandwidth, the controller periodically polls bandwidth of egress ports from every switch. The controller also polls the multipath table information from every switch to obtain the capacity of their multipath tables. The NIB provides network topology, link bandwidth information and multipath table capacities for the multipath scheduler to calculate traffic splitting ratios.

FIB: The Flow Information Base keeps track of all the ongoing traffic flows in the network. The controller periodically polls every switch for the flow table information. Each entry in the flow table describes the information of a live flow, which includes its source, destination IP prefix, the number of matches packets and the associated egress port. After polling flow tables from all switches, the FIB then identifies every single flow in the network. These flows are grouped into flow groups based on their source edge switches and destination edge switches. The FIB provides the information of all the flow groups to the multipath scheduler. Suppose that sthe traffic demand of each flow group is known to the controller as an input based on network measurement.

Multipath Scheduler: The Multipath Scheduler implements the proposed table fitting solution, which calculates the traffic splitting ratio for each flow group and its rule allocation in each related switch. The scheduler re-calculates traffic distributions on a periodical basis. The multipath scheduler obtains the network topology, link bandwidth, multipath table capacities and flow groups information from the NIB and FIB. Then, the multipath scheduler calls the proposed rule allocation algorithm and returns the rule allocation result for each live flow group. The rule allocation result includes the number of rules associated with each egress port in each switch for each flow group. The multipath scheduler calls the forwarding manager to implement the computed rule allocation results for each flow groups in all the related switches. The forwarding manager modifies the flow table entries in the switches based on the rule allocation results.

Forwarding Manager: The Forwarding Manager sends Flow Group Modification messages to switches based on the rule allocation results returned from the multipath scheduler. For example, port-1, port-2 and port-3 are allocated to 2, 3, 4 rules respectively for flow-group-1 in switch-A. Then, the forwarding manager creates 2 rules for flow-group-1 to port-1, 3 rules for flow-group-1 to port-2, 4 rules for flow-group-1 to port-3. The created flow rules are packed in OpenFlow FlowMod messages by the forwarding manager and are sent to the related switches. The switch will hash over the number of flow rules associated with the flow group to determine one matched flow rule. All the flow rules matched to one flow group expire once the flow group completes its transmission. Thus, the flow rules belong to two different flow groups will not interfere with each other.

VIII. Performance Evaluation

We implemented NAMP as an application in RYU controller and evaluate the performance in NS3 [23], which is a commonly adopted virtual OpenFlow network simulator. All the switches in the NS3 connects to an external RYU controller, which is running the NAMP application. We evaluated the TCP performance of NAMP against Niagara, WCMP and ECMP for different scale of fat-tree topologies: $N = 2, 3, 4$. Note that N is equal to the number of edge switches and also the number of hosts connected to each edge switch. $2N$ is equal to the number of pods in the network. There are totally $2N^3$ hosts in the network. There are totally 16, 54 and 128 hosts respectively for the fat-tree network with $N = 2, 3, 4$. Based on the number of hosts, the fat-tree network with $N = 2, 3, 4$ can be viewed as a small-scale, medium-scale and large-scale network respectively. In each of the three networks, every switch has the same number of multipath rules, which is specified as an input of the experiments. For the multipath table capacities, the number of available rules in the multipath table is set to be 200, 2000, 4000 for every switch in the network with $N = 2, 3, 4$ respectively.

We simulated three traffic patterns, which are one-to-one traffic, all-to-all traffic and real-world data center traffic from [24]. We measured the flow group transmission time (FGT for short) in NAMP and the state-of-the-art multipathing mechanisms: WCMP, Niagara and ECMP, under different network size and different traffic patterns. A flow group consists of multiple TCP flows traversing the same source edge

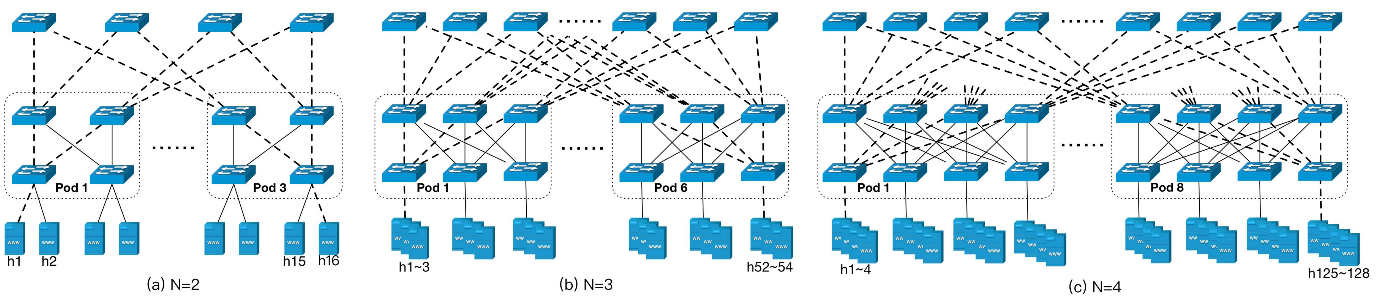


Fig. 6. Experimental topologies.

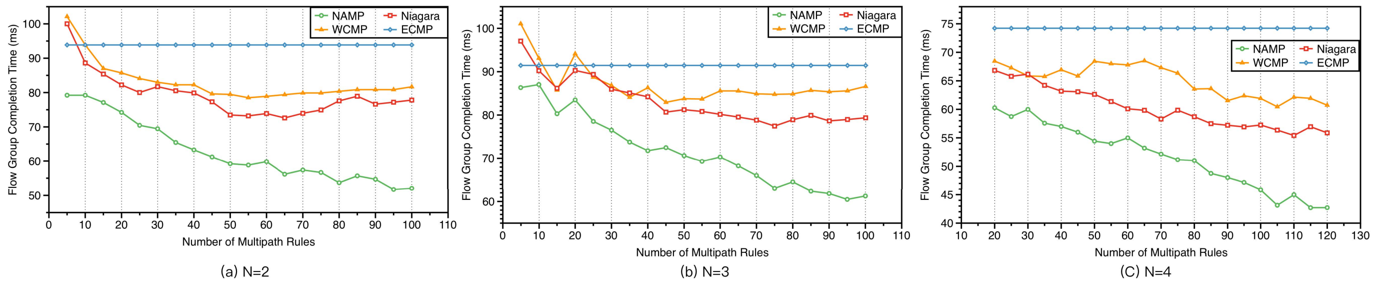


Fig. 7. One-to-one traffic.

switch and the same destination edge switch. The performance comparison results are summarized as follows:

- One-to-one traffic pattern generates a single flow group. Among all the three networks, NAMP reduces the FGT by up to 33% shorter than Niagara, up to 36% shorter than WCMP, and up to 44.5% shorter than ECMP.
- All-to-all communication generates multiple flow groups with identical size. Among all the three networks, NAMP reduces the FGT by up to 18% shorter than Niagara, up to 25.6% shorter than WCMP, and up to 45.4% shorter than ECMP.
- Real-world data center traffic generates multiple flow groups with size in lognormal distribution [24]. Among all the three networks, NAMP reduces the FGT by up to 45.4% shorter than Niagara, up to 50% shorter than WCMP, and up to 60% shorter than ECMP.

Next, we separately presented the experiment details and performance analysis under one-to-one traffic, all-to-all traffic and real-world DCN traffic.

A. *One-to-One Traffic*

We generated traffic belonging to a single flow group originating from one edge switch to another edge switch in the network. The flow group consists of 100 flows, and each flow is an unique TCP connection initiated by one client and one server. In each connection, the client sends 1M Bytes data to the server. All the 100 flows starts at the same time. We evaluated the flow group transmission time (FGT) in different fat-tree topologies with N equal to 2, 3 and 4. The three topologies are shown in Fig. 6. For each network, each link has heterogeneous bandwidth, which is randomly generated between 1 Gbps and 10 Gbps. In each of the three network, each of the N hosts connected with the first edge switch sends data to all the hosts connected with the last edge switch. For example, in Fig. 6(a), h1 and h2 are clients, h15 and h16 are servers. In the topology shown in Fig. 6(b), h1, h2, h3 are clients and h52, h53, h54 are servers. In the topology shown in Fig. 6(c), h1, h2, h3, h4 are clients and h125, h126, h127, h128 are servers. Each client/server launches the same number of flows simultaneously. For example, in the topology of Fig. 6(a), h1 transmits 25 flows to h15 and 25 flows to h16. h2 also transmits 25 flows to h15/h16. Each flow has identical size of 1Mbytes. The FGT (flow group transmission time) is equal to the total transmission time of the 100 flows.

In order to measure the transmission time of a flow, both the client and the server are started with the specified flow size. The client and server first establish a connection. Then, the client marks the start time and sends a flow of the specified size. Once receiving the specified number of bytes, the server sends back a single-byte token. When receiving the token, the client marks the end time. The client computes the flow transmission time as the difference between the start and end tup toime. The clients returns the flow transmission time after the connection completes. The FGT is equal to the difference between the time the first flow starts and the time the last flow ends.

In each of the three topologies, we set the number of rules in the multipath table to be equal in every switch. We observed the relation between the FGT and the number of multipath rules under the four multipath forwarding schemes. Fig. 7 shows the results on the FGT in the three topologies. Fig. 7(a) illustrates the performance comparison in the fat-tree network with $N = 2$. Fig. 7(b) and Fig. 7(c) presents the performance in the fat-tree network with $N = 3$ and $N = 4$ respectively. The X-axis represents the number of multipath rules in every switch. The Y-axis shows the FGT in milliseconds. The FGT under ECMP keeps to be 93.86 ms regardless of the increase

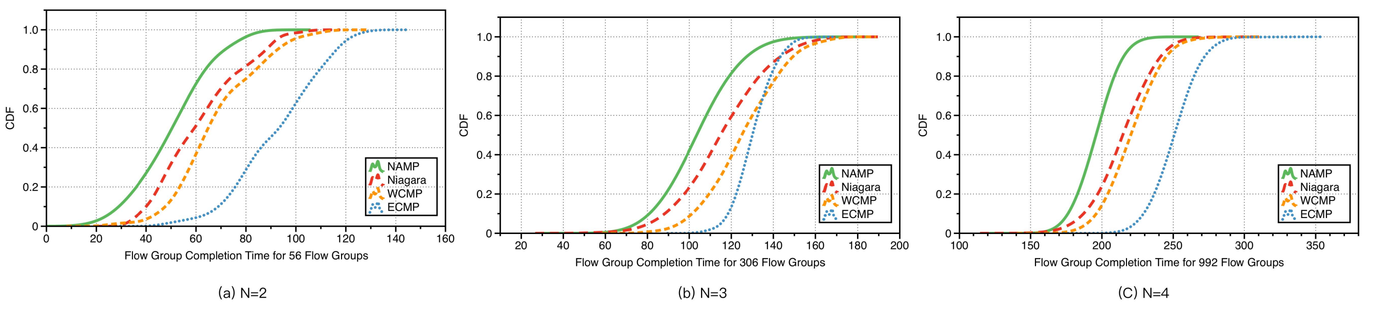


Fig. 8. All-to-all traffic.

of multipath rules size. That is because ECMP always equally splits all the ingress traffic to all the uplinks without considering the multipath table limit. By considering the multipath table size, WCMP, Niagara and our NAMP shows decreasing FGT as the number of multipath rules increases. When there are large number of multipath rules, which is larger than 100, the FGT converges for WCMP, Niagara and NAMP.

Fig. 7(a) shows the performance comparison between the four multipath schemes in the fat-tree topology with $N = 2$. Our NAMP method can improve up to 44.5% shorter FGT than ECMP, up to 36% shorter FGT than WCMP, up to 33% shorter FGT than Niagara. Fig. 7(b) shows the FGT in the fat-tree network with $N = 3$, NAMP generates up to 29.3% shorter FGT than WCMP, up to 22.8% shorter FGT than Niagara, up to 33% shorter FGT than ECMP. Fig. 7(c) presents the performance in the network with $N = 4$, NAMP improves up to 28.7%, 23.5% and 42.4% shorter FGT than WCMP, Niagara and ECMP respectively. By summarizing all the experiments for one-to-one communications among different scales of networks, NAMP reduces the FGT of flow groups by up to 44.5% shorter than ECMP, up to 36% shorter than WCMP, and up to 33% shorter than Niagara.

B. All-to-All Traffic

We then compared the performance of the four schemes in terms of all-to-all communication, which indicates that each host sends data to all the other hosts in the network. Each host sends flows to every other host in the network. In the fat-tree topologies illustrated in Fig. 6, every host simultaneously sends 5 flows to every other hosts except itself. Each flow has the same size of 1 Mbytes. All flows that traverse from the same source edge switch to the same destination edge switch belongs to one flow group. For the all-to-all communication, 56 flow groups are generated in the network with $N = 2$, and each flow group consists of 20 flows. 306 flow groups are generated in the fat-tree network with $N = 3$, and each flow group consists of 45 flows. 992 flow groups are generated in the network with $N = 4$, and each flow group consists of 80 flows. All the flow groups in the network starts at the same time. The value of every link's bandwidth in the network is randomly chosen between 1 and 10 Gbps.

We recorded the FGT of every flow group and observe their distribution. The Cumulative Distribution Functions (CDF) of the FGT in the three topologies are present in Fig. 8, where X-axis represents the transmission time of flow groups in milliseconds, Y-axis shows the cumulative percentage of flow groups. Fig. 8(a) shows the CDF of all the 56 flow groups in the fat-tree topology with $N = 2$. The number of multipath rules in every switch is set to be 200. 80% flow groups complete their transmission before 65 ms under NAMP. 80% flow groups complete before 80 ms under Niagara. 80% flow groups complete before 86 ms under WCMP. 80% flow groups complete before 112 ms under ECMP. In summary, our NAMP method can reduce the FGT of most flow groups by 45.4% shorter than ECMP, 25.6% shorter than WCMP and 18% shorter than Niagara in a small-scale DCN.

Fig. 8(b) shows the CDF of all the 306 flow groups in the fat-tree topology with $N = 3$. The number of multipath rules in every switch is set to be 2000. 80% flow groups complete their transmission before 118 ms under NAMP. 80% flow groups complete before 133 ms under Niagara. 80% flow groups complete before 142 ms under WCMP. 80% flow groups complete before 139 ms under ECMP. For most flow groups, NAMP reduces the FGT by 17.3% shorter than ECMP, 20.2% shorter than WCMP and 10% shorter than Niagara in a medium-scale DCN.

Fig. 8(c) shows the CDF of all the 992 flow groups in the fat-tree topology with $N = 4$. The number of multipath rules in every switch is set to be 4000. 80% flow groups complete their transmission before 211 ms under NAMP. 80% flow groups complete before 230 ms under Niagara. 80% flow groups complete before 233 ms under WCMP. 80% flow groups complete before 267 ms under ECMP. Thus, NAMP reduces the FGT of most flow groups by 21.2% shorter than ECMP, 11.2% shorter than WCMP and 8.7% shorter than Niagara in a large-scale DCN. By summarizing all the experiments for all-to-all communications among different scales of networks, NAMP reduces the FGT of flow groups in fat-tree networks for up to 45.4% shorter than ECMP, up to 25.6% shorter than WCMP, and up to 18% shorter than Niagara.

C. Real-World Data Center Traffic

We also compared NAMP with WCMP, Niagara, ECMP for map-reduce style real-world data center traffic as measured by Benson et.al. [24]. We generated traffic between randomly selected selected inter-switch source-destination hosts in the three topologies shown in Fig. 6. The flow sizes and flow

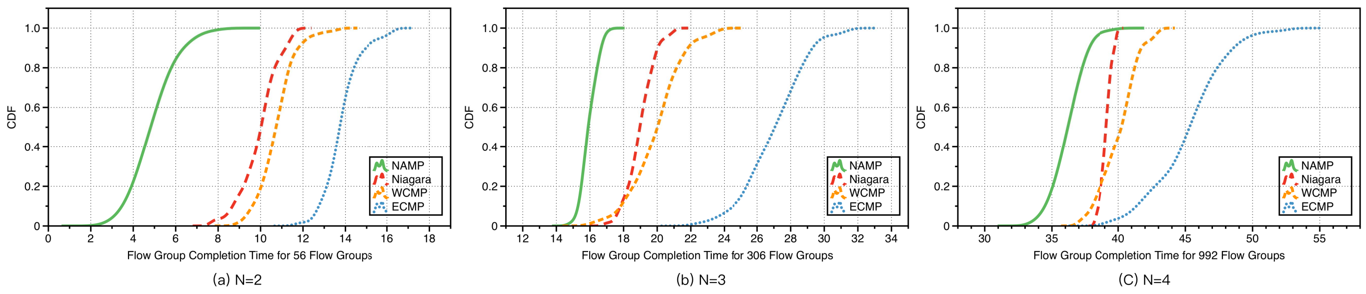


Fig. 9. Real-world data center traffic.

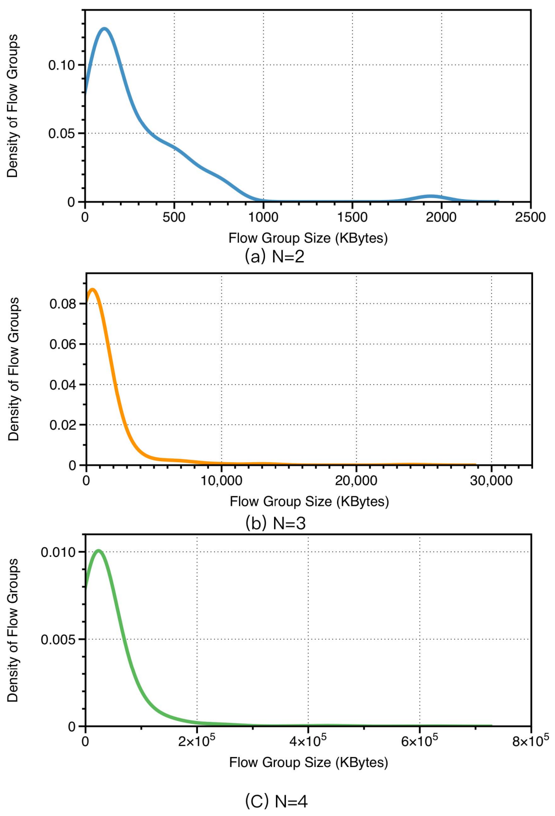


Fig. 10. Real-world data center flow size distribution.

inter-arrival times show a lognormal distribution as described in [24]. The traffic is generated using a empirical traffic generator, which is a client/server application for generating user-defined traffic patterns. In the three fat-tree networks, each host is set to run the client application. Each client randomly selects a subset of others hosts as the servers. For each request, the client samples from the input request size and fanout distributions to determine the request size and how many flows to generate in parallel for the request. Each server listens for incoming requests and replies with a flow with the requested size for each request.

We generated multiple flow groups each of which includes 20 flow. Each flow has a different size. Fig. 10(a) shows the distribution for the size of the 56 flow groups in the fat-tree network with $N = 2$. X-axis shows the size of flow groups. Y-axis presents the density of flow groups regarding different sizes. Most of the flow groups in Fig. 10(a) are less than 500 KB, while there are some large flows that are over 1 MB. Fig. 10(b) presents the distribution for the size of the 306 flow groups in the fat-tree network with $N = 3$. Most of the flow groups in Fig. 10(b) are less than 5MB, while few flow groups are larger than 10 MB. Fig. 10(c) presents the distribution for the size of the 992 flow groups in the network with $N = 4$. Most of the flow groups in Fig. 10(c) are less than 10 MB, while a small subset of flow groups are larger than 20 MB. The value of bandwidth for every link is randomly chosen between 1 Gbps and 10 Gbps.

Fig. 9 shows the cumulative distribution of all flow groups' FGT under NAMP, Niagara, WCMP and ECMP in the three fat-tree topologies. Fig. 9(a) presents the CDF of all the 56 flow groups in the fat-tree network with $N = 2$. Note that the size of all the flow groups forms a lognormal distribution illustrated in Fig. 10(a). The number of rules in the multipath table of every switch is set to 200. As it is shown in Fig. 9(a), 80% flow groups complete their transmission before 6 ms under NAMP. 80% flow groups complete the transmission before 11 ms under Niagara. 80% flow groups complete before 12 ms under WCMP. 80% flow groups complete before 15 ms under ECMP. Therefore, NAMP reduces the FGT for most of flow groups by 45.4% shorter than Niagara, 50% shorter than WCMP and 60% shorter than ECMP.

Fig. 9(b) presents the CDF of all the 306 flow groups in the fat-tree network with $N = 3$. The size of all the flow groups forms a lognormal distribution illustrated in Fig. 10(b). The number of rules in the multipath table of every switch is set to 2000. As it is shown in Fig. 9(b), 80% flow groups complete their transmission before 17 ms under NAMP. 80% flow groups complete the transmission before 20 ms under Niagara. 80% flow groups complete before 21 ms under WCMP. 80% flow groups complete before 29 ms under ECMP. Therefore, NAMP reduces the FGT for most of flow groups by 15% shorter than Niagara, 19% shorter than WCMP and 41.4% shorter than ECMP.

Fig. 9(c) presents the CDF of all the 992 flow groups in the fat-tree network with $N = 4$. The size of all the flow groups forms a lognormal distribution illustrated in Fig. 10(c). The number of rules in the multipath table of every switch is set to 4000. As it is shown in Fig. 9(c), 80% flow groups complete their transmission before 37 ms under NAMP. 80% flow groups complete the transmission before 40 ms under Niagara. 80% flow groups complete before 42 ms under WCMP. 80% flow groups complete before 48 ms under ECMP. Therefore, NAMP reduces the FGT for most of flow groups

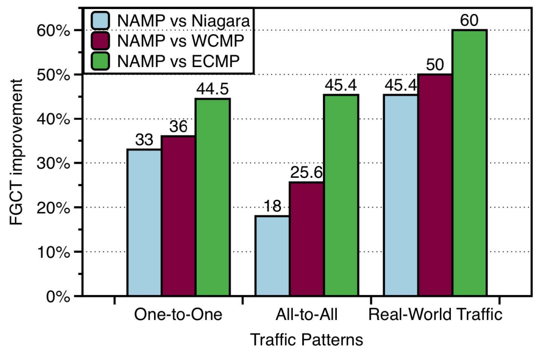


Fig. 11. Performance comparison results.

by 7.5% shorter than Niagara, 11.9% shorter than WCMP and 23% shorter than ECMP. For the real-world data center traffic among all the experimental networks, NAMP reduces FGT by up to 45.4% shorter than Niagara, up to 50% shorter than WCMP and up to 60% shorter than ECMP.

In summary, Fig. 11 shows the performance for NAMP against Niagara, WCMP and ECMP. X-axis presents the three traffic patterns simulated in the experiments. Y-axis shows the amount of reduced FGT in percentage for NAMP. As it is described in Fig. 11, for one-to-one traffic, NAMP reduces the FGT by up to 33% shorter than Niagara, up to 36% shorter than WCMP, and up to 44.5% shorter than ECMP. For all-to-all communication, NAMP reduces the FGT by up to 18% shorter than Niagara, up to 25.6% shorter than WCMP, and up to 45.4% shorter than ECMP. For the real-world data center traffic, NAMP reduces the FGT by up to 45.4% shorter than Niagara, up to 50% shorter than WCMP, and up to 60% shorter than ECMP. NAMP efficiently reduces FGT than the WCMP, Niagara and ECMP. The reason is that NAMP is designed to minimize FGT by considering the heterogeneous bandwidth, while the other three schemes ignore the issue on the quality of service.

IX. Conclusion

SDN-based multipath routing aims at distributing the traffic of a flow group among multiple paths in a weighted distribution. Since the number of rules in the multipath table is limited, it is not scalable to implement the ideal traffic splitting ratio for every flow group. State-of-the-art solutions, WCMP and Niagara, do not minimize the flow group transmission time when re-allocating the routing rules to flow groups. We proposed NAMP, an efficient network-aware multipathing scheme, to minimize the flow group transmission time by considering the heterogeneous network bandwidth. The table fitting problem is firstly formulated as a bi-convex integer programming problem. A novel LP-based alternate convex search solution is proposed by discovering the special structure of the problem. NAMP works in a online event-triggered manner to new flow groups without affecting existing flows. Extensive experimental results show that NAMP outperforms Niagara, WCMP and ECMP by up to 45.4%, 50% and 60% respectively. It is also easy to implement NAMP in existing SDN controller platforms such as RYU by using existing LP solvers and OpenFlow APIs.

Appendix

Theorem 2: Minimizing $g(\mathbf{t})$ generates the lexicographical minimum of $\mathbf{t}$.

Proof: The proof is elaborated in two steps. Suppose we choose two feasible vectors in the feasibility region of $\mathbf{t}$: $\mathbf{u}$ and $\mathbf{v}$, which are expressed as follows. $\mathbf{u} = (u_1, \ldots, u_{ij}, \ldots, u_{NN})$, $\mathbf{v} = (v_1, \ldots, v_{ij}, \ldots, v_{NN})$, where $i = 1, \ldots, N$ and $j = 1, \ldots, N$. Note that both $\mathbf{u}$ and $\mathbf{v}$ are two N^2-dimensional vectors whose elements are sorted in non-decreasing order. Since each element t_{ij} in $\mathbf{t}$ is defined in (15), we can express each element in $\mathbf{u}$ and $\mathbf{v}$ as follows:

$$u_{ij} = e_{ij} y_{ij}^u + c_{ij}, \quad v_{ij} = e_{ij} y_{ij}^v + c_{ij}, \tag{41}$$

where e_{ij} and c_{ij} are defined in (11). y_{ij}^u and y_{ij}^v are two feasible values of the variable y_{ij} subjective to the constraint (13) and (14).

We first prove that $g(\mathbf{u}) < g(\mathbf{v})$ if we have $\mathbf{u} \prec \mathbf{v}$. Then, we prove that $\mathbf{u} \prec \mathbf{v}$ if we have $g(\mathbf{u}) < g(\mathbf{v})$. Given $\mathbf{u} \prec \mathbf{v}$, let a and b denote the value of i and j for the first positive element of $\mathbf{v} - \mathbf{u}$, which means $v_{ab} > u_{ab}$. We can obtain $v_{ij} = u_{ij}$ if (1) $i < a$ or (2) $i = a$ and $j \leq b$. Then, we can compute

$$\begin{aligned} g(\mathbf{v}) - g(\mathbf{u}) &= \sum_{i=1}^{N}\sum_{j=1}^{N} N^{2Mv_{ij}} - \sum_{i=1}^{N}\sum_{j=1}^{N} N^{2Mu_{ij}} \\ &= \sum_{i=a}^{N}\sum_{j=b}^{N} N^{2Mv_{ij}} - \sum_{i=a}^{N}\sum_{j=b}^{N} N^{2Mu_{ij}} \\ &> \sum_{i=a}^{N}\sum_{j=b}^{N} N^{2Mv_{ij}} - N^2 N^{2Mu_{ab}} \\ &= (N^{2Mv_{ab}} - N^{2Mu_{ab}+2}) + \sum_{j=b+1}^{N} N^{2Mv_{aj}} \\ &\quad + \sum_{i=a+1}^{N}\sum_{j=b}^{N} N^{2Mv_{ij}}. \end{aligned} \tag{42}$$

Notice that the second and the third term in the last expression are positive values. Next we prove the first term $(N^{2Mv_{ab}} - N^{2Mu_{ab}+2})$ is a non-negative integer, which means to compare the value between Mv_{ab} and $Mu_{ab} + 1$. The term Mv_{ab} can be expanded as follows:

$$Mv_{ab} = Me_{ij} y_{ab}^v + Mc_{ab}. \tag{43}$$

Notice that M is a number such that $M \times e_{ij}$ and $M \times c_{ij}$ are integers for every i, j. Since y_{ij}^v is a non-negative integer, the value of Mv_{ab} is an integer. Same reason, the value of Mu_{ab} is an integer. Since $v_{ab} > u_{ab}$, we can obtain $Mv_{ab} > Mu_{ab}$. Since Mv_{ab} and Mu_{ab} are both integers, Mv_{ab} must be at least 1 larger than Mu_{ab}. Thus, $Mv_{ab} \geq Mu_{ab} + 1$. Therefore, the last expression of (42) is positive. This completes the proof that we can obtain $g(\mathbf{u}) < g(\mathbf{v})$ if $\mathbf{u} < \mathbf{v}$.

In the second step, we prove the reverse relation that $\mathbf{u} < \mathbf{v}$ if $g(\mathbf{u}) < g(\mathbf{v})$. Let a and b denote the value of i and j for the first non-zero value element of $\mathbf{u} - \mathbf{v}$, which means $u_{ab} \neq v_{ab}$.

We can further obtain that $u_{ij} = v_{ij}$ if (1) $i < a$ or (2) $i = a$ and $j \leq b$. Next we prove that

$$\begin{aligned} g(\mathbf{v}) - g(\mathbf{u}) &= \sum_{i=1}^{N}\sum_{j=1}^{N} N^{2Mv_{ij}} - \sum_{i=1}^{N}\sum_{j=1}^{N} N^{2Mu_{ij}} \\ &= \sum_{i=a}^{N}\sum_{j=b}^{N} N^{2Mv_{ij}} - \sum_{i=a}^{N}\sum_{j=b}^{N} N^{2Mu_{ij}} \\ &< N^2 N^{2Mv_{ab}} - N^{2Mu_{ab}} - \sum_{j=b+1}^{N} N^{2Mu_{aj}} \\ &\quad - \sum_{i=a+1}^{N}\sum_{j=b}^{N} N^{2Mu_{ij}} < N^{2Mv_{ab}+2} - N^{2Mu_{ab}}. \end{aligned} \tag{44}$$

Since $g(\mathbf{u}) < g(\mathbf{v})$, we can obtain that $N^{2Mv_{ab}+2} - N^{2Mu_{ab}} > 0$. Thus, $Mv_{ab}+1 > Mu_{ab}$. The proof elaborating on (43) shows that both Mv_{ab} and Mu_{ab} are integers. Suppose $Mu_{ab} > Mv_{ab}$, which means Mu_{ab} is at least 1 larger than Mv_{ab}. We can obtain $Mu_{ab} \geq Mv_{ab} + 1$, which contradicts with the result that $Mv_{ab} + 1 > Mu_{ab}$. Thus, it is certain that $Mu_{ab} \leq Mv_{ab}$. Since $v_{ab} \neq u_{ab}$ in the definition of a and b, we can obtain that $Mu_{ab} < Mv_{ab}$, which further derives $u_{ab} < v_{ab}$. It also indicates that the first non-zero element of $\mathbf{u} - \mathbf{v}$ is a negative value. Therefore, we have $\mathbf{u} < \mathbf{v}$, which completes the proof. □

REFERENCES

[1] N. Katta, O. Alipourfard, J. Rexford, and D. Walker, "Infinite CacheFlow in software-defined networks," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw. (HotSDN)*, 2014, pp. 175–180.

[2] Y. Nakagawa, K. Hyoudou, C. Lee, S. Kobayashi, O. Shiraki, and T. Shimizu, "Domainflow: Practical flow management method using multiple flow tables in commodity switches," in *Proc. CoNEXT*, 2013, pp. 399–404.

[3] J. Zhou *et al.*, "WCMP: Weighted cost multipathing for improved fairness in data centers," in *Proc. EuroSys*, 2014, p. 5.

[4] N. Kang, M. Ghobadi, J. Reumann, A. Shraer, and J. Rexford, "Efficient traffic splitting on commodity switches," in *Proc. 11th ACM Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*, 2015, p. 6.

[5] X.-N. Nguyen, D. Saucez, C. Barakat, and T. Turletti, "OFFICER: A general optimization framework for openflow rule allocation and endpoint policy enforcement," in *Proc. IEEE Conf. Comput. Commun.(INFOCOM)*, Apr. 2015, pp. 478–486.

[6] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "On the effect of forwarding table size on SDN network utilization," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2014, pp. 1734–1742.

[7] G. Zhao, L. Huang, Z. Yu, H. Xu, and P. Wang, "On the effect of flow table size and controller capacity on SDN network throughput," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2017, pp. 1–6.

[8] G. Zhao, H. Xu, S. Chen, L. Huang, and P. Wang, "Deploying default paths by joint optimization of flow table and group table in SDNs," in *Proc. IEEE 25th Int. Conf. Netw. Protocols (ICNP)*, Oct. 2017, pp. 1–10.

[9] X.-N. Nguyen, D. Saucez, C. Barakat, and T. Turletti, "Optimizing rules placement in openflow networks: Trading routing for better efficiency," in *Proc. HotSDN*, 2014, pp. 127–132.

[10] H. Huang, P. Li, S. Guo, and B. Ye, "The joint optimization of rules allocation and traffic engineering in software defined network," in *Proc. IEEE 22nd Int. Symp. Qual. Service (IWQoS)*, May 2014, pp. 141–146.

[11] F. Giroire, J. Moulierac, and T. K. Phan, "Optimizing rule placement in software-defined networks for energy-aware routing," in *Proc. IEEE Global Commun. Conf.*, Dec. 2014, pp. 2523–2529.

[12] R. Wang *et al.*, "Openflow-based server load balancing gone wild," *Proc. USENIX Workshop Hot Topics Manage. Internet, Cloud, Enterprise Netw. Services (Hot-ICE)*, vol. 11, 2011, p. 12.

[13] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments," in *Proc. OSDI*, 2008, vol. 8, no. 4, p. 7.

[14] W. Xia, P. Zhao, Y. Wen, and H. Xie, "A survey on data center networking (DCN): Infrastructure and operations," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 1, pp. 640–656, 1st Quart., 2017.

[15] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, p. 63, Oct. 2008.

[16] T. Benson, A. Anand, A. Akella, and M. Zhang, "Microte: Fine grained traffic engineering for data centers," in *Proc. 7th Conf. Emerg. Netw. Exp. Technol.*, 2011, p. 8.

[17] A. Greenberg *et al.*, "Vl2: A scalable and flexible data center network," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 51–62, 2009.

[18] A. Singh *et al.*, "Jupiter rising: A decade of Clos topologies and centralized control in Google's datacenter network," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 183–197, 2015.

[19] F. A. Al-Khayyal and J. E. Falk, "Jointly constrained biconvex programming," *Math. Oper. Res.*, vol. 8, no. 2, pp. 273–286, May 1983.

[20] R. R. Meyer, "A class of nonlinear integer programs solvable by a single linear program," *SIAM J. Control Optim.*, vol. 15, no. 6, pp. 935–946, Nov. 1977.

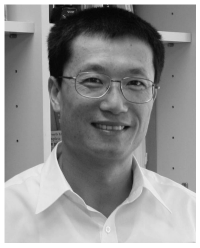
[21] P. Camion, "Characterization of totally unimodular matrices," *Proc. Amer. Math. Soc.*, vol. 16, no. 5, p. 1068, May 1965.

[22] *RYU*. Accessed: Dec. 2017. [Online]. Available: https://osrg.github.io/ryu/

[23] *NS3: Discrete-Event Network Simulator for Internet Systems*. Accessed: Feb. 2018. [Online]. Available: https://www.nsnam.org/

[24] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. 10th Annu. Conf. Internet Meas. (IMC)*, 2010, pp. 267–280.

Yingying Cheng received the B.E. degree in computer science from the South China University of Technology in 2012, the M.E. degree in computer science from the Harbin Institute of Technology in 2014, and the Ph.D. degree from the Department of Computer Science, City University of Hong Kong in 2018. She was a Research Fellow at the Department of Computer Science, City University of Hong Kong in 2019. She is also a Visiting Research Student at the University of Toronto in 2017. Her research interests include distributed systems, optimization systems, cloud computing, and software-defined networks.



Xiaohua Jia (Fellow, IEEE) received the B.Sc. and M.Eng. degrees from the University of Science and Technology of China in 1984 and 1987, respectively, and the D.Sc. degree in information science from the University of Tokyo in 1991. He is currently the Head and Chair Professor with the Department of Computer Science, City University of Hong Kong. His research interests include cloud computing and distributed systems, computer networks, and mobile computing. He is the General Chair of the ACM MobiHoc 2008, the TPC Co-Chair of the IEEE GlobeCom 2010 Ad Hoc and Sensor Networking Symposium, and the Area-Chair of the IEEE INFOCOM 2010 and 2015. He is an Editor of the IEEE INTERNET OF THINGS, the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS (2006–2009), *Wireless Networks*, the *Journal of World Wide Web*, and the *Journal of Combinatorial Optimization*.