Joint Switch Upgrade and Controller Deployment in Hybrid Software-Defined Networks

Zehua Guo, Weikun Chen, Ya-Feng Liu, *Senior Member, IEEE*, Yang Xu, *Member, IEEE*, and Zhi-Li Zhang, *Fellow, IEEE*

***Abstract*— To improve traffic management ability, Internet Service Providers (ISPs) are gradually upgrading legacy network devices to programmable devices that support Software-Defined Networking (SDN). The coexistence of legacy and SDN devices gives rise to a hybrid SDN. Existing hybrid SDNs do not consider the potential performance issues introduced by a centralized SDN controller: flow requests processed by a highly loaded controller may experience long-tail processing delay; inappropriate multi-controller deployment could increase the propagation delay of flow requests. In this paper, we propose to jointly consider the deployment of SDN switches and their controllers for hybrid SDNs. We formulate the joint problem as an optimization problem that maximizes the number of flows that can be controlled and managed by the SDN and minimizes the propagation delay of flow requests between SDN controllers and switches under a given upgrade budget constraint. We show this problem is NP-hard. To efficiently solve the problem, we propose some techniques (e.g., strengthening the constraints and adding additional valid inequalities) to accelerate the global optimization solver for solving the problem for small networks and an efficient heuristic algorithm for solving it for large networks. The simulation results from real network topologies illustrate the effectiveness of the proposed techniques and show that our proposed heuristic algorithm uses a small number of controllers to manage a high amount of flows with good performance.**

***Index Terms*— Complexity analysis, controller deployment, heuristic algorithm, hybrid software-defined networking (SDN), switch upgrade, upgrade budget.**

I. Introduction

SOFTWARE-DEFINED Networking (SDN) has been widely studied and gradually adapted for campus networks [1], data center networks [2], Wide Area Networks (WANs) [3], enterprise networks [4], and Internet exchange points [5]. Due to the cost and operational considerations, SDN technology is usually deployed in an incremental fashion. In particular, at each time of network upgrade, only a set of selected legacy network devices (i.e., layer-3 routers and layer-2 switches) are upgraded to SDN switches. AT&T converted 34% of its network to SDN by the end of 2016 and virtualized 55% of its network to software by the end 2017. Its final goal is to reach 75% softwarization of its network by 2020 [6]. Therefore, the legacy network devices and SDN switches may coexist for a long time. In this paper, we will refer to such a network as a hybrid SDN.

A WAN usually consists of many network devices at geo-distributed locations. A straightforward method to upgrade legacy network devices in WANs to SDN switches is based on the locations of network devices, for example, upgrade a part of WAN with network devices in proximity. The partial upgraded network can enjoy the benefit of SDN, but the performance improvement of the entire network is limited. An efficient solution for network providers is to spread the benefit of upgraded SDNs in the entire network. Based on this consideration, existing studies proposed to upgrade legacy network devices in WANs to SDN switches for different reasons or with different motivations, such as traffic engineering [7], [8], flexible routing [9], link failure recovery [10], power saving [11], [12], and safe update [13]. Essentially, the benefit of SDN is to flexibly control flows. Once a flow traverses an SDN switch, its forwarding path can be flexibly controlled. We call such traffic the programmable traffic. The selection of switches to be upgraded to SDN switches has a great impact on the network's programmable performance. In a WAN, switches tend to have quite different numbers of flows. If we randomly select some of them to upgrade, we may not be able to achieve our objective to maximize the number of programmable flows. Some studies aim to maximize the amount of programmable traffic under the constraint of a given upgrade budget [14], [15]. However, the impact of SDN controller on the network upgrade is not taken into consideration in these works.

The control plane of SDN has evolved from one single controller to multiple controllers to circumvent the limited computational resource of a controller server and avoid single point of failure. For a large wide-area SDN, multiple distributed controllers are physically deployed at different locations to achieve the function of a logically centralized control plane, and the controllers synchronize with each other to guarantee

Manuscript received October 10, 2018; revised January 30, 2019 and March 10, 2019; accepted March 13, 2019. Date of publication March 21, 2019; date of current version April 16, 2019. The work of Z. Guo and Z.-L. Zhang was supported by NSF under Grants CNS-1411636, CNS-1618339, CNS-1617729, CNS-1814322, and CNS-1836772. The work of W. Chen and Y.-F. Liu was supported in part by the National Natural Science Foundation of China (NSFC) under Grants 11671419, 11571221, 11688101, and 11631013, and in part by the Beijing Natural Science Foundation under Grant L172020. *(Corresponding author: Ya-Feng Liu.)*

Z. Guo is with the School of Automation, Beijing Institute of Technology, Beijing 100081, China, and also with the Department of Computer Science and Engineering, University of Minnesota Twin Cities, Minneapolis, MN 55455 USA (e-mail: guolizihao@hotmail.com).

W. Chen and Y.-F. Liu are with the State Key Laboratory of Scientific and Engineering Computing, Institute of Computational Mathematics and Scientific/Engineering Computing, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing 100190, China (e-mail: cwk@lsec.cc.ac.cn; yafliu@lsec.cc.ac.cn).

Y. Xu is with the School of Computer Science, Fudan University, Shanghai 200433, China (e-mail: xuy@fudan.edu.cn).

Z.-L. Zhang is with the Department of Computer Science and Engineering, University of Minnesota Twin Cities, Minneapolis, MN 55455 USA (e-mail: zhzhang@cs.umn.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSAC.2019.2906743

the consistency of the entire network [16], [17]. To avoid the single-point-of-failure problem, backup controllers maybe deployed [18], [19]. If one controller instance crashes, other active instances will still work without service interruption. Popular SDN controllers (e.g., ONOS and OpenDayLight) usually use three controllers to provide resilient service at one location, and the three controllers communicate with each other (e.g., using Raft [20]) to guarantee the network state consistency [21], [22].

Deploying multiple controllers in a hybrid SDN should take the following two factors into account. First, controllers should be able to process flow requests from the upgraded SDN switches in a timely manner. If a controller is overloaded, the requests handled by it may suffer from long-tail latency [23], which might significantly degrade the network performance [24]. Second, the deployment locations of controllers affect the propagation delay of flow requests and network state pulling because the propagation delay in WANs is usually a significant part of the total delay [25], [26]. Existing works [7], [8], [14], [15] for the network upgrade did not consider the above factors, which may lead to some undesirable performance degradations. We detail the two factors in Section II.

In this paper, we consider the above two factors and propose two objectives: (1) maximizing the number of flows managed by SDN and (2) minimizing the propagation delay of flow requests from SDN switches with an upgrade budget constraint. The first objective aims to control as many flows as possible by upgrading legacy devices to SDN switches, while the second objective aims to reduce the propagation delay between the SDN controllers and switches by effectively deploying a few controllers near SDN switches. Our problem is to find the locations of upgraded switches, the locations of deployed controllers, and the mappings between the controllers and the upgraded switches (i.e., which controllers control which upgraded SDN switches) to maximize the number of controlled programmable flows and at the same time minimize the propagation delay between the controllers and upgraded switches under individual controllers' processing ability and upgrade budget constraints. We first formulate a two-stage optimization problem that optimizes one objective at each stage and then transform the two-stage problem into a one-stage problem to simplify the solution procedure. We prove that with a careful choice of the parameter the optimal solution of the one-stage formulation is also the optimal solution of the two-stage formulation.

It is worth noting that our work is different from the controller deployment in pure SDNs. In pure SDNs, all switches are SDN switches, and the controllers' deployment only needs to consider two aspects: (1) the controllers' location and number and (2) mapping between SDN switches and controllers. In hybrid SDNs, our problem needs to consider one more aspect: the location and number of the upgraded SDN switches. In our problem, the three aspects are related to each other, and our problem in hybrid SDNs is more complicated than the controllers' deployment in pure SDNs.

To efficiently solve the problem, we analyze the problem's structure and propose several solutions. For the problem in small networks, we accelerate the global optimization solver by proposing a new problem formulation. For the problem in large networks, to further improve the computational efficiency, we propose a heuristic algorithm named MapFirst, which first orders the (relaxed) mapping variables between controllers and upgraded switches according to their importance/weights (obtained by solving a linear program relaxation) and then sequentially determines the (binary) mapping variables based on their impacts on the objective function.

We conduct simulations using real network topologies from Topology Zoo [27]. The simulation results on multiple topologies verify the effectiveness and efficiency of the new problem formulation. We further compare MapFirst with optimal solutions and other heuristic algorithms, and the results show (1) MapFirst outperforms other heuristic algorithms by using a small number of controllers to facilitate a high amount of flows from upgraded SDN switches, and (2) compared to the optimal solution, MapFirst is able to achieve a comparable performance but enjoys a significantly low complexity.

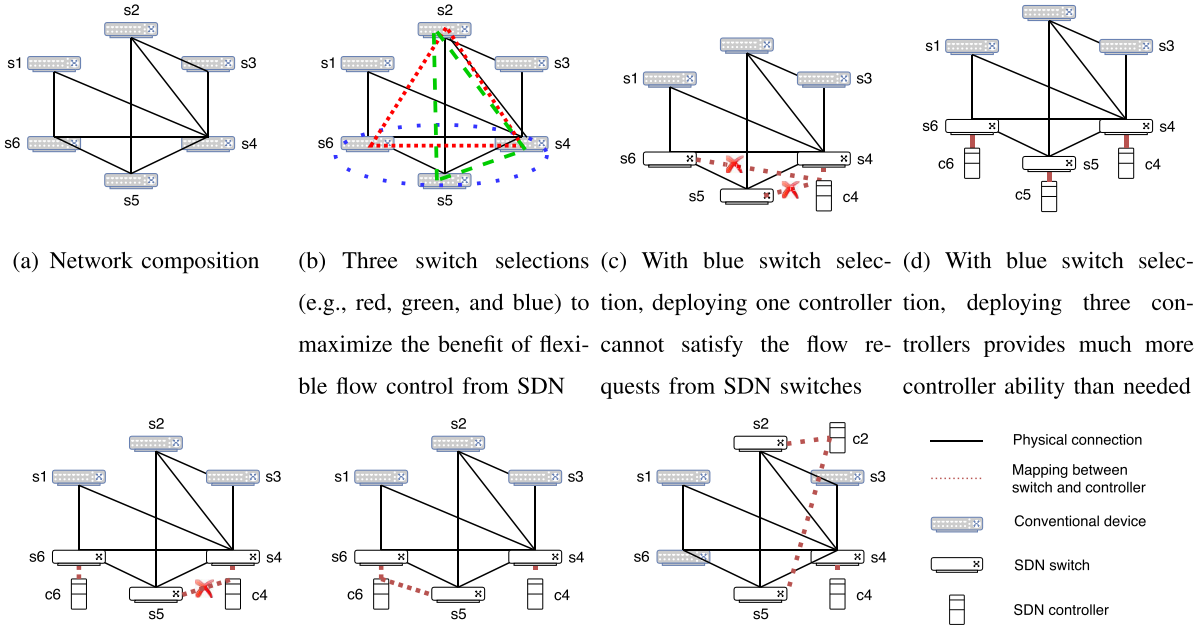
The contributions of the paper are summarized as follows:

1) *Novel Problem Formulation:* We identify the impact of the control plane on the network upgrade and formulate an optimization problem to guarantee the performance of the hybrid SDN by jointly upgrading switches and deploying controllers.
2) *Efficient Solutions:* We propose efficient exact and heuristic solutions to solve the problem for both small and large networks. Our simulation results (on real network topologies) demonstrate that our exact solution significantly accelerates the optimization solver, and our proposed MapFirst is able to return a high-quality solution with a significant less CPU time.
3) *Theoretical Analysis:* We provide some analysis on the problem and the solutions. For the problem, we provide a rigorous NP-hardness proof and shed a useful insight that the problem (probably) does not admit constant-ratio approximation algorithms to only approximate the total delay in our problem. In addition, we also identify a special case of the problem which is strongly polynomial time solvable. For the proposed MapFirst, we analyze its worst-case complexity and prove that it is able to return the global solution of the problem if each controller can control at most one SDN switch.

The rest of this paper is organized as follows: Section II illustrates the motivation of the paper with some examples. Section III formulates the Joint Switch Upgrade and Controller Deployment (JSUCD) problem, and Section IV analyzes the parameter selection and the complexity of the problem. Section V introduces exact and heuristic solutions for the JSUCD problem. Section VI presents the simulation results and analysis. Section VII introduces the related works. Section VIII concludes the paper and presents our future work.

II. An Example and Motivation

In this section, we use a motivation example in Fig. 1 to show how the network upgrade affects the performance of the hybrid SDN.



(a) Network composition

(b) Three switch selections (e.g., red, green, and blue) to maximize the benefit of flexible flow control from SDN

(c) With blue switch selection, deploying one controller cannot satisfy the flow requests from SDN switches

(d) With blue switch selection, deploying three controllers provides much more controller ability than needed

(e) The impact of the controller processing ability on the switch-controller mapping of blue switch selection

(f) A feasible solution of blue switch selection that maximizes the number of programmable flows and minimizes the propagation delay of flow requests with a few controllers

(g) A feasible solution of green switch selection that has a higher propagation delay of flow requests than the one with the solution in 1(f)

Fig. 1. A motivation example.

A. Example Background

Fig. 1(a) shows the network composition: the network has 6 legacy switches s_1 to s_6 deployed at six different locations with uneven interconnection. We assume that the number of flows in each switch is proportional to the number of its links, and the total number of flows on a link is normalized as 1. The number of normalized flows at the six switches are: $f_{s1} = 2, f_{s2} = 3, f_{s3} = 2, f_{s4} = 5, f_{s5} = 3, f_{s6} = 3$. The normalized processing ability of each controller is $p_c = 6$. The upgrade budget includes switch upgrade cost and controller deployment cost. In this example, the upgrade budget allows upgrading at most four legacy devices to SDN switches, and the cost of an SDN switch is three times of a controller. In other words, if we upgrade one less switch, we can deploy three more controllers at three locations.[1] We assume that one SDN switch can only be controlled by exactly one controller, and one controller can potentially control multiple SDN switches. Our problem is to find a feasible solution that contains a set of upgraded switches, a set of controllers, and the mappings between the upgraded switches and controllers to maximize the number of controlled programmable flows and at the same time minimize the delay between the controllers and upgraded switches under the constraints of the upgrade budget and individual controller's processing ability.

[1]At a location, a controller is physically implemented by a controller instance cluster to prevent single-node-of-failure [18], [19]. We can either use two instances or three instances for a cluster. However, the odd number accelerates the primary controller selection in the case of the controller failure, and the production networks usually use three instances [21], [22]. In the rest of the paper, we use a controller to represent a controller cluster at a location since a cluster usually uses one controller to process requests, and the other two are just backup and thus do not process any requests.

B. Impact of the Network Upgrade on the Hybrid SDN

1) Maximizing the Benefit of SDN by Selecting Upgraded SDN Switches: If a flow traverses one SDN switch, it is a programmable flow and we can enjoy the benefit of SDN by flexibly controlling the flow's forwarding path. Thus, the benefit of SDN depends on the number of programmable flows. We use A_s to denote the total number of programmable flows from the upgraded SDN switches. Our first goal is to maximize A_s with a given switch upgrade budget. There are 20 switch combinations for selecting three switches from six switches to upgrade. Fig. 1(b) shows three switch selection combinations that control the largest number of flows $A_s = 3 + 3 + 5 = 11$: $red = \{s_2, s_4, s_6\}$, $blue = \{s_4, s_5, s_6\}$, $green = \{s_2, s_4, s_5\}$.

2) Guaranteeing the Processing Performance of Flow Requests by Considering the Processing Ability of Controllers:

We use $blue$ switch selection in the following explanation. The red and $green$ switch selections follow the similar explanation. We use A_c to denote the total processing ability of the deployed controllers. Existing works show when a controller processes more requests than its normal processing load, the processing delay of the requests could be five times longer than the one under the normal load [23], [24]. To maintain the processing performance of flow requests, after a network upgrade, for the entire network, it is necessary to require $A_s \leq A_c$, and for controllers, each one should not process the number of flow requests larger than its normal processing ability. In Fig. 1(c), deploying one controller can either control the flows at switch s_4 or the flows at the other two switches, and thus deploying one controller is not enough. In Fig. 1(d), deploying one controller for each switch can satisfy the control requirement, but the controllers' processing ability of such a deployment is much larger than the number of flows from SDN switches. The extra number of controllers could also bring extra overhead. If we deploy many controllers, the number of switches controlled by a controller would be small, and a controller's control ability on the network would be reduced. Thus, after a network upgrade, we prefer to deploy a few controllers to satisfy the demand of SDN switches. In this example, we only need two controllers since $A_s < A_c = p_c * 2 = 12, f_{s_4} < p_c$, and the load of the other two switches in red, $green$, or $blue$ switch selection is less than p_c. Thus, a good solution is to upgrade three switches with the number of flows 5, 3, and 3 and deploy only two controllers.

3) Maintaining the Good Propagation Performance of Flow Requests by Considering the Locations of SDN Switches and Controllers: An SDN switch and a controller interact with each other in two ways: one SDN switch can send a flow request to the controller when it does not know how to process the flow, and a controller can dynamically change the paths of flows on its controlled SDN switches to improve the network performance (e.g., when identifying a congestion). Previous works [25], [26] show that the propagation latency in WANs is the dominant factor among all latencies because the propagation latency bounds the control reactions of a controller that can be executed at a reasonable speed. A long propagation latency could limit availability (e.g., link failure recovery) and convergence time (e.g., network state pulling, routing convergence). Thus, for WANs, an important factor for controller placement is to minimize the total propagation delay among SDN switches and controllers. In WANs, the propagation delay of a request is proportional to the distance between a sender and a receiver. To maintain the good propagation performance of flow requests between the SDN's control plane and data plane, we should minimize the propagation delay of flow requests between SDN switches and controllers, which motivates us to deploy controllers near the upgraded SDN switches.

In the example of $blue$ switch selection, the number of flows in switch s_4 is the largest one, and it is very close to the control capacity of a controller. Hence, we select switch s_4 to upgrade and deploy controller c_4 at location 4 for the switch. The rest two switches are switches s_5 and s_6, which can be controlled by a controller deployed at location 5 or 6. We use M_{s_i,c_j} to denote the mapping between switch s_i and controller c_j. Since the delay between a switch and a controller is minimized when the controller is deployed at the same place of the switch, we have two candidate solutions: $blue_1 = \{s_4, s_5, s_6, c_4, c_5, M_{s_4,c_4}\}$, $blue_2 = \{s_4, s_5, s_6, c_4, c_6, M_{s_4,c_4}\}$.

4) Impact of Individual Controller's Processing Ability on Controller-Switch Mappings: After selecting upgraded switches and deployed controllers, we should also map each upgraded switch to a controller for control. Each SDN switch can be managed by only one controller, and one controller could possibly manage multiple switches. Fig. 1(e) and 1(f) show two switch-controller mappings of the solution $blue_2$. In the two subfigures, switch s_4 is mapped to controller c_4, and switch s_6 is mapped to controller c_6. However, switch s_5 has different mappings. In Fig. 1(e), switch s_5 is mapped to controller c_4, and the controller's load reaches $3 + 5 = 8$, which exceeds its maximum processing ability 6 while the load of controller c_6 is only 3. Thus, the mapping between switch s_5 and controller c_4 is not allowed given that s_4 is already mapped to c_4. In Fig. 1(f), switch s_5's mapping is feasible since its mapping to c_6 keeps c_6's load at $3 + 3 = 6$, which does not exceed its maximum processing ability. Thus, switch-controller mappings without considering the controller processing ability constraint could also lead to an infeasible solution.

Fig. 1(g) shows a solution of $green$ switch selection. We use D_{s_i,s_j} to denote the distance (e.g., the length of the direct link or the shortest path) between switches s_i and s_j. Because of $D_{s_2,s_5} = 2 * D_{s_6,s_5}$, we have $D_{s_5,c_2} = 2 * D_{s_6,s_5}$. One can check that optimal solutions of $blue_1$ and $blue_2$ switch selections outperform the optimal solution of $green$ switch selection in terms of the total propagation delay.

C. Design Criteria of the Network Upgrade

In the above network upgrade, we consider the following factors to upgrade a legacy network to the hybrid SDN:

1) maximizing the number of programmable flows from the upgraded SDN switches,
2) guaranteeing the processing performance of flow requests at controllers, and
3) minimizing the total propagation delay of flow requests between controllers and SDN switches.

The first factor depends on the upgraded switches. The second factor relates to the processing ability of individual controllers. The third factor must take the locations and mappings of controllers and SDN switches into consideration. By considering all these factors, an optimal result would come from a very complicated procedure since the factors couple with each other. For example, to get a better result, mapping selection in Section II-B4 could be determined before or simultaneously when selecting switches to upgrade and controllers to deploy. In the next section, we will formulate an optimization problem that takes into account all the above factors for an efficient network upgrade.

III. Problem Formulation

In this section, we first introduce constraints and objective functions and then formulate our optimization problem.

A. System Description

In a network upgrade from a legacy network to a hybrid SDN, we upgrade some of its switches to SDN switches and deploy some controllers to manage all SDN switches. The network consists of N switches deployed at N locations. The number of flows in switch s_i is $R_i \geq 0$ $(1 \leq i \leq N)$. We use homogeneous controllers, and the processing ability of controller c_j is $A > 0$ $(1 \leq j \leq N)$.

We use $x_i = 1$ to denote that switch s_i is upgraded to the SDN switch; otherwise $x_i = 0$. Similarly, we use $y_j = 1$ to denote that controller c_j is deployed at location j and otherwise $y_j = 0$; we use $z_{ij} = 1$ to denote that an SDN switch s_i is mapped to controller c_j and otherwise $z_{ij} = 0$. The default relationships between x_i, y_j, and z_{ij} are shown below:

$$z_{ij} \leq x_i, \quad \forall\ i,\ \forall\ j, \tag{1}$$

$$z_{ij} \leq y_j, \quad \forall\ i,\ \forall\ j. \tag{2}$$

Equations (1) and (2) imply that a feasible switch-controller mapping must satisfy two conditions: a switch is upgraded to an SDN switch, and its mapping controller is deployed.

B. Constraints

1) Controller-Switch Mapping Constraint: If switch s_i is upgraded, it must be controlled by only one controller; if switch s_i is not upgraded, it is not controlled by any controller. Thus, we have

$$x_i = \sum_{j=1}^{N} z_{ij}, \quad \forall\ i. \tag{3}$$

If controller c_j is not deployed, it does not control any switches; if controller c_j is deployed, it must control at least one SDN switch. That is:

$$y_j \leq \sum_{i=1}^{N} z_{ij}, \quad \forall\ j. \tag{4}$$

2) Individual Controller's Processing Ability Constraints: An SDN switch will send a flow request to a controller when it does not know how to process a new flow. The number of flow requests from an SDN switch equals the number of flows traversing the switch, and it should not exceed the controller's processing ability. This can be mathematically written as

$$\sum_{i=1}^{N} (R_i * x_i * z_{ij}) \leq A, \quad \forall\ j,$$

Since both x_i and z_{ij} are binary variables, we substitute (1) into the above nonlinear constraints and reformulate them as the following linear constraints:

$$\sum_{i=1}^{N} (R_i * z_{ij}) \leq A, \quad \forall\ j. \tag{5}$$

3) Upgrade Budget Constraint: The upgrade budget is that the total cost of upgraded switches and deployed controllers is at most $M > 0$. The number of upgraded switches is $\sum_{i=1}^{N} x_i$, and the number of deployed controllers is $\sum_{j=1}^{N} y_j$. Suppose that the cost of an SDN switch is γ $(\gamma \geq 1)$ times of the cost of a controller. Thus, we have

$$\gamma * \sum_{i=1}^{N} x_i + \sum_{j=1}^{N} y_j \leq M. \tag{6}$$

C. Objective Functions

There are two objectives in our problem. The first one is to maximize the total number of flows by updating legacy switches to SDN switches:

$$obj_1 = \sum_{i=1}^{N} (R_i * x_i). \tag{7}$$

The second one is to minimize the total propagation delay of flow requests between SDN switches and controllers by deploying the controllers and mapping the SDN switches to the controllers. We use D_{ij} $(D_{ij} \geq 0)$ to denote the propagation delay between SDN switch s_i and controller c_j, which is proportional to their distance. Thus, we formulate the total propagation delay as follows:

$$obj_2 = \sum_{i=1}^{N} \sum_{j=1}^{N} (D_{ij} * z_{ij}). \tag{8}$$

One main reason for upgrading traditional switches to SDN switches is to enjoy the benefit of programmability to flexibly select the route path. Many existing works adopt the first objective as their objectives [14], [15]. The first objective decides the number and location of SDN switches in a network. The second objective is typically used to optimize the network performance for a network with given SDN switches [25], [26]. In a hybrid SDN, the first objective is usually (much) more important than the second objective.

D. Problem Formulation

The goal of our problem is to maximize the number of flows from the SDN switches and minimize the total propagation delay of flow requests between the SDN switches and controllers by smartly upgrading legacy switches to SDN switches, deploying controllers, and mapping the SDN switches to the controllers. In practice, maximizing the number of programmable flows in our objectives has the first priority, and minimizing the total delay between the SDN switches and controllers has the second priority. Hence, we model the Joint Switch Upgrade and Controller Deployment (JSUCD) problem as a two-stage problem. In the first stage, we maximize the total number of programmable flows without considering the delay objective. This can be done by solving the following problem:

$$\begin{aligned} F^* = \max_{x,y,z} \ & \sum_{i=1}^{N} R_i x_i \\ \text{s.t. } & (2)(3)(4)(5)(6), \\ & x_i,\ y_j,\ z_{ij} \in \{0,1\}, \quad \forall\ i,\ \forall\ j. \end{aligned} \tag{P1}$$

With the maximum number of programmable flows obtained from problem (P1), we minimize the total delay by solving the following problem:

$$\begin{aligned}\min_{x,y,z}\ & \sum_{i=1}^{N}\sum_{j=1}^{N} D_{ij} z_{ij} \\ \text{s.t. } & F^* = \sum_{i=1}^{N} R_i x_i, (2)(3)(4)(5)(6), \\ & x_i,\, y_j,\, z_{ij} \in \{0,1\}, \quad \forall\, i,\ \forall\, j. \quad \text{(P2)}\end{aligned}$$

The above two-stage problem formulation needs to solve two problems. An alternative formulation is to combine the two objectives into one objective as follows:

$$\begin{aligned}\max_{x,y,z}\ & \sum_{i=1}^{N} R_i x_i - \lambda \sum_{i=1}^{N}\sum_{j=1}^{N} D_{ij} z_{ij} \\ \text{s.t. } & (2)(3)(4)(5)(6), \\ & x_i,\, y_j,\, z_{ij} \in \{0,1\}, \quad \forall\, i,\ \forall\, j, \quad \text{(P)}\end{aligned}$$

where $\lambda \geq 0$ is a constant number that gives different weights of the two objective terms. In Section IV, we prove that by choosing the parameter λ appropriately problem (P) is equivalent to the two-stage problem.

IV. Problem Analysis

In this section, we prove that under a certain condition, problem (P) is equivalent to the two-stage problem, and analyze the computational complexity of problems (P1),(P2), and (P).

A. Choice of the Parameter λ in Problem (P)

Notice that if $D_{ij} = 0$ for all i and j, problem (P) is equivalent to problem (P2) for any λ. In this case, the objective function in problem (P2) is zero and hence problem (P) is equivalent to the two-stage problem. In the following, we show that even though there are i_0 and j_0 such that $D_{i_0 j_0} \neq 0$, by appropriately choosing the parameter λ, this equivalence still holds.

Proposition 1: Suppose all R_i ($1 \leq i \leq N$) are integers and there are i_0 and j_0 such that $D_{i_0 j_0} \neq 0$. Let d be the greatest common divisor of $R_1, \ldots, R_N$, i.e., $d = max\{\bar{d} \mid \frac{R_i}{\bar{d}} \in \mathbb{Z},\ i = 1, \ldots, N\}$. If

$$0 < \lambda < \frac{d}{\sum_{i=1}^{N} \max_{1\leq j\leq N}\{D_{ij}\}}, \tag{9}$$

then problem (P) is equivalent to the two-stage problem.

Proof: Notice that $\frac{d}{\sum_{i=1}^{N} \max_{1\leq j\leq N}\{D_{ij}\}} > 0$ since $D_{ij} \geq 0$ for all i and j and there exist i_0 and j_0 such $D_{i_0 j_0} > 0$. In the following, we shall use the contradiction argument to show that the solution of problem (P) is also the solution of problems (P1) and (P2).

Let (x^*, y^*, z^*) be an optimal solution of problem (P). Assume that (x^*, y^*, z^*) is not an optimal solution of problem (P1). Then there exists an optimal solution $(\bar{x}, \bar{y}, \bar{z})$ of problem (P1) with $\sum_{i=1}^{N} R_i \bar{x}_i > \sum_{i=1}^{N} R_i x_i^*$. Because R_i, $\bar{x}_i$, and x_i^* are all integers, by the choice of d, we have

$$\sum_{i=1}^{N} R_i \bar{x}_i - \sum_{i=1}^{N} R_i x_i^* \geq d. \tag{10}$$

Since $\bar{x}_i = \sum_{j=1}^{N} \bar{z}_{ij} \leq 1$, it follows $\sum_{j=1}^{N} D_{ij}\bar{z}_{ij} \leq \max_{1\leq j\leq N}\{D_{ij}\}\bar{z}_{ij}$, which, together with the choice of λ in (9), further implies that

$$\begin{aligned}\lambda \sum_{i=1}^{N}\sum_{j=1}^{N} D_{ij}\bar{z}_{ij} &\leq \lambda \sum_{i=1}^{N} \max_{1\leq j\leq N}\{D_{ij}\}\bar{z}_{ij} \\ &\leq \lambda \sum_{i=1}^{N} \max_{1\leq j\leq N}\{D_{ij}\} < d. \end{aligned} \tag{11}$$

Combining (10) with (11), we obtain

$$\begin{aligned}\sum_{i=1}^{N} R_i \bar{x}_i - \lambda \sum_{i=1}^{N}\sum_{j=1}^{N} D_{ij}\bar{z}_{ij} &> \sum_{i=1}^{N} R_i x_i^* + d - d \\ &= \sum_{i=1}^{N} R_i x_i^* \geq \sum_{i=1}^{N} R_i x_i^* - \lambda \sum_{i=1}^{N}\sum_{j=1}^{N} D_{ij} z_{ij}^*,\end{aligned}$$

where the last inequality follows from $D_{ij} \geq 0$ for all i and j and $\lambda > 0$. However, this contradicts the fact that (x^*, y^*, z^*) is an optimal solution of problem (P). Hence (x^*, y^*, z^*) is also optimal for problem (P1). Let F^* be the optimal value of problem (P1). Since adding the constraint $F^* = \sum_{i=1}^{m} R_i x_i$ to problem (P) does not change its optimal solution set, then it follows that the optimal solution of problem (P) is also optimal for problem (P2). Similarly, one can show that the optimal solution of problem (P2) is also optimal for problem (P). This completes our proof. □

Proposition 1 shows that one-stage problem (P) with λ satisfying the condition in (9) is equivalent to the two-stage problem, and hence we can solve problem (P) to obtain the desired result instead of solving two problems (P1) and (P2).

B. Complexity Analysis

In this subsection, we analyze the computational complexity of problems (P1), (P2), and (P).

Proposition 2: Problems (P1), (P2), and (P) are all strongly NP-hard.

Proof: We only prove the strong NP-hardness of problem (P1) since the proof of the other two problems is similar. This can be done by establishing a polynomial-time reduction from the 3-partition problem, which is strongly NP-complete [28], [29]. Next, we first introduce the 3-partition problem.

Given a finite set $\mathcal{S}$ of $3m$ elements, a bound $B \in \mathbb{Z}_+$, and a size $a_i \in \mathbb{Z}_+$ for the i-th element with $\frac{B}{4} < a_i < \frac{B}{2}$ and $\sum_{i=1}^{3m} a_i = mB$, can $\mathcal{S}$ be partitioned into m disjoint sets $\mathcal{S}_1, \ldots, \mathcal{S}_m$ such that $\sum_{i\in\mathcal{S}_j} a_i = B$, $1 \leq j \leq m$?

Given any instance of the 3-partition problem, we construct an instance of problem (P1) as follows:

set $M = 4m$, $A = B$, $\gamma = 1$, and N to be any integer satisfying $N \geq 3m$;

set $R_i = a_i$ for $i \in \{1, \ldots, 3m\}$ and $R_i = 0$ for $i \in \{3m+1, \ldots, N\}$.

By construction, the objective function in problem (P1) reduces to $\sum_{i=1}^{3m} a_i x_i$. It is easy to see that, for this constructed instance of problem (P1), there exists an optimal solution (x, y, z) such that $x_i = 0$ for $i \in \{3m+1, \ldots, N\}$ and $z_{ij} = 0$ for $i \in \{3m+1, \ldots, N\}$, $j \in \{1, \ldots, N\}$.

Hence the constructed instance of problem (P1) can be written as

$$\begin{aligned}
&\max_{x,y,z} \sum_{i=1}^{3m} a_i x_i \\
&\text{s.t. (2)(3)(4)}, \\
&\sum_{i=1}^{3m} a_i z_{ij} \le B,\ 1 \le j \le N, \\
&\sum_{i=1}^{3m} x_i + \sum_{j=1}^{N} y_j \le 4m, \\
&x_i,\ y_j,\ z_{ij} \in \{0,1\}, \quad 1 \le i \le 3m,\ 1 \le j \le N. \qquad \text{(P1')}
\end{aligned}$$

In the following, we will show that the answer to the given instance of the 3-partition problem is true if and only if the optimal value of problem (P1') is mB.

Suppose that $\mathcal{S}$ can be partitioned into m disjoint sets $\mathcal{S}_1, \ldots, \mathcal{S}_m$ such that $\sum_{i \in \mathcal{S}_j} a_i = B$ for all $j \in \{1, \ldots, m\}$. We construct a point $(\bar{x}, \bar{y}, \bar{z})$ by setting

$\bar{x}_i = 1$ for $i \in \{1, \ldots, 3m\}$;
$\bar{y}_j = 1$ for $j \in \{1, \ldots, m\}$ and $\bar{y}_j = 0$ for $j \in \{m+1, \ldots, N\}$;
$\bar{z}_{ij} = 1$ for $i \in \mathcal{S}_j$, $j \in \{1, \ldots, m\}$ and $\bar{z}_{ij} = 0$ for the others.

Clearly, $(\bar{x}, \bar{y}, \bar{z})$ is a feasible solution of problem (P1') with $\sum_{i=1}^{3m} a_i \bar{x}_i = mB$. As $\sum_{i=1}^{3m} a_i \bar{x}_i \le \sum_{i=1}^{3m} a_i = mB$, we know the optimal value of problem (P1') is mB.

Now suppose that the optimal value of problem (P1') is mB and the corresponding solution is $(\bar{x}, \bar{y}, \bar{z})$. Then $\sum_{i=1}^{3m} a_i \bar{x}_i = mB$. This, together with the fact that $\sum_{i=1}^{3m} a_i = mB$, implies $\sum_{i=1}^{3m} \bar{x}_i = 3m$.

Then it must follow

$$\sum_{j=1}^{N} \bar{y}_j = m. \tag{12}$$

Otherwise, by $\sum_{i=1}^{3m} \bar{x}_i + \sum_{j=1}^{N} \bar{y}_j \le 4m$, we know $\sum_{j=1}^{N} \bar{y}_j < m$. However, this is impossible since

$$\sum_{i=1}^{3m} a_i \bar{x}_i = \sum_{i=1}^{3m} a_i \sum_{j=1}^{N} \bar{z}_{ij} = \sum_{j=1}^{N} \sum_{i=1}^{3m} a_i \bar{z}_{ij} \le \sum_{j=1}^{N} \bar{y}_j B < mB,$$

where the first equality follows from $\bar{x}_i = \sum_{j=1}^{N} \bar{z}_{ij}$ in problem (P1') and the first inequality follows from $\bar{z}_{ij} \le \bar{y}_j$ and $\sum_{i=1}^{3m} a_i \bar{z}_{ij} \le B$ in problem (P1'). By (12), without loss of generality, we can assume that $\bar{y}_j = 1$ for $j \in \{1, \ldots, m\}$ and $\bar{y}_j = 0$ for $j \in \{m+1, \ldots, N\}$. We further show

$$\sum_{i=1}^{3m} a_i \bar{z}_{ij} = B, \quad 1 \le j \le m. \tag{13}$$

Otherwise, we have $\sum_{i=1}^{3m} a_i \bar{z}_{ij} < B$ for some j. From $\bar{y}_j = 0$ for $j \in \{m+1, \ldots, N\}$ and $\bar{z}_{ij} \le \bar{y}_j$, we know $\bar{z}_{ij} = 0$ for $i \in \{1, \ldots, N\}$, $j \in \{m+1, \ldots, N\}$. Using this, we have $\sum_{i=1}^{3m} a_i \bar{x}_i = \sum_{j=1}^{N} \sum_{i=1}^{3m} a_i \bar{z}_{ij} = \sum_{j=1}^{m} \sum_{i=1}^{3m} a_i \bar{z}_{ij} < mB$, which contradicts the fact that the optimal value of problem (P1') is mB. Now, let $\mathcal{S}_j = \{i \,|\, \bar{z}_{ij} = 1,\ 1 \le i \le 3m\}$ for $j \in \{1, \ldots, m\}$. Then it follows from (13) that

$$\sum_{i \in S_j} a_i = \sum_{i=1}^{3m} a_i \bar{z}_{ij} = B.$$

From $\bar{x}_i = \sum_{j=1}^{N} \bar{z}_{ij}$, we have $\mathcal{S}_{j_1} \cap \mathcal{S}_{j_2} = \emptyset$ for any j_1, $j_2 \in \{1, \ldots, m\}$ with $j_1 \ne j_2$. Furthermore, combining (13) with $\frac{B}{4} < a_i < \frac{B}{2}$, it follows $\sum_{i=1}^{3m} \bar{z}_{ij} = 3$, or equivalently $|\mathcal{S}_j| = 3$, $1 \le j \le m$. This, together with the fact that $\mathcal{S}_1, \ldots, \mathcal{S}_m$ are disjoint, indicates

$$\mathcal{S}_1 \cup \cdots \cup \mathcal{S}_m = \{1, \ldots, 3m\}.$$

Hence the answer to the given instance of the 3-partition problem is true.

Finally, the above transformation can be done in polynomial time. Since the 3-partition problem is strongly NP-complete, we conclude that problem (P1) is strongly NP-hard. □

Proposition 2 implies that, unless P=NP, there are no polynomial time algorithms, which can solve problem (P) to global optimality. Thus, we should develop efficient algorithms for approximately solving the problem especially when the dimension of the problem is large.

C. More Insight Into Intrinsic Difficulty of Problem (P)

In the above subsection, we basically show that the problem of maximizing the total number of programmable flows is NP-hard. In this subsection, we study another two special cases of problem (P), i.e., the problem of minimizing the total propagation delay between controllers and SDN switches if all switches can be upgraded to SDN switches and all flows can be programmable, and show that they are still NP-hard. The analysis of the second case shows that there probably does not exist a constant-ratio approximation algorithm for problem (P). These analysis results provide more insight into the intrinsic difficulty of problem (P).

To simplify the following analysis, we replace constraint (5) in problem (P) with the following constraint

$$\sum_{i=1}^{N} (R_i * z_{ij}) \le A * y_j, \quad \forall\ j. \tag{14}$$

Due to the binary nature of variables y and z and constraint (2), this replacement does not change the solution of the problem.

1) Case 1: In this case, we assume: (i) the processing capacity of the controller is large enough to control all switches' requests, and (ii) the upgrade budget is large enough to upgrade all switches to SDN switches. Mathematically, the above two assumptions can be written as (i) $A > \sum_{i=1}^{N} R_i$ and (ii) $M - \gamma N \ge 1$. Under these two assumptions, we can reduce problem (P) via the following three steps. First, combining assumption (i) with constraint (2), we can see that constraint (14) becomes redundant for problem (P), and thus we can

remove it. Second, the above assumptions (i) and (ii) can guarantee $x_i = 1$ for all $i \in [1, N]$ in the optimal solution of problem (P). Hence, constraint (3) changes into

$$\sum_{j=1}^{N} z_{ij} = 1, \quad \forall\, i, \tag{15}$$

and constraint (6) changes into

$$\sum_{j=1}^{N} y_j \leq M - \gamma N. \tag{16}$$

Third, we can remove constraint (4) from problem (P). We argue this as follows: Let us suppose the optimal solution of problem (P) without constraint (4) is $(\bar{x}, \bar{y}, \bar{z})$ with $\bar{y}_j > \sum_{i=1}^{N} \bar{z}_{ij}$ for some $j \in \mathcal{J} \subseteq [1, N]$. By setting $\bar{y}_j = 0$ for all $j \in \mathcal{J}$, we obtain a feasible solution for problem (P), which yields the same objective value as that at $(\bar{x}, \bar{y}, \bar{z})$. Combining the above three steps together, we can reduce problem (P) to the problem of selecting appropriate controllers to control all the SDN switches to minimize the total propagation delay as follows:

$$\begin{aligned} \min_{y,z} \ & \sum_{i=1}^{N}\sum_{j=1}^{N} D_{ij} z_{ij} \\ \text{s.t. } & (2)(15)(16), \\ & y_j,\, z_{ij} \in \{0, 1\}, \quad \forall\, i,\ \forall\, j. \end{aligned}$$

The above problem is called *p-Median Problem* and is NP-hard [30]. Thus, problem (P) in this special case is also NP-hard.

2) Case 2: In the case, we assume: (i) all switches have the same number of flows and (ii) the upgrade budget is large enough to upgrade all switches to SDN switches and to deploy controllers to control all upgraded switches. These two assumptions can be written as (i) $R_1 = \cdots = R_N \triangleq R$ and (ii) $(M - \gamma N)\lfloor \frac{A}{R} \rfloor \geq N$, where $\lfloor \cdot \rfloor$ is the floor operator. Substituting assumption (i) into constraint (14), we obtain

$$\sum_{i=1}^{N} z_{ij} \leq \left\lfloor \frac{A}{R} \right\rfloor y_j, \quad \forall\, j. \tag{17}$$

It is not difficult to argue that assumptions (i) and (ii) guarantee that $x_i = 1$ for all $i \in [1, N]$ in the optimal solution of (P). Following steps 2 and 3 of case 1, we can reformulate problem (P) in this special case as follows:

$$\begin{aligned} \min_{y,z} \ & \sum_{i=1}^{N}\sum_{j=1}^{N} D_{ij} z_{ij} \\ \text{s.t. } & (2)(15)(16)(17), \\ & y_j,\, z_{ij} \in \{0, 1\}, \quad \forall\, i,\ \forall\, j. \end{aligned}$$

The above problem is called *Uniform Capacitated p-Median Problem*. It is a NP-hard problem, and existing studies do not find a constant-ratio approximation algorithm for it [31]. Our problem (P) is more general than the above problem. Based on the above analysis, we can conclude that there probably does not exist a good approximation algorithm for our problem (P).

V. Problem Solution

In this section, we propose an exact solution for solving the problem of small networks and an efficient heuristic algorithm for solving the problem of large networks.

A. Exact Solution

Typically, we can use existing integer program solvers to obtain problem (P)'s optimal solution. Here we use GUROBI solver [32] for solving our problem. GUROBI uses a branch-and-cut [33] framework and is recognized as one of the fastest integer program solvers [34]. The branch-and-cut framework usually uses Linear Programming (LP) relaxation to obtain an upper bound. However, the LP relaxation is usually very weak [35], and thus it is difficult for the solver to quickly solve the integer programming problem. In our experiments, we also observe the weakness of the LP relaxation of problem (P). To accelerate the solution process, we propose a better (re)formulation by strengthening some constraints and adding some valid inequalities by exploiting problem (P)'s structure.

1) Strengthening Constraints: First, we strengthen constraint (5) as (14). Compared to constraint (5), in (14) we use a tighter upper bound $A * y_j$, which improves the objective value of the LP relaxation of problem (P) and thus reduces the solution time of the branch-and-cut algorithm in GUROBI. The details can be found in [33].

2) Adding Valid Inequalities: Generating efficient cutting planes is a key step in the branch-and-cut framework. In the mixed integer problems, one novel technique is to aggregate multiple constraints together to generate redundant but efficient constraints for the problem. This is because, in the mixed integer problems, some redundant constraints could be used as base constraints to generate cuts and may accelerate the solution process of the branch-and-cut framework [36]. Modern solvers can generate cuts automatically but usually do not consider the problem's structure. By exploiting the structure of problem (P), below we add some aggregated constraints to problem (P) to help the solver efficiently generate cuts.

Adding all the constraints in (14) and using the constraints (3), we obtain

$$\sum_{i=1}^{N} (R_i * x_i) \leq A * \sum_{j=1}^{N} y_j. \tag{18}$$

Combining (18) with (6), we obtain the following inequality

$$\sum_{i=1}^{N} (R_i + \gamma A) * x_i \leq A * M. \tag{19}$$

The inequalities (18) and (19) are redundant for the LP relaxation problem of (P), but they can be used as base constraints to help the solver to find some knapsack cuts [37] and further accelerate the solution process. The similar technique has been used for the problem of single-source capacitated facility location in [38].

TABLE I
NOTATIONS

Notation	Meaning
N	the number of switches
M	the upgrade budget
A	the processing capacity of a controller
i	the index of a switch, $i \in [1, N]$
j	the index of a controller, $j \in [1, N]$
γ	the cost ratio of an SDN switch to a controller
$\mathcal{W}$	the set of weights, $\mathcal{W} = \{\{w_{11}, ..., w_{1j}, ..., w_{1N}\}, ..., \{w_{i1}, ..., w_{ij}, ..., w_{iN}\}, ..., \{w_{N1}, ..., w_{Nj}, ..., w_{NN}\}, i, j \in [1, N]\}$
$\mathcal{R}$	the set of the number of flows in each switch, $\mathcal{R} = \{R_i, i \in [1, N]\}$
$\mathcal{X}$	the set of updated switches, $\mathcal{X} = \{i \in [1, N] \mid x_i = 1\}$
$\mathcal{Y}$	the set of deployed controllers, $\mathcal{Y} = \{j \in [1, N] \mid y_j = 1\}$
$\mathcal{Z}$	the set of the mapping relationship between upgraded switches and controllers, $\mathcal{Z} = \{(i, j) \in [1, N] \times [1, N] \mid z_{ij} = 1\}$

Substituting (3) into the objective of problem (P), we have

$$\begin{aligned} obj &= \sum_{i=1}^{N} R_i \sum_{j=1}^{N} z_{ij} - \lambda \sum_{i=1}^{N} \sum_{j=1}^{N} D_{ij} z_{ij} \\ &= \sum_{i=1}^{N} \sum_{j=1}^{N} (R_i - \lambda D_{ij}) z_{ij}. \end{aligned}$$

Based on the above analysis, we reformulate our problem as the final problem:

$$\begin{aligned} &\max_{x,y,z} \sum_{i=1}^{N} \sum_{j=1}^{N} \omega_{ij} z_{ij} \\ &\text{s.t. } (2)(3)(4)(6)(14)(18)(19), \\ &\quad x_i,\ y_j,\ z_{ij} \in \{0, 1\}, \quad \forall\ i,\ \forall\ j, \quad \text{(P')} \end{aligned}$$

where x_i, y_j, z_{ij} are design variables and

$$\omega_{ij} = R_i - \lambda D_{ij}. \tag{20}$$

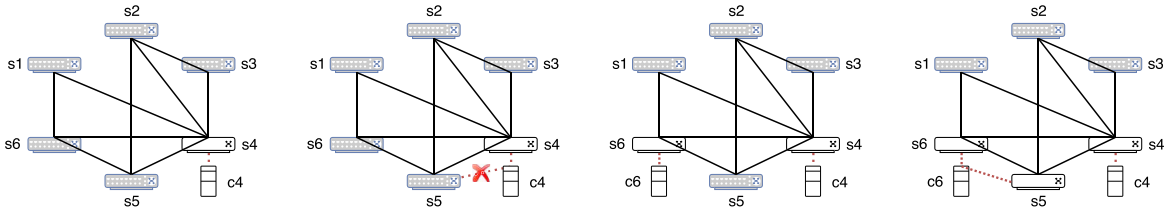
We will illustrate the effectiveness and efficiency of the new formulation with simulation results in Section VI-C.

B. Heuristic Algorithm

The new formulation helps the optimization solver to accelerate the solution process in small networks. However, it still requires a very long time or sometimes is impossible for the solver to find a feasible solution for the problem of large networks. In this section, we propose a heuristic algorithm for solving the problem to achieve the tradeoff between the performance and the time complexity. The heuristic algorithm is based on formulation (P').

The intrinsic difficulty of our problem lies in the interrelationship of the selection of switches to upgrade, the selection of controllers to deploy, and the switch-controller mappings. Due to the interrelationship complexity of the three variables, we cannot change them at the same time. Based on our previous analysis, the mapping variables are more crucial (than switch update and controller deployment variables) since one mapping variable potentially can determine the other two variables. In the following part, we propose the MapFirst algorithm that determines the variables in the order of mappings, switches, and controllers.

The notations used in the algorithm are listed in Table I. The idea of our proposed algorithm, MapFirst, is to first select a switch-controller mapping in the descending order of their importance/weights and then tests whether building the mapping will satisfy the upgrade budget constraint: if yes, the switch and the controller in the mapping is selected; otherwise, a new mapping is tested. The procedure is terminated until there is no budget to build any mapping. Details of MapFirst are summarized in Algorithm 1. In line 1, at the beginning of the algorithm, the sets $\mathcal{X}$, $\mathcal{Y}$, and $\mathcal{Z}$ are set to be empty since no switches are upgraded, no controllers are deployed, and there are no mappings between switches and controllers. In line 2, we generate vector $\mathcal{Z}^{vec} = \{z_l^{vec}, l \in [1, N * N]\}$. We first relax the binary variables in problem (P') to continuous variables in [0,1], and get the LP relaxation solution $\mathcal{Z}^*$ of problem (P'). We then sort the solution $\mathcal{Z}^*$ in the descending order to get vector $\mathcal{Z}^{vec}$. The sorting operation enables us to test the mapping variables based on their probabilities. Next, we use our customized rounding technique to find the result by sequentially testing each $z_l^{vec} \in \mathcal{Z}^{vec}$. In line 4, we get z_l^{vec}'s corresponding switch index i_0 and controller index j_0. Lines 5-7 guarantee that we do not test a switch if it is already upgraded. In lines 8-18, we test the mapping between switch s_{i_0} and controller c_{j_0}. If the mapping satisfies the constraints of problem (P'), we upgrade the switch and deploy the controller at two specific conditions: (1) in lines 9-12, controller c_{j_0} is already deployed, and we only upgrade switch s_{i_0} when the remaining upgrade budget is γ or more; (2) in lines 13-16, controller c_{j_0} is not deployed, and we upgrade switch s_{i_0} and deploy controller c_{j_0} when the remaining upgrade budget is $\gamma + 1$ or more. If either of the two conditions is satisfied, we set the mapping between switch s_{i_0} and controller c_{j_0}.



(a) Test mapping M_{s_4,c_4} and select it. (b) Test mapping M_{s_5,c_4} and not select it. (c) Test mapping M_{s_6,c_6} and select it. (d) Test mapping M_{s_5,c_6} and select it.

Fig. 2. An example of applying MapFirst to solve the problem. The weights of the mappings satisfy $z_{s_4,c_4} > z_{s_5,c_4} > z_{s_6,c_6} > z_{s_6,c_5}$.

Algorithm 1 MapFirst algorithm

Input: N, M, A, γ, $\mathcal{W}$, $\mathcal{R}$;
Output: $\mathcal{X}$, $\mathcal{Y}$, $\mathcal{Z}$;
1: $\mathcal{X} = \emptyset$, $\mathcal{Y} = \emptyset$, $\mathcal{Z} = \emptyset$;
2: Generate vector $\mathcal{Z}^{vec} = \{z_l^{vec}, l \in [1, N * N]\}$ by solving the LP relaxation of problem (P') and sorting the results in the descending order;
3: **for** $z_l^{vec} \in \mathcal{Z}^{vec}$ **do**
4: find switch index i_0 and controller index j_0 of z_l^{vec};
5: **if** $i_0 \in \mathcal{X}$ **then**
6: continue;
7: **end if**
8: **if** $\mathcal{Z} \cup (i_0, j_0)$, $\mathcal{X} \cup i_0$, and $\mathcal{Y} \cup j_0$ satisfy the constraints in (P') **then**
9: **if** $j_0 \in \mathcal{Y}$ and $|\mathcal{X}| * \gamma + |\mathcal{Y}| + \gamma \leq M$ **then**
10: // controller c_{j_0} is already deployed, just upgrade switch s_{i_0}
11: // map switch s_{i_0} to controller c_{j_0}
12: $\mathcal{X} \leftarrow \mathcal{X} \cup i_0$, $\mathcal{Z} \leftarrow \mathcal{Z} \cup (i_0, j_0)$;
13: **else if** $j_0 \notin \mathcal{Y}$ and $|\mathcal{X}| * \gamma + |\mathcal{Y}| + \gamma + 1 \leq M$ **then**
14: // deploy controller c_{j_0} and upgrade switch s_{i_0}
15: // map switch s_{i_0} to controller c_{j_0}
16: $\mathcal{Y} \leftarrow \mathcal{Y} \cup j_0$, $\mathcal{X} \leftarrow \mathcal{X} \cup i_0$, $\mathcal{Z} \leftarrow \mathcal{Z} \cup (i_0, j_0)$;
17: **end if**
18: **end if**
19: **if** $M - \gamma < |\mathcal{X}| * \gamma + |\mathcal{Y}|$ **then**
20: break;
21: **end if**
22: **end for**
23: return $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$;

In lines 19-21, if the remaining budget is less than γ, then we cannot upgrade any switch, and the algorithm returns the result and stops.

Fig. 2 shows an example of applying MapFirst to solve the problem. The mappings are tested in the decreasing order of the mappings' weights. In the figure, the mapping M_{s_4,c_4} is first tested. Since the mapping satisfies the constraints in problem (P'), controller c_4, switch s_4 and the mapping M_{s_4,c_4} are selected. Then, the mapping M_{s_5,c_4} is tested but is not selected because the mapping does not satisfy the processing capacity constraint. Similarly, mappings M_{s_6,c_6} and M_{s_5,c_6} are tested one by one. If building one mapping does not violate the upgrade budget, the related switch, controller, and the mapping itself are selected.

C. Worst-Case Complexity of MapFirst

As shown in Algorithm 1, MapFirst has two main steps: step 1 solves the LP relaxation and gets the weights by sorting the returned solution of the LP relaxation; step 2 generates a binary solution with a customized rounding technique. An LP can be solved in $\mathcal{O}(n^3 * L)$ arithmetic operations by the interior-point methods, where n is the number of variables and L is the length of the input data of the problem [39]. Our problem has in total N^2+2N variables, and the computational complexity of solving our LP relaxation is $\mathcal{O}(N^6 * L)$. Sorting the returned solution of the LP relaxation takes $\mathcal{O}(N^2 \log N^2)$ operations, and the customized rounding procedure runs at most N^2 iterations. In summary, the dominant computational cost of MapFirst is to solve one LP relaxation, and its worst-case complexity is $\mathcal{O}(N^6 * L)$. In sharp contrast, the branch-and-cut framework in the worst case needs to solve an exponential number of LP relaxations. Therefore, the worst-case complexity of MapFirst is significantly smaller than that of the branch-and-cut framework.

D. A Polynomial Time Solvable Case

Our analysis in Sections IV-B and IV-C shows that problem (P) is NP-hard, and there probably does not exist a constant-ratio approximation algorithm for it. Therefore, our proposed MapFirst algorithm generally does not have a (constant-ratio approximation) performance guarantee. In this subsection, we consider a special case of problem (P), where each controller can control at most one SDN switch (i.e., $R_i + R_j > A$ for all $i \neq j$), and show that MapFirst is guaranteed to find the optimal solution of problem (P) in this special case.

Proposition 3: If $R_i + R_j > A$ for all $i \neq j$, problem (P) can be solved (to globally optimality) in $\mathcal{O}(N^3)$, and MapFirst is guaranteed to find the optimal solution of problem (P).

Proof: Without loss of generality, we assume $0 < R_i \leq A$ for all $i \in [1, N]$. Combining this assumption with the assumption $R_i + R_j > A$ for all $i \neq j$, constraint (14) reduces

to $\sum_{i=1}^{N} z_{ij} \leq y_j$, which, together with (4), implies

$$y_j = \sum_{i=1}^{N} z_{ij}, \quad \forall\ j. \tag{21}$$

By substituting (5) and (21) into (6), we have

$$\sum_{i=1}^{N}\sum_{j=1}^{N} z_{ij} \leq \Big\lfloor \frac{M}{\gamma+1} \Big\rfloor. \tag{22}$$

From (21), we can simply remove constraint (2) from problem (P) and reformulate it as follows:

$$\begin{aligned} \max_{z}\ & \sum_{i=1}^{N}\sum_{j=1}^{N} \omega_{ij} z_{ij} \\ \text{s.t.}\ & \sum_{j=1}^{N} z_{ij} \leq 1,\ \forall\ i, \sum_{i=1}^{N} z_{ij} \leq 1, \forall\ j, \\ & (22),\ z_{ij} \in \{0,1\}, \quad \forall\ i,\ \forall\ j. \end{aligned} \tag{23}$$

Due to $R_i > 0$ for all $i \in [1, N]$, the definition of w_{ij} in (20), and the selection of λ in (9), we know $\omega_{ij} > 0$ for all $i,\ j \in [1, N]$. Hence, inequality (22) can be rewritten as an equality, and problem (23) is a *k-cardinality assignment* problem. It has been shown in [40] that the LP relaxation of problem (23) is tight, i.e., solving the linear relaxation of problem (23) with an appropriate method (e.g., the simplex method) returns a binary solution. Furthermore, using the *primal algorithm* presented in [40], we can solve problem (23) with $\mathcal{O}(N^3)$ complexity. The proof is completed. □

The above proposition shows that, if each controller can control at most one SDN switch, after some preprocessing steps, the LP relaxation in MapFirst is tight, and MapFirst can return an optimal solution of problem (P).

VI. Simulation Results

A. Simulation Setup

In our simulation, we choose some backbone topologies from Topology Zoo [27] to evaluate the performance of our proposed solution. In the topologies, each node has a latitude and a longitude. Since the propagation delay of a flow request is usually proportional to the distance from a switch to a controller, we use the distance between two nodes to represent the propagation delay between an SDN switch and a controller. We use $M\_percent$ to denote the ratio of the given upgrade budget to the cost of upgrading all switches in the network. In our simulation, $M\_percent$ changes from 5% to 50%. We follow the assumption in [14], [15] that the number of programmable flows in an SDN switch is proportional to the number of its links. In practice, we can analyze the traffic history at each switch to get the real traffic statistic. We have analyzed more than 50 topologies in Topology Zoo and find that most nodes have two or three links. We set the normalized processing ability of a controller $A = 50$, and thus one controller is able to control at least ten SDN switches on average. Note that our problem can take into consideration of heterogeneous controllers by setting the processing abilities of different controllers with different values. The recommended system requirement of one OpenDayLight controller instance is 8 Cores, 8G RAM and 64GB storage [41], and the resilient three-controller deployment requires at least three physical servers and costs about $2000. One typical SDN switch is about $8000. Hence, we set the cost ratio between an SDN switch and one controller $\gamma = 4$.

B. Compared Algorithms

1) *Optimal:* the optimal solution of problem (P') that maximizes the number of programmable flows and minimizes the total propagation delay between upgraded SDN switches and controllers. We solve the problem by using GUROBI [32].
2) *FlowOnly:* the optimal solution of problem (P1) that only maximizes the number of programmable flows. Problem (P1) is also solved by using GUROBI [32].
3) *MapFirst:* we first use GUROBI [32] to solve the LP relaxation of problem (P'), and then use a customized rounding technique to sequentially determine the variables in the order of mappings, switches, and controllers. The details can be found in Section V-B.
4) *WeightFirst:* this algorithm is similar to MapFirst, but the key difference is that it greedily tests and picks the switch-controller mapping in the descending order of weights $\{w_{ij}\}, i, j \in [1, N]$.

C. Effectiveness of Formulation (P')

We test the effectiveness of new formulation (P') under different topologies from Topology Zoo [27]. We set a time limit of 3600 seconds for the branch-and-cut algorithm in GUROBI (i.e., we terminate the algorithm if it does not find the solution within 3600 seconds) and set $M\_percent = 50\%$ for each topology. Table II summarizes the computational results. We can see from the table that: (1) for the problem instances Cogentco and Condensed_west_europe, GUROBI can successfully solve the new formulation (P') within the given time limit but fail to solve the original formulation (P); (2) for the problem instances Colt, GtsCe, and Condensed, GUROBI can successfully solve both problem formulations within the given time limit but solving formulation (P') takes significantly less time than solving formulation (P). These simulation results clearly show the effectiveness of formulation (P'), i.e., the newly added constraints (14), (18), and (19) indeed work and significantly accelerate the solution process.

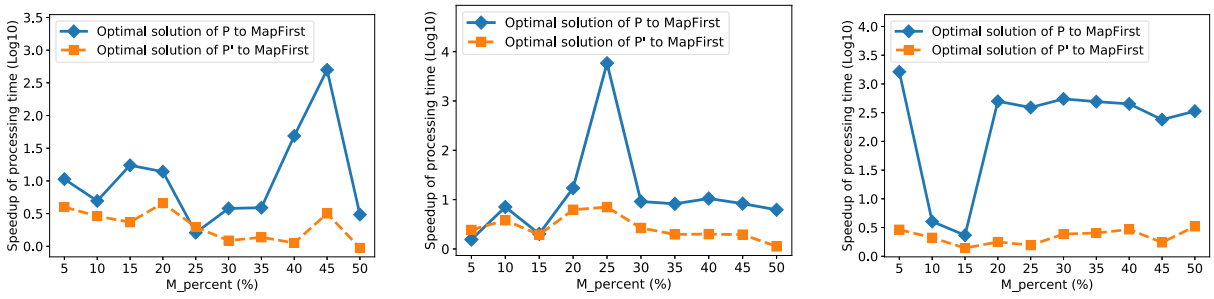
D. Performance of CPU Time

In the rest of this section, we focus on three topologies Att, Cernet, and Cogentco from Topology Zoo [27]. More specifically, Att is a small topology with 25 nodes and 57 links, Cernet is a medium-size topology with 41 nodes and 59 links, and Cogentco is a large topology with 197 nodes and 245 links. All of our simulations below are performed on these three topologies.

We use the ratio of the CPU time of an algorithm to that of MapFirst as the metric to measure the efficiency of the algorithm. Fig. 3 shows the results, where y-axis is in the

TABLE II
COMPUTATIONAL RESULTS OF (A) ORIGINAL FORMULATION (P), AND (B) FORMULATION (P').

Topology name	Topology info		Elapsed CPU time (seconds)	
	# of nodes	# of links	(a)	(b)
Colt	153	185	2046.46	33.04
GtsCe	150	193	1365.47	89.17
Cogentco	197	245	3600.00	419.49
Condensed_west_europe	278	394	3600.00	576.93
Condensed	463	620	2721.63	1013.31



(a) Att (b) Cernet (c) Cogentco

Fig. 3. Speedup of the CPU time in units of $Log10$.

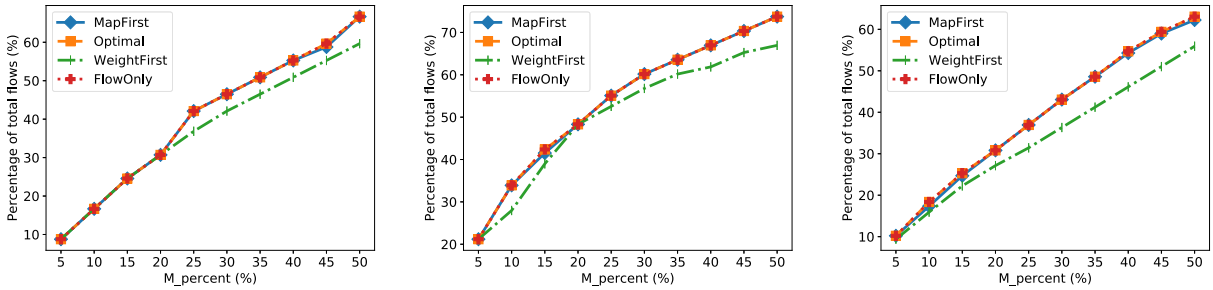
$Log10$ scale. Recall that problem (P) is the original problem, and problem (P') is problem (P) with strengthened constraints and valid inequalities. We can clearly observe from Fig. 3 that MapFirst is the fastest solution in all cases. More specifically, MapFirst is 499 and 5828 times faster than directly using GUROBI to solve problem (P) when M_percent = 45% for Att and M_percent = 25% for Cernet, respectively. In Fig. 3(c), we set a time limit of 3600 seconds for solving (P). We only get the results of M_percent = 10% and 20%, and we use the CPU time limit 3600 seconds as the CPU time of other cases of M_percent. In other words, GUROBI failed to solve problem (P) directly within 3600 seconds. However, for all cases of M_percent, GUROBI can successfully solve problem (P') within the time limit. In fact, from Fig. 3 we can see that it is much more efficient to solve problem (P') than problem (P) in most cases. These simulation results verify that problem (P') is indeed a better formulation than problem (P) because added constraints and inequalities in (P') are very effective to speed up the GUROBI solver, and our proposed MapFirst is effective.

E. Performance of Programmable Flows

Fig. 4 shows the performance of the programmable flows of different algorithms. In all the three topologies, the performance of the four algorithms increases as $M\_percent$ becomes larger. Optimal and FlowOnly are the optimal solutions to maximize the number of programmable flows. We can observe from Fig. 4 that WeighFirst's performance is the worst and MapFirst's performance is very close to that of Optimal and FlowOnly. Recall weight ω_{ij} is equal to $R_i - \lambda D_{ij}$ in WeightFirst. The maximum value of ω_{ij} is R_i when deploying a controller at the location of a selected switch. Since λ is usually a small value, R_i plays the dominant role in ω_{ij} and thus WeightFirst first greedily tests the switch-controller pair based on the descending order of the number of flows in switches. Even though WeightFirst considers the delay in the later tests, it only focuses on a single switch-controller pair without the global view of the problem. In sharp contrast, MapFirst also tests the switch-controller pair but based on the result of the LP relaxation. The LP relaxation considers the entire problem to generate its result that reflects the probability of selecting switch-controller pairs. Therefore, the testing order of switch-controller pairs in MapFirst is more efficient than WeightFirst. This is the reason why MapFirst achieves much better performance than WeightFirst.

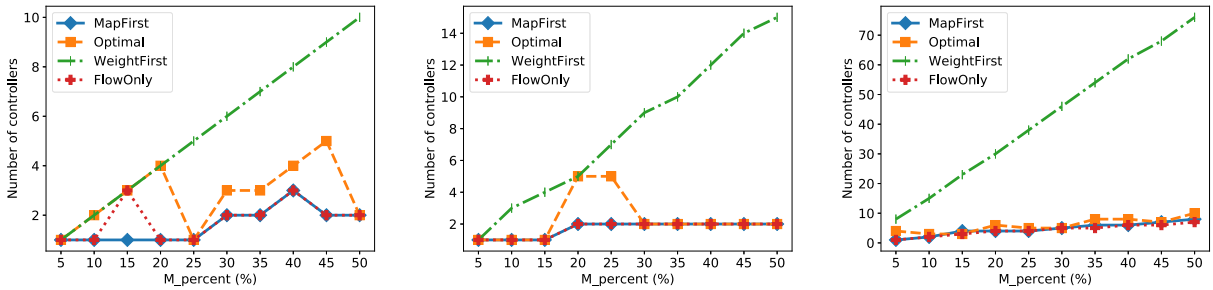
F. Performance of Deployed Controllers

Fig. 5 shows the number of controllers deployed by different algorithms under the three topologies. WeightFirst deploys more controllers than all the other algorithms since it always deploys a controller for each upgraded switch. Thus, under the same upgrade budget, WeightFirst upgrades fewer switches than all the other algorithms. FlowOnly performs the best since it does not consider the propagation delay in its objective function and deploys enough controllers for the upgraded SDN switches. Optimal deploys more switches than FlowOnly and MapFirst. Recall the cost ratio of an SDN switch and a controller is γ. If the upgrade budget is not enough to upgrade a switch, the rest budget can also be used to deploy 1 to



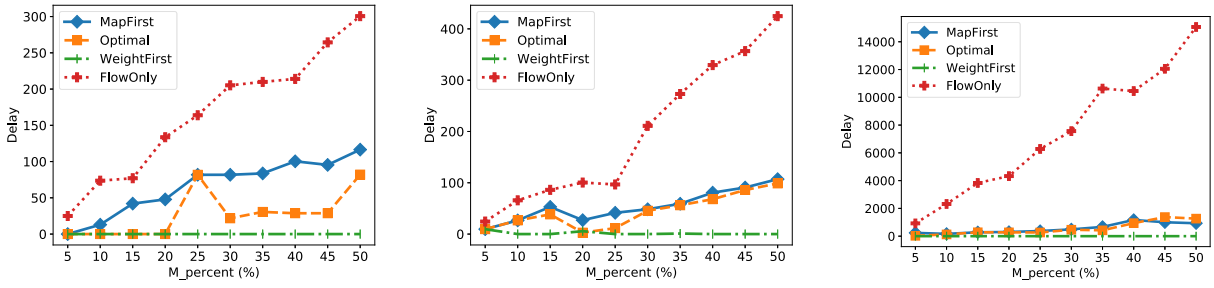
(a) Att (b) Cernet (c) Cogentco

Fig. 4. Number of programmable flows. The higher, the better.



(a) Att (b) Cernet (c) Cogentco

Fig. 5. Number of controllers. The lower, the better.



(a) Att (b) Cernet (c) Cogentco

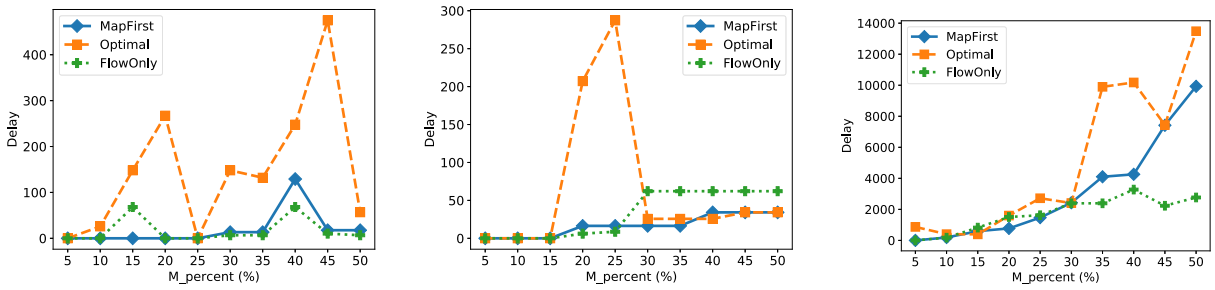
Fig. 6. Propagation delay between SDN switches and controllers. The lower, the better.

$\gamma - 1$ controllers. In this case, MapFirst and Optimal make different decisions: MapFirst does not deploy extra controllers and just stop the program (see lines 19–21 in Algorithm 1), while Optimal will deploy more controllers to minimize the propagation delay between SDN switches and controllers.

G. Performance of the Propagation Delay

Fig. 6 shows the propagation delay between SDN switches and controllers. In all topologies, WeightFirst performs the best as it deploys one controller at each SDN switch in most cases. Among all algorithms, FlowOnly performs the worst because problem formulation (P1) does not consider the propagation delay between SDN switches and controllers. In all the three topologies, Optimal and MapFirst's propagation delays increase as $M\_percent$ increases, but their increasing rate is much lower than FlowOnly's. Compared to MapFirst, Optimal achieves a better propagation delay performance since it deploys more controllers to minimize the propagation delay. From Fig. 5 and 6, we can see that in most cases, when MapFirst and Optimal deploy the same number of controllers, they have the same performance, except two cases $M\_percent = 50\%$ in Fig. 5(a) and $M\_percent = 15\%$ in Fig. 5(b). In these two cases, Optimal performs better than MapFirst even though they deploy the same number of controllers.

Fig. 7 shows the propagation delay between controllers. We do not show the results of WeightFirst because its performance is very bad. We can observe, from the figure, Optimal performs the worst since it does not consider this factor in



(a) Att (b) Cernet (c) Cogentco

Fig. 7. Propagation delay between controllers. The lower, the better.

the problem formulation (An interesting future is to consider the propagation delay in the problem formulation). FlowOnly performs better than Optimal since it deploys much fewer controllers than Optimal (see Fig. 5), and the deployed controllers are near to each other. In most cases, MapFirst has the best performance. This is because MapFirst always deploys the minimum number of controllers. In our problem formulation, we use the budget constraint to implicitly limit the number of controllers. In particular, our objective is to maximize the number of flows, which is equivalent to maximize the number of upgraded SDN switches. Given the upgrade budget, upgrading more switches will reduce the number of controllers to deploy.

H. Performance of Controller Control Ability on Switches

We use the ratio of the number of switches to the number of controllers to measure the controller control ability on switches. If a controller can control many switches, it will simplify the network control. Fig. 8 shows the controller control ability performance of all algorithms. In most cases, MapFirst performs the best since it upgrades the same number of switches as Optimal but deploys fewer controllers.

I. Summary of Simulation Findings

From our simulation results, we can conclude that the following two design principles of MapFirst are crucial: (1) among the three variables (i.e., the selection of upgraded switches, the selection of deployed controllers, and the mappings between SDN switches and controllers), the mapping variables are more important than the other two variables since they reflect the interrelationship between the other two variables, and (2) the solution of the LP relaxation can effectively reveal the importance of the mappings and thus the solution structure of the problem. WeightFirst only takes the first principle into consideration and performs the worst. Optimal performs the best in terms of the two objectives: maximizing the number of programmable flows and minimizing the propagation delay between SDN switches and controllers. However, it will deploy more controllers and has a higher propagation delay among controllers than MapFirst. More importantly, its worst-case complexity is significantly larger than MapFirst and WeightFirst. In summary, compared to Optimal, MapFirst not only achieves a comparable performance of the two objectives with significantly lower complexity but also deploys less number of controllers and thus has a shorter propagation delay among controllers. Considering all performance metrics, MapFirst achieves the best performance. Therefore, to efficiently handle a similar problem with interrelated variables, one can use our design principles.

VII. Related Works

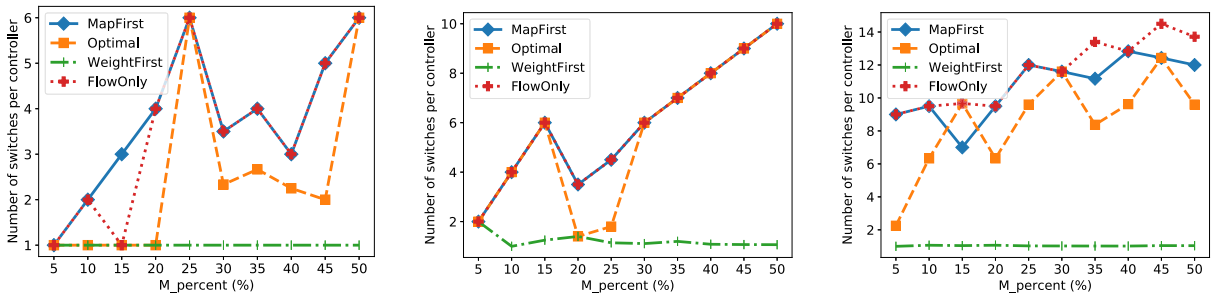
The evolution from legacy networks to SDN is a long journey [42]. Considering SDN still being a fast developing technology, the full deployment of SDN not only requires a huge upgrade cost but also brings unexpected risks. The hybrid SDN is a promising alternative option and is receiving quickly increasing attention from industry and academia. Vissicchio *et al.* [43] first analyzed different models of hybrid SDNs. The research of the hybrid SDN can be categorized into two classes based on the application scenarios:

A. Layer 2 Hybrid SDN

Panopticon [44] proposed an optimization framework to determine the partial SDN deployment and assumed that each flow in the network traverses at least one SDN switch. HybNET [45] designed a configuration mechanism to automatically translate network configurations between legacy networks and SDN networks. Telekinesis [46] manipulated forwarding entries in the legacy switches with a customized flow control primitive and enabled the forwarding flow on a user-defined path rather than the path in the spanning tree. Magneto [4] improved the work in [46] by providing more fine-grained flow controls and quick and stable user-defined path establishment.

B. Layer 3 Hybrid SDN

1) Switch Upgrade: Poularakis *et al.* [14] and Jia *et al.* [15] proposed to incrementally upgrade SDN to maximize the amount of programmable traffic under the given upgrade budget constraint. Xu *et al.* [47] compared the performance of hybrid SDNs by either replacing legacy devices with SDN devices or adding new SDN devices. Caria *et al.* [48] exploited the property of the network topologies and considered the centrality in the network to update switches in a hybrid SDN.



(a) Att (b) Cernet (c) Cogentco

Fig. 8. Ratio of the number of upgraded switches to the number of deployed controllers. The higher, the better.

2) Traffic Engineering: Agarwal *et al.* [7] first formulated an optimization problem for achieving traffic engineering in SDN with the partial deployment of SDN devices. Fibbing [9] introduced a centralized control over distributed IP routing by injecting crafted routing messages via OSPF and enhanced the flexibility, diversity, and reliability of L3 routing. Chu *et al.* [10] designed an approach to fast recover from single link failure while maintaining load-balancing performance for the post-recovery network. Wang *et al.* [11] and Jia *et al.* [12] explored power saving in Layer 3 hybrid SDN. Guo *et al.* [49] proposed to adjust the weights of links and flow split ratio at the SDN nodes to achieve load balancing in a given hybrid SDN. Hong *et al.* [8] proposed to satisfy a variety of traffic engineering goals in the hybrid SDN. However, all the above works did not introduce the real deployment of the SDN control plane and did not consider the impact of the control plane deployment on the hybrid SDN.

C. Multi-Controller Data Plane

1) Controller Deployment: Controller deployment is an important issue in multi-controller research. In WANs, the propagation latency is the critical part of the total latency [25], [26], and some works aim to minimize the propagation latency among controllers and switches [50], [51]. In data center networks, Wang *et al.* [52] proposed to dynamically map switches to controllers to mitigate the load imbalance among controllers and reduce the response time. Wang *et al.* [53] considered the maintenance cost of the controller cluster and assigned controllers to minimize the total cost of controller response time and maintenance on the cluster of controllers.

2) Resiliency: Resiliency is a critical concern for designing the multi-controller data plane. Li *et al.* [54] presented to manage each SDN device with multiple controllers at the cloud to resist Byzantine attacks on controllers and the communication links between controllers and SDN switches. Hu *et al.* [55], [56] designed a fault-tolerant multi-controller control plane that mitigates the performance degradation of the controller chain failure caused by unreasonable slave controller assignment. MORPH [57] is a multi-controller framework, which is tolerant to unavailability failures of SDN controllers and Byzantine failures caused by malicious attacks by efficiently distinguishing and localizing faulty controller instances and appropriately reconfiguring the control plane. In the future, we will consider this factor to extend the work in this paper.

VIII. Conclusion and Future Work

In this paper, we considered the impact of the control plane on the hybrid SDN and proposed to deploy the multi-controller control plane during upgrading a legacy network to a hybrid SDN. We formulated an optimization problem that jointly maximizes the number of flows from SDN switches and minimizes the propagation delay of flow requests between the SDN's control plane and data plane under the given upgrade budget constraint. By carefully analyzing the problem's structure, we proposed efficient solutions to solve the problem. The simulation results based on the real network topologies show the effectiveness and efficiency of our solutions. In the future, we will consider other practical factors in our problem formulation, such as the variation of the traffic pattern, the queuing delay in controllers, and multiple network upgrades.

References

[1] N. McKeown *et al.*, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Apr. 2008.
[2] A. Bates, K. Butler, A. Haeberlen, M. Sherr, and W. Zhou, "Let SDN be your eyes: Secure forensics in data center networks," in *Proc. NDSS Workshop Secur. Emerg. Netw. Technol.*, Feb. 2014, pp. 1–7.
[3] S. Jain *et al.*, "B4: Experience with a globally-deployed software defined wan," in *Proc. ACM SIGCOMM Conf.*, Aug. 2013, pp. 3–14.
[4] C. Jin, C. Lumezanu, Q. Xu, H. Mekky, Z.-L. Zhang, and G. Jiang, "Magneto: Unified fine-grained path control in legacy and openflow hybrid networks," in *Proc. SOSR 17 Symp. SDN Res.*, Apr. 2017, pp. 75–87.
[5] A. Gupta *et al.*, "SDX: A software defined Internet exchange," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 551–562, Oct. 2014.
[6] *As Providers Push NFV/SDN, 3 Key Issues Remain*. Accessed: Apr. 23, 2017. [Online]. Available: https://www.rcrwireless.com/20170423/opinion/readerforum/reader-forum-n%fv-sdn-issues-tag10
[7] S. Agarwal, M. Kodialam, and T. V. Lakshman, "Traffic engineering in software defined networks," in *Proc. IEEE INFOCOM*, Apr. 2013, pp. 2211–2219.
[8] D. K. Hong, Y. Ma, S. Banerjee, and Z. M. Mao, "Incremental deployment of SDN in hybrid enterprise and ISP networks," in *Proc. Symp. SDN Res.*, Mar. 2016, p. 1.
[9] S. Vissicchio, O. Tilmans, L. Vanbever, and J. Rexford, "Central control over distributed routing," in *Proc. ACM SIGCOMM Comput. Commun. Rev.*, 2015, vol. 45, no. 4, pp. 43–56.

[10] C.-Y. Chu, K. Xi, M. Luo, and H. J. Chao, "Congestion-aware single link failure recovery in hybrid SDN networks," in *Proc. IEEE Conf. Comput. Commun.*, May 2015, pp. 1086–1094.

[11] H. Wang, Y. Li, D. Jin, P. Hui, and J. Wu, "Saving energy in partially deployed software defined networks," *IEEE Trans. Comput.*, vol. 65, no. 5, pp. 1578–1592, May 2016.

[12] X. Jia, Y. Jiang, Z. Guo, G. Shen, and L. Wang, "Intelligent path control for energy-saving in hybrid SDN networks," *Comput. Netw.*, vol. 131, pp. 65–76, Feb. 2018.

[13] S. Vissicchio, L. Vanbever, L. Cittadini, G. G. Xie, and O. Bonaventure, "Safe update of hybrid SDN networks," *IEEE/ACM Trans. Netw.*, vol. 25, no. 3, pp. 1649–1662, Jun. 2017.

[14] K. Poularakis, G. Iosifidis, G. Smaragdakis, and L. Tassiulas, "One step at a time: Optimizing SDN upgrades in ISP networks," in *Proc. IEEE Conf. Comput. Commun.*, May 2017, pp. 1–9.

[15] X. Jia, Y. Jiang, and Z. Guo, "Incremental switch deployment for hybrid software-defined networks," in *Proc. IEEE Conf. Local Comput. Netw.*, Nov. 2016, pp. 571–574.

[16] Z. Guo *et al.*, "Improving the performance of load balancing in software-defined networks through load variance-based synchronization," *Comput. Netw.*, vol. 68, pp. 95–109, Aug. 2014.

[17] N. Gude *et al.*, "NOX: Towards an operating system for networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 105–110, 2008.

[18] *ONOS Controller*. Accessed: Mar. 10, 2019. [Online]. Available: https://onosproject.org

[19] *OpenDayLight Controller*. Accessed: Mar. 10, 2019. [Online]. Available: https://www.opendaylight.org

[20] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *Proc. USENIX Annu. Tech. Conf.*, 2014, pp. 305–319.

[21] *Setting Up Clustering*. Accessed: Mar. 10, 2019. [Online]. Available: https://docs.opendaylight.org/en/stable-nitrogen/getting-started-guide/common-features/clustering.html

[22] *ONOS Clustering*. Accessed: Mar. 10, 2019. [Online]. Available: https://wiki.onosproject.org/pages/viewpage.action?pageId=28836788

[23] J. Xie, D. Guo, X. Li, Y. Shen, and X. Jiang, "Cutting long-tail latency of routing response in software defined networks," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 384–396, Mar. 2018.

[24] A. Ksentini, M. Bagaa, T. Taleb, and I. Balasingham, "On using bargaining game for optimal placement of SDN controllers," in *Proc. IEEE Int. Conf. Commun.*, May 2016, pp. 1–6.

[25] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in *Proc. 1st Workshop Hot Topics Softw. Defined Netw.*, Aug. 2012, pp. 7–12.

[26] G. Yao, J. Bi, Y. Li, and L. Guo, "On the capacitated controller placement problem in software defined networks," *IEEE Commun. Lett.*, vol. 18, no. 8, pp. 1339–1342, Aug. 2014.

[27] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The Internet topology zoo," *IEEE J. Sel. Areas Commun.*, vol. 29, no. 9, pp. 1765–1775, Oct. 2011.

[28] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1979.

[29] M. R. Garey and D. S. Johnson, "'Strong' NP-completeness results: Motivation, examples, and implications," *J. ACM*, vol. 25, no. 3, pp. 499–508, Jan. 1978.

[30] O. Kariv and S. L. Hakimi, "An algorithmic approach to network location problems. I: The p-centers," *SIAM J. Appl. Math.*, vol. 37, no. 3, pp. 513–538, Dec. 1979.

[31] S. Li, "On uniform capacitated k-median beyond the natural LP relaxation," *ACM Trans. Algorithms*, vol. 13, no. 2, p. 22, May 2017.

[32] Gurobi Optimization. (2018). *Gurobi Optimizer Optimizer Reference Manual*. [Online]. Available: http://www.gurobi.com

[33] G. Nemhauser and L. Wolsey, *Integer and Combinatorial Optimization*. Hoboken, NJ, USA: Wiley, 1988.

[34] H. D. Mittelmann. *Latest Benchmark Results*. Accessed: Nov. 2018. [Online]. Available: http://plato.asu.edu/talks/informs2018.pdf

[35] T. L. Magnanti, P. Mirchandani, and R. Vachani, "Modeling and solving the two-facility capacitated network loading problem," *Oper. Res.*, vol. 43, no. 1, pp. 142–157, 1995.

[36] T. Achterberg, R. E. Bixby, Z. Gu, E. Rothberg, and D. Weninger, "Presolve reductions in mixed integer programming," *ZIB-Rep.*, 2016, pp. 1–70.

[37] H. Crowder, E. L. Johnson, and M. Padberg, "Solving large-scale zero-one linear programming problems," *Oper. Res.*, vol. 31, no. 5, pp. 803–834, Sep./Oct. 1983.

[38] S. L. Gadegaard, A. Klose, and L. R. Nielsen, "An improved cut-and-solve algorithm for the single-source capacitated facility location problem," *EURO J. Comput. Optim.*, vol. 6, no. 1, pp. 1–27, 2018.

[39] J. Renegar, "A polynomial-time algorithm, based on Newton's method, for linear programming," *Math. Program.*, vol. 40, nos. 1–3, pp. 59–93, 1988.

[40] M. Dell'Amico and S. Martello, "The k-cardinality assignment problem," *Discrete Appl. Math.*, vol. 76, nos. 1–3, pp. 103–121, 1997.

[41] *OpenDayLight Basic Operations Guide*. Accessed: 2018. [Online]. Available: https://wiki.opendaylight.org/view/File:ODL_Basic_Operations_Guide.pptx

[42] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[43] S. Vissicchio, L. Vanbever, and O. Bonaventure, "Opportunities and research challenges of hybrid software defined networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 70–75, Apr. 2014.

[44] D. Levin *et al.*, "Panopticon: Reaping the benefits of incremental SDN deployment in enterprise networks," in *Proc. USENIX Annu. Tech. Conf.*, Jun. 2014, pp. 333–345.

[45] H. Lu, N. Arora, H. Zhang, C. Lumezanu, J. Rhee, and G. Jiang, "HybNET: Network manager for a hybrid network infrastructure," in *Proc. Ind. Track 13th ACM/IFIP/USENIX Int. Middleware Conf.*, Dec. 2013, p. 6.

[46] C. Jin, C. Lumezanu, Q. Xu, Z.-L. Zhang, and G. Jiang, "Telekinesis: Controlling legacy switch routing with openflow in hybrid networks," in *Proc. 1st ACM SIGCOMM Symp. Softw. Defined Netw. Res.*, Jun. 2015, p. 20.

[47] H. Xu, X.-Y. Li, L. Huang, H. Deng, H. Huang, and H. Wang, "Incremental deployment and throughput maximization routing for a hybrid SDN," *IEEE/ACM Trans. Netw.*, vol. 25, no. 3, pp. 1861–1875, Jun. 2017.

[48] M. Caria, T. Das, A. Jukan, and M. Hoffmann, "Divide and conquer: Partitioning OSPF networks with SDN," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage. (IM)*, May 2015, pp. 467–474.

[49] Y. Guo, Z. Wang, X. Yin, X. Shi, and J. Wu, "Traffic engineering in SDN/OSPF hybrid network," in *Proc. IEEE 22nd Int. Conf. Netw. Protocols*, Oct. 2014, pp. 563–568.

[50] T. Hu, Z. Guo, P. Yi, T. Baker, and J. Lan, "Multi-controller based software-defined networking: A survey," *IEEE Access*, vol. 6, pp. 15980–15996, Mar. 2018.

[51] T. Hu, P. Yi, Z. Guo, J. Lan, and J. Zhang, "Bidirectional matching strategy for multi-controller deployment in distributed software defined networking," *IEEE Access*, vol. 6, pp. 14946–14953, 2018.

[52] T. Wang, F. Liu, J. Guo, and H. Xu, "Dynamic SDN controller assignment in data center networks: Stable matching with transfers," in *Proc. 35th Annu. IEEE Int. Conf. Comput. Commun.*, Apr. 2016, pp. 1–9.

[53] T. Wang, F. Liu, and H. Xu, "An efficient online algorithm for dynamic SDN controller assignment in data center networks," *IEEE/ACM Trans. Netw.*, vol. 25, no. 5, pp. 2788–2801, Oct. 2017.

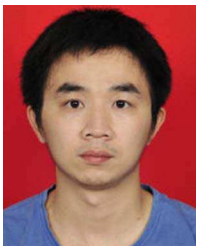
[54] H. Li, P. Li, S. Guo, and A. Nayak, "Byzantine-resilient secure software-defined networks with multiple controllers in cloud," *IEEE Trans. Cloud Comput.*, vol. 2, no. 4, pp. 436–447, Oct./Dec. 2014.

[55] T. Hu, Z. Guo, J. Zhang, and J. Lan, "Adaptive slave controller assignment for fault-tolerant control plane in software-defined networking," in *Proc. IEEE Int. Conf. Commun.*, May 2018, pp. 1–6.

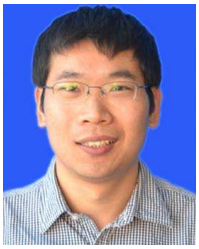
[56] T. Hu, P. Yi, Z. Guo, J. Lan, and Y. Hu, "Dynamic slave controller assignment for enhancing control plane robustness in software-defined networks," *Future Gener. Comput. Syst.*, vol. 95, pp. 681–693, Jun. 2019.

[57] E. Sakic, N. Đerić, and W. Kellerer, "MORPH: An adaptive framework for efficient and Byzantine fault-tolerant SDN control plane," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 10, pp. 2158–2174, Oct. 2018.

Zehua Guo received the B.S. degree from Northwestern Polytechnical University, the M.S. degree from Xidian University, and the Ph.D. degree from Northwestern Polytechnical University. He was a Research Fellow of the Department of Electrical and Computer Engineering, New York University Tandon School of Engineering, New York City, NY, USA, and a Post-Doctoral Research Associate at the Department of Computer Science and Engineering, University of Minnesota Twin Cities, Minneapolis, MN, USA. His research interests include software-defined networking, network function virtualization, data center networks, cloud computing, content delivery networks, network security, green networks, machine learning, and Internet exchange. He was the Session Chair of the IEEE International Conference on Communications 2018. He serves as an Associate Editor for the IEEE ACCESS and the *EURASIP Journal on Wireless Communications and Networking* (Springer), an Editor of the *KSII Transactions on Internet and Information Systems*, and the Technical Program Committee of the *Computer Communications* (Elsevier).

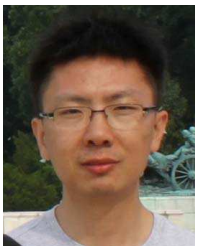


Weikun Chen received the B.Sc. degree in information and computing science from Sun Yat-sen University, China, in 2014. He is currently pursuing the Ph.D. degree with the Institute of Computational Mathematics and Scientific/Engineering Computing, Chinese Academy of Sciences, Beijing, China. His main research interests include the mixed integer programming, in particular the cutting planes.

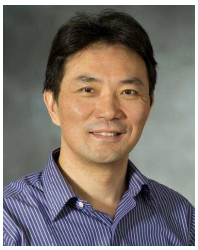


Ya-Feng Liu (M'12–SM'18) received the B.Sc. degree in applied mathematics from Xidian University, Xi'an, China, in 2007, and the Ph.D. degree in computational mathematics from the Chinese Academy of Sciences (CAS), Beijing, China, in 2012. During his Ph.D. study, he was supported by the Academy of Mathematics and Systems Science (AMSS), CAS, to visit Prof. Z.-Q. (Tom) Luo at the University of Minnesota (Twins Cities), Minneapolis, MN, USA, from 2011 to 2012. In 2012, he joined the Institute of Computational Mathematics and Scientific/Engineering Computing, AMSS, CAS, where he is currently an Associate Professor. His main research interests are nonlinear optimization and its applications to signal processing, wireless communications, and machine learning. He is especially interested in designing efficient algorithms for solving optimization problems arising from the above-mentioned applications.

Dr. Liu received the Best Paper Award from the IEEE International Conference on Communications (ICC) in 2011, the Best Student Paper Award from the International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt) in 2015, the Chen Jingrun Star Award from the AMSS in 2018, and the Science and Technology Award for Young Scholars from the Operations Research Society of China in 2018. He has been serving as a Guest Editor for the *Journal of Global Optimization* since 2016.



Yang Xu (S'05–M'07) received the B.E. degree from the Beijing University of Posts and Telecommunications in 2001 and the M.Sc. and Ph.D. degrees in computer science and technology from Tsinghua University, China, in 2003 and 2007, respectively. From 2007 to 2019, he was with the Department of Electrical and Computer Engineering, New York University Tandon School of Engineering, New York City, NY, USA, where he has been a Research Associate Professor since 2013. Since 2019, he has been a Full Professor with the School of Computer Science, Fudan University, China. He has published more than 60 journal and conference papers. He holds more than ten U.S. and international granted patents on various aspects of networking and computing. His research interests include software-defined networks, data center networks, network function virtualization, and network security. He served as a TPC Member for many international conferences, and as an Editor for the *Journal of Network and Computer Applications* (Elsevier), the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS–Special Series on Network Softwarization and Enablers, and *Wiley Security and Communication Networks Journal*–Special Issue on Network Security and Management in SDN.



Zhi-Li Zhang (F'12) received the B.S. degree in computer science from Nanjing University, China, in 1986, and the M.S. and Ph.D. degrees in computer science from the University of Massachusetts, Amherst, MA, USA, in 1992 and 1997, respectively. In 1997, he joined the Computer Science and Engineering Faculty, University of Minnesota, Minneapolis, MN, USA, where he is currently a Qwest Chair Professor and a Distinguished McKnight University Professor. His research interests lie broadly in computer communication and networks, Internet technology, content distribution systems, and cloud computing and emerging Internet of Things (IoT) applications. He is a member of ACM. He was a co-recipient of several best paper awards. He has received a number of other awards. He has co-chaired several conferences/workshops (as a General or TCP Chair) including IEEE INFOCOM, IEEE ICNP, ACM CONEXT, and IFIP Networking. He has served on the TPC for numerous conferences/workshops.