# Towards a Flexible Design of SDN Dynamic Control Plane: An Online Optimization Approach

Mu He, Amir Varasteh, and Wolfgang Kellerer

***Abstract*—With a centralized control over the forwarding devices and the embedded flows, Software Defined Networking promises to increase the flexibility of communication networks. Meanwhile, a dynamic control plane would adapt itself in a timely manner to sustain flow setup performance in the face of traffic variations. Such adaptation depends on a careful decision of the controller placement, which is challenging because we need to consider two contradictory objectives, namely the cost of operating the control plane and the cost of its adaptation. In this work, we model the problem of operating the control plane as a multi-period offline optimization problem to minimize the total cost induced by the flow setup performance and the control plane adaptation. We leverage the lookahead control scheme and decompose the intractable offline problem into smaller instances, which are solved in an online fashion efficiently with an algorithm based on simulated annealing. We perform extensive simulations on real world topologies and show that our proposed algorithm can reduce the total cost by up to 20% compared with the reference algorithms. Further, we analyze the need of frequent control plane adaptation, and compare different control plane design choices according to a novel flexibility measure.**

***Index Terms*—Software Defined Networking, Controller Placement Problem, Control Plane Reconfiguration, Online Cost Optimization.**

## I. Introduction

Managing the Internet's core has become one of the first priorities of network operators. Besides the number of end-users that keeps increasing, new types of network services keep emerging because of new technologies such as IoT and tactile network, which have different requirements in terms of latency, jitter, bandwidth, etc. Such trend stimulates network operators to deploy and manage the network resources with novel networking concepts, e.g., Software Defined Networking (SDN) and Network Function Virtualization (NFV).

Software Defined Wide Area Network (SD-WAN), the application of SDN in wide area networks, is proposed to simplify network configuration and management tasks. With a global view of the underlying data plane, the centralized control plane can make forwarding decisions and manage link and node resources in a more efficient manner. A WAN typically covers very large geographical area, and the underlying data traffic requires both high processing capacity [1] and reliability [2]. Therefore, SD-WAN deploys several physically distributed controllers, with each controller managing a subset of forwarding devices [3]. The controllers synchronize periodically to maintain the global view of the network. This network design is more flexible than the legacy networks running only MPLS or OSPF [4], [5].

Recently, flexibility has become a buzzword in the research of communication networks [6]. It refers to the ability of a network to accommodate new requests that can be represented as different traffic requirements and distributions [6], [7]. SD-WAN endows more degrees of freedom in managing the forwarding devices and is more flexible for the following reasons. *(i)* In the face of temporal burden of the control plane, more controllers can be instantiated to alleviate the potential increase of controllers' response time [1]. After the burden, some controllers can be released to avoid unnecessary operational cost. A controller can be migrated from one location (e.g., Data Center (DC)) to another location [8] for the sake of better performance indicators such as shorter switch-controller and inter-controller latency. *(ii)* Switches with high traffic load can be reassigned to different controllers with enough capacity to ensure control plane load balancing and decrease controller's response time [1], [9]. *(iii)* In terms of traffic control, different traffic engineering algorithms can be implemented in the controllers [5], [10]. When the traffic distribution changes, the network operator can adapt the applied algorithm, without manually updating the forwarding devices.

The advantages of SD-WAN, its flexibility in particular, do not come at no cost: the flow setup time overhead [11] and the controller synchronization [12] are two major factors that worth consideration. Working in the reactive mode, a switch needs to contact the controller for each new flow before it knows where to forward the packets. The distributed controllers also need to frequently synchronize to maintain the same logical view of the network, which can create a huge overhead in the control plane, especially when the number of switches increases or the controller replicas scale out [12].

Meanwhile, it is challenging to operate the control plane of SD-WAN to adapt to the dynamic traffic requirement and distribution. Therefore, decisions for migrating a controller or reassigning switches to controllers should be carefully determined. First, the implementation of controller migration and switch reassignment should ensure that the control plane is always active in order to avoid packet loss [1]. Second, we need to efficiently decide the dynamic controller placement with the target of maximizing the static performance while minimizing the overhead, which is NP-hard [13]. Last but not least, the control plane may become unstable and/or suffer from performance degradation during its adaptation [1]. Therefore, a trade-off stands between static performance and dynamic reconfiguration [1] For instance, when traffic distribution changes, control plane adapts itself towards an optimal average flow setup time. The adaptation, however, can take

[1]Unless specified, we use "reconfiguration" and "adaptation" interchangeably in this paper.

a long time, which can lead to increasing switch-controller latency and SLA violations. In this case, it is wiser to reconfigure the network with a new controller placement. This new configuration can be sub-optimal in terms of flow setup performance but does not incur SLA violation during reconfiguration.

We mainly focus on the latter two challenges in this paper. The controller placement problem has been initiated by Heller et al. [14]. The state-of-the-art mainly optimizes controller placement towards minimum control latency [14]–[18], minimum control load imbalance [9], [15], and maximum reliability [19]. However, the end-to-end flow setup time has not drawn enough attention. Thus, a proper modeling that considers both the latency between the switch and the controller and the latency of controller processing is still missing in the literature.

For the trade-off analysis during reconfiguration, the reconfiguration cost should be thoughtfully modeled. Wang et al. [18] consider an SDN network in a DC with a control plane that can dynamically scale according to the traffic. However, they model the reconfiguration cost only as the variation in the number of controllers, which is designed for the DC network scenario and may not be directly applicable for the SD-WAN scenario we consider in this paper. The duration of control plane reconfiguration should also be considered: controller migration can happen between two data centers which are far away from each other, and same for the switch reassignment.

In this paper, we first model the end-to-end flow setup time in an SDN-WAN, which includes both propagation delay due to geographical distance and processing delay due to controller's queuing behavior. We then model the control plane reconfiguration cost as the total duration of the controller migration and the switch reassignment. Based on the modeling, we formulate the Dynamic Controller Placement Problem (DCPP) as a multi-period global cost minimization problem. To form the cost objective, we consider both the performance cost in terms the average end-to-end flow setup time, and the reconfiguration cost in terms of the control plane adaptation time. The optimization takes the traffic of all time slots as input, and produces for each time slot an optimal controller placement and switch assignment.

In order to solve the global optimization problem efficiently, we leverage the lookahead control scheme and decompose the original problem into smaller subproblems at different time slots. We propose an algorithm based on Simulated Annealing (SA) to solve the subproblems in an online fashion. The algorithm can also be applied in the situation that only near future traffic is available. Since each subproblem outputs the controller placement results of the next few time slots, we can pre-plan the control plane reconfiguration even before the traffic actually changes and therefore react faster to them.

In summary, we make the following contributions in this paper.

- Modeling of end-to-end flow setup time in an SD-WAN considering both propagation delay due to geographical distance and processing delay because of queuing behavior of controller.
- Mathematical formulation of DCPP as a multi-period optimization problem, which minimizes the total cost of static operation and dynamic reconfiguration.
- Application of the lookahead control scheme and design of a heuristic algorithm to solve the optimization problem efficiently.
- Extensive evaluation and comprehensive analysis of our proposed algorithm with different input parameters. Comparison of the flexibility across different control plane design choices, based on a flexibility measurement framework [6].

The remainder of this paper is structured as follows. Section II presents an overview of the background, as well as the related state-of-the-art research. In Section III, we introduce the dynamic control plane architecture, highlighting the modeling of the end-to-end flow setup time and the adaptation of the control plane. Section IV provides the mathematical formulations for DCPP. In Section V, we propose online algorithms to solve the optimization problem efficiently. The evaluation setup is explained in Section VI. We present the results of our extensive evaluations of the model and proposed algorithms in Section VII. Conclusions and future work are outlined in Section VIII.

## II. Background and Related Work

### A. Background

*1) **Distributed Control Plane Architecture**:* As the most essential building block in the SDN architecture, the control plane is responsible for exchanging the information between network applications and forwarding devices (i.e., data plane) [20]. It listens to the requirements from the application layer above, translates the requirements into flow rules and thereafter, configures the underlying data plane through the control plane. Further, it forwards messages, such as *Packet-In* and *Port-Status*, from the data plane to the application plane so that the applications are aware of current network status and behave in a reactive manner.

Due to the scalability and reliability concerns of a single centralized controller architecture, the control plane can be physically distributed among several instances (yet logically centralized). There are two main categories of distributed control plane architectures. The first category implies that the SDN switches are partitioned horizontally into multiple areas, each area controlled by a single controller instance. Examples of this category are ONOS [21] and OpenDaylight [22]. The other category (e.g, Kandoo [23] and Espresso [24]) employs the hierarchical control structure: the lower layer handles local and frequent events from the data plane, and upper layer handles events that need the global knowledge of the network. In this paper, we focus on the horizontal control plane for the modeling.

*2) **Control Plane Consistency & Controller Migration**:* Maintaining consistency while keeping the performance is one of the challenges that a distributed control plane needs to address. Distributed data store is applied to keep relevant network and application state and guarantees that all SDN applications operate with a consistent network view. If the consistency is violated, abnormal behaviors of controllers can arise. For instance, when the link between two switches in one control domain goes down, and the information is not synchronized in time among all controllers, other controllers can still consider this active link, while calculating flow path which may induce packet drop.

Since controller instances are pieces of software running in Virtual Machines (VMs) hosted in DCs, we can migrate them using two different approaches. As the first approach, the VM is directly migrated from one DC to another with live migration technique. However, the downtime during live migration can not be ignored [25], especially for the VM hosting a controller that needs to process flow setup continuously. To eliminate the downtime, we can apply another approach: instantiate a new controller instance, synchronize the state information, and trigger the handover between the old and new instance [1]. Notably, both approaches can increase the controller's response time, which motivates our modeling of the control plane adaptation cost.

*3) **Cost Optimization**:* Communication networks are dynamic in terms of traffic distribution, service requirements, link and node stability, etc. The management of network resources to adapt to such dynamicity during a time span can be formulated as an offline cost minimization problem. The time span consists of several time slots, and the management decisions are made at each time slot. The cost consists of two parts, namely static operational cost (which is related to the current decision variables) and dynamic switching cost (which is related to both the current and previous decision variables) [18]. Efficient algorithms are needed to solve the cost optimization problem in a timely manner.

*4) **Network Traffic Prediction**:* We need to predict the network traffic, i.e., flows in particular, as input of our optimization problem. The prediction of a single network traffic curve (flow rate) can be regarded as an online learning problem, where temporal correlation is investigated. Researchers have proposed algorithms such as RDA (Regularized Dual Averaging) and FTRL (Follow-The-Regularized-Leader) [26] to address it. Besides temporal correlation, spatial correlation emerges when different traffic curves of different source and destination pairs are predicted simultaneously [27]. Recently, machine learning techniques are widely applied to incorporate both correlation factors, such as Stacked Autoencoder [28] and LSTM [29]. In this work, we assume a precise prediction of network traffic over all time slots.

## *B. Related Work*

In this section, we present the state-of-the-art of the controller placement problem (CPP) and the application of lookahead control in online optimization. We then distinguish our present study from the related work.

*1) **Controller Placement Problem (CPP)**:* We formally define it as the problem of finding the optimal number and locations of controllers, as well as the assignment of switches to the controllers, towards a certain objective [14]. Initial CPP formulations only optimize for static objectives such as average control latency and average inter-controller latency. We define them as *Stacic CPP (SCPP)*. Optimizing for other metrics that depend on dynamic past and/or current input, such as average flow setup time and controller load imbalance, is defined as *Dynamic CPP (DCPP)*. Moreover, if we can incorporate future input, e.g., traffic distribution or link status in the future, we have *DCPP considering the future*. Table I classifies the most representative and recent publications in CPP into different categories and summarize their optimization metrics. A comprehensive survey of conducted research on CPP can be found in [30] and [13].

TABLE I
REPRESENTATIVE PUBLICATIONS IN CPP
(✗: STATIC, ✓: DYNAMIC,✓✓: DYNAMIC CONSIDERING THE FUTURE)

| Ref. | Dyn. | Main Optimization Metrics(s) |
|---|---|---|
| [14] | ✗ | Average control latency |
| [11] | ✗ | Average flow setup time |
| [31] | ✓ | Control plane operational cost |
| [15] | ✗ | Control latency & load imbalance |
| [32] | ✓ | Controller utilization and operational cost |
| [16] | ✗ | Average control latency |
| [17] | ✓ | Max. control latency upon controller failures |
| [18] | ✓✓ | Avg. control latency & controller operational cost |
| [9] | ✓ | Controller load imbalance |
| [33] | ✓ | Average flow setup time |
| [19] | ✗ | Reliability of control path |
| [34] | ✓ | # Controllers under QoS constraints |

The state-of-the-art mainly focuses on SCPP and DCPP, with little consideration of future traffic input. Besides, it is commonly assumed that the controller can migrate across various nodes and the switch can be reassigned from one controller to another without any explicit adaptation cost involved, which is not realistic. The work of Wang et al. [18] has been the first endeavor to address the above two issues but with a simplified model: the adaptation cost only depends on the variation of the number of controllers.

*2) **Lookahead Control**:* To explore the temporal variations of flow distribution and achieve a fair trade-off between static operational cost and dynamic switching cost, lookahead control scheme [35] looks into the future and solves the CPP for the current time slot by optimizing over a window of predicted future flow distributions. As a realization of lookahead control, RHC (Receding Horizon Control) [36] has been applied to solve online cost optimization problems with low competitive ratio, such as DC load balancing [37] and DC energy reduction [38]). However, RHC may change the decision variables frequently with a wrong assumption that the switching (adaptation) cost would get paid off within the prediction window [37]. In other words, RHC does not provide "robust" performance guarantees, which potentially induces large adaptation cost. To address this issue, different variations of FHC (Fixed Horizon Control) have been proposed, such as AFHC [37] and RFHC [18]. From another perspective, with lookahead control, the offline optimization problem (e.g., with 24 time slots if we consider hourly traffic pattern) is split into a series of online problems (each with only a few time slots), which makes the solution process faster.

To the best of our knowledge, this is the first paper that models and investigates DCPP with future traffic input, considering both end-to-end flow setup time and realistic control plane adaptation cost. furthermore, this work leverages the lookahead control scheme to minimize the overall cost efficiently in an online fashion. Evaluation results compare different control plane design choices in terms of flexibility.

## III. Dynamic Control Plane Architecture

In this work, we consider a horizontally distributed SDN control plane shown in Fig. 1. In this architecture, each controller is responsible for a subset of switches that constitute a control domain. According to the example in Fig. 1, controller $C_1$ takes care of $S_{11}$, $S_{12}$ and $S_{13}$ (in blue), and Controller $C_2$ takes care of $S_{21}$, $S_{22}$ and $S_{23}$ (in orange). Both controllers work in the reactive mode, and no forwarding rules are pre-installed in the switches. To ensure the same view of the underlying data plane, controllers synchronize with each other periodically.

### A. End-to-End Flow Setup Time

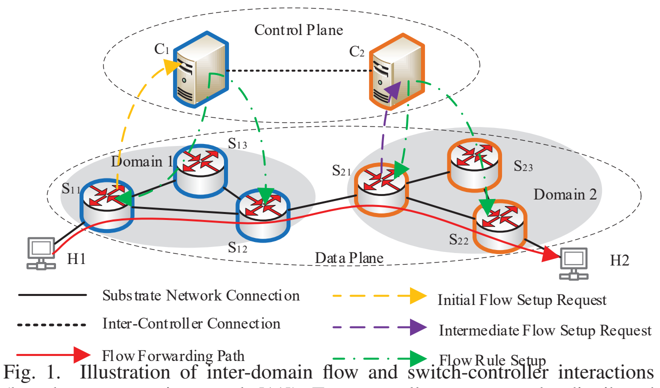
We elaborate the path setup process of an inter-domain flow from $H_1$ to $H_2$ with distributed control plane, which is illustrated in Fig. 1. When the first data packet of the flow reaches switch $S_{11}$, the switch sends an initial flow setup request to its domain controller $C_1$ for processing. $C_1$ calculates the path of this flow with the configured forwarding scheme, e.g., shortest-path or ECMP (Equal-Cost Multi-Path). Thereafter, $C_1$ translates the path into OpenFlow flow rules and sends them to every involved switch inside its own control domain, (in this case $S_{11}$ and $S_{12}$. The data packet is then routed through the first domain until it enters the second domain. Similarly, switch $S_{21}$ initiates an intermediate flow setup request to its controller $C_2$, waits for the flow rule setup from the controller, and forwards it to the next switch in the path. We define the setup time of this flow as the difference between the time $S_{11}$ receives the first data packet from the source host $H_1$ and the time $S_{22}$ successfully starts to forward it to the destination host $H_2$.

Admittedly, there is another way to implement the path setup of a new flow: after setting up the flow rules in its own domain, $C_1$ contacts $C_2$ via the inter-controller channel and asks $C_2$ to set up the rules for the second domain. However, when flow rate increases, this option can create high traffic volume in the control plane and therefore impact inter-controller synchronization performance [12].

### B. Control Plane Adaptation

Fig. 2 illustrates the adaptation of the control plane due to a change of traffic distribution from time $T_1$ to $T_2$ in order to maintain the performance of the flow setup. State-of-the-art research that considers the adaptation of the control domain, such as [18], [31], [33], only considers it in terms of the number of controllers, regardless of where they are. In this paper, we model the control plane adaptation in a more realistic fashion.

Specifically, the adaptation is composed of two parts: controller migration and switch reassignment. We assume that


Control Plane
C1
C2
Domain 1
S13
S11
S12
S21
S23
Domain 2
S22
H1
Data Plane
H2
Substrate Network Connection
Inter-Controller Connection
Flow Forwarding Path
Initial Flow Setup Request
Intermediate Flow Setup Request
Flow Rule Setup
Fig. 1. Illustration of inter-domain flow and switch-controller interactions (based on our previous work [11]). Two controllers compose the distributed control plane, and each one controls three switches. A flow initiates at $H_1$ with the destination of $H_2$, which traverses two control domains.

the control plane is fully distributed, and that each controller has complete knowledge of the underlying network state, such as network topology and flows that are embedded. The state is maintained in each controller's local data store. During controller migration, the data store is transferred from the old controller to the new controller. After the new controller starts running, each switch in its control domain needs to stop the old control channel and establish a new one with the new controller. In Fig. 2, controller $C_2$ is migrated from the location of $S_5$ to the location of $S_4$. Afterwards, switch $S_2$ is reassigned from $C_1$ to $C_2$. The total adaptation time is the sum of controller migration time and switch reassignment time.

## IV. Mathematical Model

This section reveals the mathematical formulation of the cost minimization problem considering the flow setup delay and the control plane reconfiguration cost. We first introduce the variables and notations in our formulation, followed by an elaboration of the operational (flow setup) and the adaptation (reconfiguration) cost factors, which we intend to minimize for a dynamic control plane scenario. Using the formulation of the cost factors, we formally define the offline cost minimization problem with an objective and a set of constraints. Finally, we present some performance metrics for evaluating our proposed algorithms.

### A. Problem Input and Variables

Table II summarizes the notations in our formulation. We consider an SDN network with the topology represented as an undirected and connected graph $\mathcal{G}(\mathcal{V},\mathcal{E})$. Each switch in $\mathcal{V}$ is assigned to one controller that is in one possible location in $C$. All controllers compose the distributed control plane.

There are in total $|T|$ time slots. At each time slot $t \in T$, a distinct $\mathcal{F}^{(t)}$ represents the current set of new flows, where each new flow $f \in \mathcal{V} \times \mathcal{V}$ is defined as a source-destination node pair. Shortest-path algorithm decides the forwarding path for both control and data-plane. The forwarding path and latency between two switches $v$ and $u$ are pre-calculated and denoted as $(v,u)$ and $\ell(v,u)$, respectively.

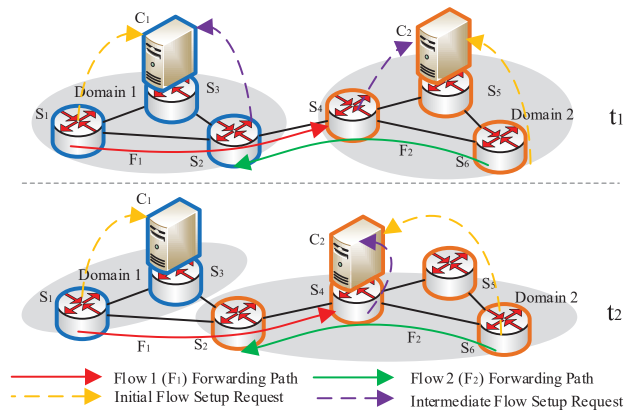

C1
C2
Domain 1
Domain 2
S1
S2
S3
S4
S5
S6
F1
F2
t1
t2
Flow 1 (F1) Forwarding Path
Flow 2 (F2) Forwarding Path
Initial Flow Setup Request
Intermediate Flow Setup Request

Fig. 2. Illustration of control plane adaptation when change of traffic distribution happens. At $t_2$, controller $C_2$ migrates from the location of $S_5$ to the location of $S_4$, and then $S_2$ is reassigned from $C_1$ to $C_2$. For the two flows $F_1$ and $F_2$, the second controller placement at $t_2$ has less incurred control load (3 setup requests) compared with the first one at $t_1$ (4 setup requests).

In order to mathematically model the flow setup time (as in Sec. III-A), we need to consider the processing time (i.e., packet sojourn time) of each controller, which consists of the queuing and the serving delays. A few assumptions are made in this regard. We assume that each controller uses only one thread (i.e., one server) and the flow setup request arrivals follow a Poisson process [18]. Therefore, we can model each controller as an M/M/1 queue. This model has been validated to approximate the processing time of a controller with one switch [39] and multiple switches [40], and it has been applied in previous work of CPP for SD-WAN [16]. Nevertheless, other more accurate queuing models can be incorporated and therefore extend our problem formulation. The expected processing time of a controller $\phi$ is calculated as the follows:

$$\mu_{\Phi}^{(t)}(\phi) = \frac{1}{\theta_\phi - \lambda_{\Phi}^{(t)}(\phi)}. \tag{1}$$

Table III specifies all the variables. All variables have superscript "$(t)$" to represent the values at a particular time slot $t$. At each $t$, we have to make two decisions: *i)* the placement of the controllers with binary variables $p_{\Phi,C}^{(t)}(\phi, c)$, *ii)* the assignment of switches to controllers with binary variables $a_{\mathcal{V},\Phi,C}^{(t)}(v, \phi, c)$.

To build the objective function, we need some help variables such as the number of flow setup requests $\lambda_{\Phi}^{(t)}(\phi)$ at each controller $\phi$ and the flow setup latency $\tau_{\mathcal{V}}^{(t)}(v, u)$ if a flow is forwarded from node $v$ to node $u$. For the illustration of the variable $\lambda_{\Phi}^{(t)}(\phi)$, we refer to the example shown in Fig. 2, Suppose there are two flows which are forwarded through the respective shortest-paths: one $S_1 \longrightarrow S_2 \longrightarrow S_4$, and the other $S_6 \longrightarrow S_4 \longrightarrow S_2$. Regarding the controller placement at $t_1$, the number of incurred flow setup requests of $C_1$ and $C_2$ (including both initial and intermediate setups) are 2 and 2, whereas at $t_2$ the number of requests are 2 and 1. The placement at $t_2$ is obviously preferred in terms of the total incurred control load.

## *B. Cost Structure*

The total cost of operating the dynamic control plane at each time slot $t \in T$ consists of the operational cost $C_F^{(t)}$ and the reconfiguration cost $C_M^{(t)} + C_R^{(t)}$.

TABLE II
SETS AND CONSTANTS

| Notation | Description |
|---|---|
| **Substrate** | |
| $\mathcal{G}(\mathcal{V}, \mathcal{E})$ | Graph of SDN network |
| $\mathcal{V}$ | Set of SDN switches (network nodes), i.e., node locations |
| $\mathcal{E}$ | Set of physical network links |
| $C$ | Set of potential controller locations where $C \subseteq \mathcal{V}$ |
| $\mathcal{P}$ | Set of pre-calculated shortest-paths between all switch pairs |
| $(v, u)$ | Shortest path between two switches $v$ and $u$, with $(v, u) \in \mathcal{P}$ |
| $\ell(v, u)$ | Latency (distance) of the shortest-path $(v, u) \in \mathcal{P}$ |
| $\Omega(v, u)$ | Ordered set of switch pairs along the shortest path $(v, u)$ from $v$ to $u$ |
| **Controllers** | |
| $\Phi$ | Set of controllers' IDs |
| $\theta_\phi$ | Processing capacity of the controller with ID $\phi \in \Phi$ |
| $\kappa_\phi$ | Reserve factor of the controller with ID $\phi \in \Phi$ |
| $\Pi$ | Number of active controllers with $\Pi = \lvert\Phi\rvert$ |
| **Traffic** | |
| $\mathcal{F}^{(t)}$ | Set of flows (flow profile) at time slot $t$ |
| $s^f, d^f$ | source and destination of flow $f \in \mathcal{F}^{(t)}$ |
| $r^f$ | rate to trigger new flow setup of flow $f \in \mathcal{F}^{(t)}$ |
| $T$ | Set of time slots |

*1) **Operational Cost**:* In the reactive mode, SD-WAN introduces additional flow setup time for every new flow. Slow flow setup can potentially lead to SLA violations, as network service can run only after the successful setup of the relevant flows. Therefore, network operators intend to decrease this cost factor, which is formally defined as:

$$C_F^{(t)} = \frac{1}{|\mathcal{F}^{(t)}|} \sum_{f \in \mathcal{F}^{(t)}} \Bigg[ \underbrace{2 \cdot l_{\mathcal{V}}^{(t)}(s^f) + \delta_{\mathcal{V}}^{(t)}(s^f, s^f)}_{\text{Initial flow setup time } C_{F_1}^{(t)}} \\ \underbrace{+2 \cdot \sum_{(v,u) \in \Omega(s^f, d^f)} \tau_{\mathcal{V}}^{(t)}(v, u) + \delta_{\mathcal{V}}^{(t)}(v, u)}_{\text{Intermediate flow setup time } C_{F_2}^{(t)}} \\ + \underbrace{\ell(s^f, d^f)}_{\text{Forwarding time } C_{F_3}^{(t)}} \Bigg]. \tag{2}$$

For each flow setup, we consider twice the control latency of the switch and the controller processing time (denoted as $C_{F_1}^{(t)}$ and $C_{F_2}^{(t)}$). Because the flow forwarding time $C_{F_3}^{(t)}$ is fixed for each flow and does not pose any impact on the decision variables, we can leave it out in our problem formulation.

*2) **Adaptation Cost**:* As explained in Sec.III-B, control plane adaptation in the face of traffic distribution (represented as flow profile) change consists of controller migration and switch reassignment. The controller migration cost is modeled as the total latency induced by migrating controllers, given by:

$$C_M^{(t)} = \sum_{\phi \in \Phi} \sum_{c^{t-1} \in C} \sum_{c^t \in C} \left[ p_{\Phi,C}^{(t-1)}(\phi, c^{t-1}) p_{\Phi,C}^{(t)}(\phi, c^t) \ell(c^{t-1}, c^t) \right] \tag{3}$$

TABLE III
VARIABLES FOR DCPP

| Notation | Description |
|---|---|
| $p^{(t)}_{\Phi,C}(\phi,c)$ | binary variable representing if the controller with ID $\phi \in \Phi$ is placed at node $c \in C$ at time $t$ |
| $a^{(t)}_{\mathcal{V},\Phi,C}(v,\phi,c)$ | binary variable representing if the switch $v \in \mathcal{V}$ is assigned to controller with ID $\phi \in \Phi$ placed at node $c \in C$ at time $t$ |
| $l^{(t)}_{\mathcal{V}}(v)$ | non-negative variable representing the control path latency of a switch $v \in \mathcal{V}$ at time $t$ |
| $d^{(t)}_{\mathcal{V},\Phi}(v,u,\phi)$ | binary variable representing if both switches $v \in \mathcal{V}$ and $u \in \mathcal{V}$ are assigned to the same controller with ID $\phi \in \Phi$ |
| $\overline{d}^{(t)}_{\mathcal{V}}(v,u)$ | binary variable representing if both switches $v \in \mathcal{V}$ and $u \in \mathcal{V}$ are assigned to different controllers |
| $\tau^{(t)}_{\mathcal{V}}(v,u)$ | non-negative variable representing the necessary control forwarding latency if the flow goes from $v \in \mathcal{V}$ to $u \in \mathcal{V}$ |
| $\upsilon^{(t)}_{\mathcal{V}}(v)$ | non-negative variable representing the amount of flow setup requests generated from a switch $v \in \mathcal{V}$ at time $t$ |
| $\upsilon^{(t)}_{\mathcal{V},\Phi,C}(v,\phi,c)$ | non-negative variable representing the amount of flow setup requests generated from a switch $v \in \mathcal{V}$ assigned to controller with ID $\phi \in \Phi$ placed at node $c \in C$ at time $t$ |
| $\lambda^{(t)}_{\Phi}(\phi)$ | non-negative variable representing the total amount of load in terms of the number of Packet-In messages on the controller with ID $\phi \in \Phi$ at time $t$ |
| $\mu^{(t)}_{\Phi}(\phi)$ | non-negative variable representing the expected Packet-In processing time of the controller with ID $\phi \in \Phi$ at time $t$ |
| $\delta^{(t)}_{\mathcal{V}}(v,u)$ | non-negative variable representing the necessary controller processing time if the flow goes from $v \in \mathcal{V}$ to $u \in \mathcal{V}$, and representing the controller processing time when the flow originates at $v$ if $v = u$ |

The forwarding latency $\ell(c^{t-1}, c^t)$ between the previous location $c^{t-1}$ and the current location $c^t$ is the factor that decides the migration time of each controller.

The switch reassignment cost is modeled as the total latency of reassigning switches from one controller instance to another controller instance, given by:

$$C^{(t)}_R = \sum_{v\in\mathcal{V}}\sum_{\phi\in\Phi}\sum_{c^{t-1}\in C}\sum_{c^t\in C}\Big[\Big[\ell(c^{t-1},v)+\ell(c^t,v)\Big]\cdot a^{(t)}_{\mathcal{V},\Phi,C}(v,\phi,c^{t-1})a^{(t)}_{\mathcal{V},\Phi,C}(v,\phi,c^t)\Big] \quad (4)$$

For each switch, the sum of its old and new control latency contributes to its reassignment delay. Note that the switch can be reassigned only after the controllers have been migrated. As an example in Fig. 2, the controller migration cost is $\ell(S_4,S_5)$ and the switch reassignment cost is $\ell(S_2,S_3)+\ell(S_2,S_4)$.

Intuitively, the operational cost $C^{(t)}_F$ only depends on the decision variables at $t$. The adaptation cost $C^{(t)}_M$ and $C^{(t)}_R$, however, is a function of the difference of decision variables between $t-1$ and $t$.

Furthermore, the scaling of controller instances can contribute to the adaptation cost [18]. Scaling controller instances vertically, e.g., increase the number of CPU cores of an instance, requires reboot of the VM [41], which leads to temporary unreadability of that instance and incurs high cost. Horizontal scaling, on the other hand, can take place beforehand, i.e. new controller instances are instantiated before they are actually being used. The cost in this case mainly involves the deployment of new VMs and the higher inter-controller synchronization. We plan to incorporate such cost in our model as future work.

### C. The Offline Cost Minimization Problem

With the two cost factors introduced above and the traffic profiles $\mathcal{F}^{(t)}, \forall t \in T$ as input, we formulate the offline problem as **P**, as specified by (5), for cost minimization over $T$,

$$\textbf{P:}\ \text{minimize} \sum_{t\in T}\gamma_F\cdot C^{(t)}_F+\gamma_M\cdot C^{(t)}_M+\gamma_R\cdot C^{(t)}_R, \quad (5)$$

subject to constraints (6) to (17).

$$\sum_{\phi\in\Phi}\sum_{c\in C} p^{(t)}_{\Phi,C}(\phi,c)=\pi^{(t)} \quad (6)$$

$$\sum_{\phi\in\Phi}\sum_{c\in C} a^{(t)}_{\mathcal{V},\Phi,C}(v,\phi,c)=1 \quad \forall v\in\mathcal{V} \quad (7)$$

$$\sum_{v\in\mathcal{V}} a^{(t)}_{\mathcal{V},\Phi,C}(v,\phi,c)\le|\mathcal{V}|\cdot p^{(t)}_{\Phi,C}(\phi,c) \quad \forall\phi\in\Phi,\forall c\in C \quad (8)$$

$$p^{(t)}_{\Phi,C}(\phi,c)=a^{(t)}_{\mathcal{V},\Phi,C}(c,\phi,c) \quad \forall\phi\in\Phi,\forall c\in C \quad (9)$$

$$\sum_{\phi\in\Phi}\sum_{c\in C} a^{(t)}_{\mathcal{V},\Phi,C}(v,\phi,c)\cdot\ell(v,c)=l^{(t)}_{\mathcal{V}}(v) \quad \forall v\in\mathcal{V} \quad (10)$$

$$\sum_{c\in C} a^{(t)}_{\mathcal{V},\Phi,C}(v,\phi,c)\cdot a^{(t)}_{\mathcal{V},\Phi,C}(u,\phi,c)=d^{(t)}_{\mathcal{V},\Phi}(v,u,\phi) \quad \forall v,u\in\mathcal{V},\phi\in\Phi \quad (11)$$

$$1-\sum_{\phi\in\Phi} d^{(t)}_{\mathcal{V},\Phi}(v,u,\phi)=\overline{d}^{(t)}_{\mathcal{V}}(v,u) \quad \forall v,u\in\mathcal{V} \quad (12)$$

$$l^{(t)}_{\mathcal{V}}(u)\cdot\overline{d}^{(t)}_{\mathcal{V}}(v,u)=\tau^{(t)}_{\mathcal{V}}(v,u) \quad \forall f\in\mathcal{F}^{(t)},(v,u)\in\Omega(s^f,d^f) \quad (13)$$

$$\sum_{f\in\mathcal{F}^{(t)}}\Big[\lceil s^f\rceil_v+\sum_{(p,q)\in\Omega(s^f,d^f)}\lceil q\rceil_v\Big]=\upsilon^{(t)}_{\mathcal{V}}(v) \quad \forall v\in\mathcal{V} \quad (14)$$

$$\upsilon^{(t)}_{\mathcal{V}}(v)\cdot a^{(t)}_{\mathcal{V},\Phi,C}(v,\phi,c)=\upsilon^{(t)}_{\mathcal{V},\Phi,C}(v,\phi,c) \quad \forall v\in\mathcal{V},\forall\phi\in\Phi,\forall c\in C \quad (15)$$

$$\sum_{\phi\in\Phi}\sum_{c\in C}\mu^{(t)}_{\Phi}(\phi)\cdot a^{(t)}_{\mathcal{V},\Phi,C}(v,\phi,c)=\delta^{(t)}_{\mathcal{V}}(v,v) \quad \forall v\in\mathcal{V} \quad (16)$$

$$\sum_{\phi\in\Phi}\mu^{(t)}_{\Phi}(\phi)\cdot\sum_{c\in C} a^{(t)}_{\mathcal{V},\Phi,C}(u,\phi,c)\cdot\overline{d}^{(t)}_{\mathcal{V}}(v,u)=\delta^{(t)}_{\mathcal{V}}(v,u) \quad \forall f\in\mathcal{F}^{(t)},(v,u)\in\Omega(s^f,d^f) \quad (17)$$

For a better explanation, we divide the constraints into three groups. The first group (6)-(9) consists of constraints to build the basic controller placement model. To model the flow setup time, the second group (10)-(13) is composed of constraints to formulate the forwarding latency between the SDN switch and the respective controller. The last group (14)-(17) models the controller's processing time, which relates to the number of Packet-In messages it needs to process.

For the first group, Constraint (6) ensures the total number of active controllers to be $\Pi$. We assume that each SDN switch $v$ must be assigned to exactly one controller with ID $\phi$ at node $c$, which is enforced by Constraint (7). Besides, each controller can only be placed on one of the potential controller locations from $C$. Notably, our model can be easily extended to consider SDN switch with multiple controllers. According to Constraint (8), we only assign an SDN switch $v$ to an active controller with ID $\phi$ that has been placed on a certain node $c$. We assume that a switch $v$ should be assigned to a controller with ID $\phi$, if the controller shares the same location $c$ with the switch, which is enforced by Constraint (9). Note that this constraint can be omitted, when a switch is allowed to be assigned to any controller in the control plane.

The second group of constraints mainly targets the control path latency, which is triggered by initial and intermediate flow setup (explained in Section III-A). The control path latency of a switch $v$, i.e., $l_{\mathcal{V}}^{(t)}(v)$, is defined as the shortest path latency between the switch $v$ and the respective controller instance $\phi$ on node $c$, which is described in Constraint (10). For every two switches, Constraint (11) states that they are in the same control domain if they are assigned to the same controller. To derive variable $\overline{d}_{\mathcal{V}}^{(t)}(v,u)$, Constraint (12) checks all controller instances from $\Phi$ with node $v$ and $u$. If $v$ and $u$ are assigned to the same controller instance $\phi$, the variable $\overline{d}_{\mathcal{V}}^{(t)}(v,u)$ is 0; otherwise, 1. Constraint (13) iterates through every consecutive switch pair along a flow forwarding path. If one switch pair, e.g., $v$ and $u$, leads to $\overline{d}_{\mathcal{V}}^{(t)}(v,u) = 1$ (meaning the flow enters a new control domain), node $u$ will initiate a flow setup request with control latency $l_{\mathcal{V}}^{(t)}(u)$, which is denoted as $\tau_{\mathcal{V}}^{(t)}(v,u)$. If $\overline{d}_{\mathcal{V}}^{(t)}(v,u) = 0$, the control latency $\tau_{\mathcal{V}}^{(t)}(v,u)$ will also be 0, because two nodes are in the same control domain.

For the controller's processing time, the third constraints group accumulates the load of each controller in terms of the number of flow setup requests, and then calculates the processing time as the sojourn time accordingly by using queuing theory. Constraint (14) sums up the total amount of new flow setup requests, consisting of the initial and the intermediate ones, of each switch. The function $\lceil x \rceil_v$ returns 1 if $x = v$, otherwise 0. Constraint 15 maps the load of each switch to its controller. Constraint (16) models the expected controller processing time if a new flow originates at one switch, whereas Constraint (17) models the expected controller processing time if a new flow goes from one switch to another switch. For simplicity, we introduce two new variables $\mu_{\Phi}^{(t)}(\phi)$ and $\lambda_{\Phi}^{(t)}(\phi)$ in Constraint (16) and (17), defined as:

$$\mu_{\Phi}^{(t)}(\phi) = \frac{1}{\theta_{\phi} - \lambda_{\Phi}^{(t)}(\phi)} \quad \phi \in \Phi, \text{ and} \tag{18}$$

$$\lambda_{\Phi}^{(t)}(\phi) = \sum_{c \in C} \sum_{v \in \mathcal{V}} \upsilon_{\mathcal{V},\Phi,C}^{(t)}(v,\phi,c) \quad \phi \in \Phi. \tag{19}$$

.

In the objective function Eq. (5), $\gamma_F$, $\gamma_M$ and $\gamma_R$ denotes the weighting factors for operational, controller migration, and switch reassignment costs, respectively.

## D. Performance Metrics

Besides the performance metrics that can be mapped from the above cost factors, we introduce in the following three new performance metrics for a comprehensive analysis of the model and proposed algorithms.

*1) **Total Load**:* It evaluates the sum of the number incurred flow setup requests, including initial setup and intermediate setup of all switches over all time slots, given by:

$$M_N = \sum_{t \in T} \sum_{v \in \mathcal{V}} \upsilon_{\mathcal{V}}^{(t)}(v). \tag{20}$$

$M_N$ is always equal or larger than the total number of new flows $\sum_{t \in T} |\mathcal{F}^{(t)}|$. Smaller $M_N$ indicates less burden on the controllers and potentially better flow setup performance.

*2) **Controller Load Balance Factor**:* Even though the controller load is incorporated in constructing the processing time of each flow setup, the load variance of different controller instances is not explicitly considered. As reported by Zhou et. al. [9], from the security point of view, adversaries can leverage large load variance to identify strategically important controller instances and attack them specifically to increase the level of destruction. On the other hand, large load variance may trigger frequent switch migrations [42], which can degrade the flow setup performance. Therefore, we define the controller load balance factor as

$$M_L^{(t)} = \frac{\sigma\left[\{\lambda_{\Phi}^{(t)}(\phi) | \forall \phi \in \Phi\}\right]}{\mu\left[\{\lambda_{\Phi}^{(t)}(\phi) | \forall \phi \in \Phi\}\right]}, \tag{21}$$

to evaluate the load variance [9]. $\sigma$ and $\mu$ is the standard deviation and the mean of the set of controller load values respectively. This definition is also known as the relative standard deviation in the literature.

*3) **Flexibility Measure**:* Flexibility is often claimed to be the competitive advantage of a network design that can adapt in face of external changes. We have proposed a quantification framework of flexibility in our previous work [6], [7] to enable a whole new perspective for comparing different design choices. In a nutshell, the flexibility of a system is defined as the fraction of new requests that can be supported within a given time threshold from a given sequence of new requests. In our use case, we define the flexibility measure metric as

$$M_{flex} = \frac{|\text{supported } \mathcal{F}^{(t)} \text{ within time threshold}|}{|T|}, \tag{22}$$

where the nominator counts the number of flow profiles that can be satisfied within the performance (i.e., average flow setup time) and migration time constraint, and the denominator is the total number of flow profiles.

## V. Online algorithm Design

In this section, we first propose an approximation technique to solve the offline cost minimization problem with linear optimizer. Because of its intractability, we decompose the offline problem with lookahead control scheme into subproblems at different time slots, which can be solved in an online fashion. To further increase the solution efficiency, we design a simulated annealing based algorithm to solve the online problems.

## A. *Offline Optimization*

In order to be able to solve the offline optimization problem with linear optimizer, such as Gurobi and CPLEX, we need to linearize the non-linear equations in Problem **P**. The high order constraints, i.e., Constraints (11), (13), (15), (16) and (17), can be linearized without loss of optimality [11]). For the definition (18), however, we need to apply piecewise linear approximation and define the following linear relationship between $\mu_{\Phi}^{(t)}(\phi)$ (the expected processing time of a flow setup request at a controller) and $\lambda_{\Phi}^{(t)}(\phi)$ (the total number of flow setup requests the controller needs to process),

$$\mu_{\Phi}^{(t)}(\phi) = \begin{cases} \frac{1}{(\theta_\phi - \lambda_1)\theta_\phi}\lambda + \frac{1}{\theta_\phi}, & 0 \leq \lambda < \lambda_1 \\ \frac{1}{(\theta_\phi - \lambda_1)(\theta_\phi - \lambda_2)}\lambda - \frac{\lambda_1 + \lambda_2 - \theta_\phi}{(\theta_\phi - \lambda_1)(\theta_\phi - \lambda_2)}, & \lambda_1 \leq \lambda < \lambda_2. \end{cases} \tag{23}$$

The three segment points 0, $\lambda_1$ and $\lambda_2$ define two levels of controller load low $[0, \lambda_1)$ and high $[\lambda_1, \lambda_2)$. Note that $\lambda_2$ is normally smaller than controller capacity $\theta_\phi$ to model the spare capacity of controller, e.g., $\frac{\lambda_2}{\theta_\phi} \in [0.85, 0.95]$ [18].

We refer to the new optimization problem, which replaces Eq. (18) with Eq. (23), as Problem **P'**. Indeed, the approximation overestimates the processing time of controller and therefore does not guarantee that the global optimum of Problem **P'** equals to the global optimum of Problem **P**. We have Theorem 1 to show the optimality bound and the proof can be found in Appendix A.

*Theorem 1:* Suppose the controller capacity $\theta_\phi$ is 1000, and the segment points reside at 0, 700 and 900. Solving the linearized problem **P'** optimally achieves 1.79-approximation of the original problem **P**.

## B. *Online Algorithm Design*

*1) **Lookahead Control Scheme**:* We leverage lookahead control to address our cost optimization problem. With only a limited knowledge of future input [35], i.e., input within a lookahead window with size $\omega$, lookahead control decomposes an offline problem into a series of online subproblems, each with a small number of time slots (equals to $\omega$). We introduce two types of lookahead control, RHC [36] and FHC [18], in PseudoCode 1. RHC solves the optimization problem at each time slot $t$ over the lookahead window $(t, t+\omega)$. Each optimization takes place given the system state of the last time stamp $t-1$. Thereafter, only the decision variables of the first time slot $t$ are applied and the remaining decision variables, i.e., of $t+1, ..., t+\omega$, are discarded. FHC, on the other hand, keeps all the decision variables of time slot $t, .., t+\omega$ and directly jumps over the current lookahead window $\omega$. Fig. 3 illustrates the differences between the two alternatives. FHC guarantees shorter running time by solving fewer optimization problems ($\lceil |T|/(\omega+1) \rceil$) compared to RHC ($|T|$).

Lookahead control can be used as an algorithm wrapper that encapsulates any inner algorithm, including branch-and-cut-based optimization and efficient heuristics. The inner algorithm recursively solves the sub-problems within the present lookahead window in an online fashion.

**PseudoCode 1** Online Algorithm Wrapper

```
 1: procedure RHC($\omega$, $\mathcal{F}^{(t)}$)
 2:     for $t \in T$ do
 3:         Solve the problem P' over $(t, t+\omega)$ with SA-based algorithm
 4:         Update the control plane state
 5:     end for
 6: end procedure

 7: procedure FHC($\omega$, $\mathcal{F}^{(t)}$)
 8:     $T' \leftarrow \{t | t \bmod (\omega+1) = 1, t \in T\}$
 9:     while $t < |T'|$ do
10:         Solve the problem P' over $(t, t+\omega)$ with SA-based algorithm
11:         Update the control plane state
12:     end while
13: end procedure
```

*2)* **SA:** Now we present an efficient algorithm based on SA to solve the online sub-problems. SA accepts a new solution that is worse than the current solution with a varying probability. The probability depends on the difference between the objective function value of the current and the new solution, as well as the current temperature.

We detail our SA-based algorithm in PseudoCode 2. There are four parameters that determine the total number of random searches (i.e., iterations, one iteration from Line 6 to Line 23): initial temperature $T_i$, stop temperature $T_s$, temperature update ratio $\alpha$, and number of searches per temperature $R$. They need to be tuned beforehand in order to take the most advantage of the algorithm, which we will explain in Section VII-B. The symbol $R_c, R_n$ and $R_b$ represents the current, the neighbor and the best result found respectively, each with objective function value $obj_c, obj_n$ and $obj_b$. Symbol $\delta$ denotes the difference of objective function values between the current and the neighbor result. Symbol $cnt_b$ denotes the number of iterations with which the best result does not change.

In each iteration, the solution space of the controller placement problem at one time slot (defined as a *problem slice*) is explored with one of the three following procedures with equal probabilities (Line 6). *(i)* Procedure ReassignSwitch (RS) randomly selects one switch from a control domain with size larger than 1 and reassigns it to another control domain. *(ii)* Procedure SwapSwitch (SS) randomly selects two switches from two control domains and swaps their controllers. *(iii)* Procedure RelocateController (RC) randomly selects a controller and change its location to that of another switch of its control domain. Procedure Evaluate calculates the objective function value. If the lookahead window size $\omega$ is non-zero, the algorithm randomly selects a *problem slide* at one time slot $t \in \{0, 1, ..., \omega - 1\}$. An early-stop mechanism (Line 25) can expedite the algorithm by returning the best solution $R_b$, if it is not improved for $CNT$ steps. Otherwise, the algorithm continues until the current temperature $temp$ is lower than $T_s$.

*3)* **Complexity Analysis:** Considering the worst-case (i.e., without early-stop), the SA-based algorithm needs for RHC $|T| \lceil R \cdot \log_\alpha \frac{T_s}{T_i} \rceil$ or for FHC $\lceil T/(\omega+1) \rceil \cdot \lceil R \cdot \log_\alpha \frac{T_s}{T_i} \rceil$ iterations, which are independent from the number of switches and controllers. In each iteration, procedure RS can go through all controllers and switches to find a switch that can be reassigned, resulting in $O(|\Phi| + |\mathcal{V}|)$ complexity. Procedure

**PseudoCode 2** SA-Based Algorithm

```
1: procedure SA($\omega$)
2:     Initialize $R_c, R_b, obj_b, cnt_b \leftarrow 0, temp \leftarrow T_i$
3:     while $temp \geq T_s$ do
4:         $r_i \leftarrow 0$                                          ▷ Run id
5:         for $r_i < R$ do
6:             Randomly call RS, SS, or RC and get neighbor result $R_n$
7:             $obj_c, obj_n \leftarrow$ EVALUATE($R_c, R_n$)
8:             if $obj_b \geq \min(obj_c, obj_n)$ then
9:                 $cnt_b \leftarrow 0$                        ▷ Best solution updated
10:                Copy the better one of $R_c$ and $R_n$ to $R_b$
11:            else
12:                $cnt_b \leftarrow cnt_b + 1$                ▷ Best solution kept
13:            end if
14:            $\delta \leftarrow (obj_c - obj_n)/obj_n$
15:            if $\delta > 0$ then
16:                Copy $R_n$ to $R_c$                         ▷ Neighbor is better
17:            else
18:                $prob \leftarrow \exp(\delta/temp)$         ▷ Probability of acceptance
19:                $r \leftarrow U(0,1)$                       ▷ Random number
20:                if $r < prob$ then
21:                    Copy $R_n$ to $R_c$                     ▷ Accept worse neighbor
22:                end if
23:            end if
24:            if $cnt_b > CNT$ then
25:                Return $R_b$                                ▷ Early stop
26:            end if
27:        end for
28:        $temp \leftarrow temp \cdot \alpha$                 ▷ Update temperature
29:    end while
30:    Return $R_b$                                            ▷ Normal stop
31: end procedure
```

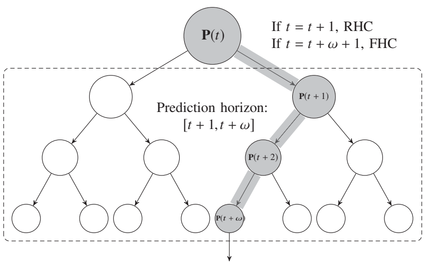


Fig. 3. The trajectory explored by a limited lookahead window size $\omega$. The shaded area represents the solution of an online algorithm, based on the predicted future problem input, i.e., traffic profiles. (Adapted from [35])

SS can search over all possible pairs of controllers, which indicate a complexity of $O(|\Phi|^2)$. For Procedure RC, all controllers may need to be examined, resulting in a complexity of $O(|\Phi|)$. The remaining steps of an iteration contribute to $O(1)$. Therefore, the overall worst-case complexity is $O(|\Phi| + |\mathcal{V}|) + O(|\Phi|^2) + O(|\Phi|) + O(1) = O(|\Phi|^2 + |\mathcal{V}|)$.

## VI. Evaluation Set-Up

For the evaluation, we extend the Python-based framework of our previous work [11] and use Gurobi 8.0 as the optimizer. In this section, we introduce our evaluation setup and parameters and, as well as the procedure to generate realistic traffic.

### A. Evaluation Input & Output

We evaluate three network topologies from the Topology Zoo [43]: Abilene (11 nodes), AttMpls (25 nodes) and OS3E (34 nodes). The number of controllers $\Pi$ varies between 2 and 5 and its impact on the evaluation metrics is analyzed. For the traffic generation, we randomly choose a subset of all possible source-destination node pairs and create flows between the selected pairs between them. We create subsets with different sizes to represent different flow densities. The details to generate realistic traffic are introduced in the next subsection. The number of time slots $|T|$ for random and realistic traffic is considered as 30 and 24, respectively. In our traffic model, there is a positive correlation between the size of the topology and the number of generated new flows: more flows indicate more flow setup requests to the controller. To avoid controller overload, we set the controller capacity for the three topologies as 1000, 5000 and 10000, respectively. The segment points for linear approximation are 0, 0.7 and 0.9 times the respective controller capacity.

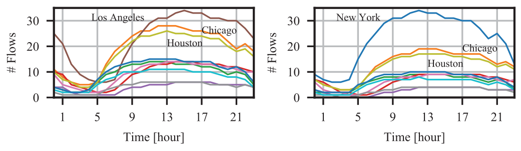
We compare the following algorithms. (1) `STA`: We fix the controller locations and keep the switch assignment, which is the optimal solution of minimizing the average control latency for the first time slot. This is an algorithm we have adopted from our previous work [11] which does not consider control plane reconfiguration. (2) `CNPA`: We take another state-of-the-art algorithm adapted from [16] which is designed based on topology clustering. We need to amend the original algorithm, so that the clustering can take the dynamicity of the flows into account. (3) `OPT(`$\omega$`)`: Due to huge computational complexity, it is not realistic to optimize for the whole time horizon, i.e., the full time slot set $T$. We therefore only optimize the sub-problems with various lookahead window sizes (i.e., different $\omega$). (4) `RHC(`$\omega$`)`: Similarly, we solve the sub-problems with our proposed online algorithm based on RHC and SA with different lookahead window sizes. (5) `FHC(`$\omega$`)`: We solve the cost minimization problem with FHC as the online algorithm wrapper.

For the output, each algorithm under evaluation should return the decision variables for the controllers' location ($p^{(t)}_{\Phi,C}(\phi, c)$) and the assignment of switches to controllers ($a^{(t)}_{\mathcal{V},\Phi,C}(v, \phi, c)$). The remaining variables defined in Table III only helps to formulate the optimization problem $\mathbf{P}$, and therefore they will not be calculated by the algorithms (except `OPT(`$\omega$`)` which returns the optimal solution).

### B. Realistic Traffic Generation

Our previous work [44] models the traffic demand of each gateway in the core gateway network and considers the population of the city that is closest to the gateway and time. The traffic demand from each gateway is equally divided and forwarded to all the other gateways. In the scenario of SD-WAN network, more factors need to be taken into consideration. On the one hand, traffic demand volume between source and destination node is potentially related to both nodes. On the other hand, we would expect shifting of the traffic demand curve between the same source node to different destination nodes, and between different source nodes to the same destination nodes.

In order to support our assumption, we analyze the traffic matrices data of the Abilene network that is publicly available in [45]. The Abilene network consists of 11 nodes (distributed in four different time zones) and 14 links. The matrices cover 6 months of traffic with a 5-minute step. We conduct statistical



(a) New York (b) Los Angeles

Fig. 4. Daily traffic patterns of different source nodes: one on the east coast and the other on the west coast. For each source node, the curves corresponding to different target nodes shift. On the other hand, the peak points among all curves of the two source nodes appear at different time.

tests and evaluate the relation of source and destination population with the traffic volume. The square root of multiplication of source and destination population outperforms other relations, e.g., source population only, destination population only, maximum of source and destination, etc. We also observe that time shifting only happens for destination nodes in different time zones. For example, the curves of traffic volume between different nodes at east coast always possess the same trend, i.e., reaching maximum at the same time, whereas the maximum point of the curve from one node at east coast to another node at west coast is postponed.

Therefore, we extend the traffic model by incorporating the above observations. The population information of the cities is taken from [46]. Fig. 4 shows the traffic patterns of two source nodes. The y-axis is the traffic volume in terms of the number of flows. Since New York and Los Angeles are the two cities with the largest population, the highest curves represent the traffic between them, i.e., brown curve NY-LA in Fig. 4a and blue curve LA-NY in Fig. 4b. The maximum points of the these two curves, however, correspond to different time slots: NY-LA at 15:00 and LA-NY at 12:00.

## VII. Evaluation Analysis

In this section, we conduct a comprehensive evaluation and analysis of the performance of the introduced algorithms. As our evaluation unfolds, we would like to study: *(i)* the performance of the algorithms, *(ii)* the impact of lookahead window size $\omega$, *(iii)* the necessity of frequent control plane reconfigurations, and *(iv)* the flexibility of different control plane design choices.

### A. Which parameter combination is preferred?

As introduced in Sec. V-B, it is critical to decide the four algorithmic parameters, i.e., $T_i$, $T_s$, $\alpha$ and $R$, that will be applied in our SA-based algorithm. In this regard, we perform a study to get a proper parameter combination. We fix $T_s$ as 0.001, while variate $T_i$ from the set $\{0.1, 0.4, 0.5, 0.75\}$, $R$ from the set $\{100, 200, 500\}$, and $\alpha$ from the set $\{0.9, 0.95\}$.

Regarding the algorithms that apply stochastic search paradigm, the longer it runs, the higher chance the solution converges to the global optimum. Indeed, we can observe a trade-off between quality of solution (in terms of average flow setup time, controller migration cost, and switch reassignment cost) and runtime. Due to space limit, we only report part of our comparison in Fig. 5 from Abilene, where $\alpha = 0.95$, $\Pi = 2$, and $\omega = 1$. The average flow setup time (as in Fig. 5a) keeps decreasing when $R$ increases from 100 to 500. For migration and reassignment costs, cost reduction is obviously improved when $R$ increases to 200. However, further improvements become marginal when it goes up to 500 and the runtime inflates several times. For the initial temperature, $T_i = 0.5$ has slightly better performance than $T_i = 0.75$. The observations also apply to other parameter combinations. Therefore, we use $T_i = 0.5$, $T_s = 0.001$, $R = 500$ and $\alpha = 0.95$ for the following evaluations.

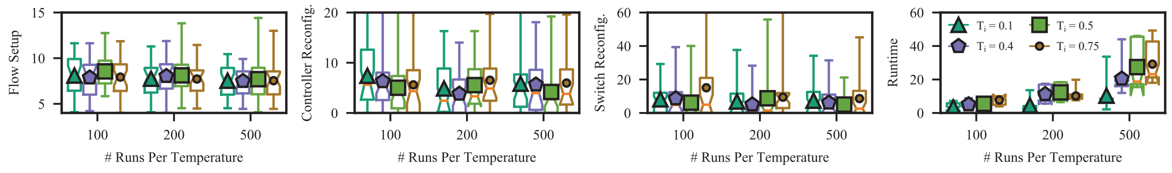
### B. How good is our algorithm?

**Cost Reduction.** Fig. 6 plots the three cost factors and the overall cost objective. We compare `STA`, `CNPA`, `OPT(0/1/2)` and `FHC(1/2)`, and set the reconfiguration cost with low priority ($\gamma_F = 1$, $\gamma_M = \gamma_R = 0.1$). The evaluation is performed on Abilene topology with flow density as 0.05 and number of controllers $\Pi$ equals to 2 and 4. We make the following observations. *(i)* When the number of controllers increases, the average flow setup time decreases for all the algorithms, which is consistent with our previous work [11], where average flow setup time is considered as the only optimization objective. *(ii)* `STA` has the worst flow setup performance on average, but enjoys zero controller and switch reconfigurations due to its static nature. *(iii)* Compared to `OPT(0/1/2)` [2] and `FHC(1/2)`, `CNPA` can achieve similar performance of controller reconfiguration, but at the sacrifice of worse performance of the other two cost factors and the total cost. *(iv)* `FHC(1/2)` can achieve similar flow setup performance compared with `OPT(1/2)`. However, when it comes to the reconfiguration, `FHC(1/2)` does not always promise smaller reconfiguration latency. We enlarge the priority of reconfiguration ($\gamma_F = 1$, $\gamma_M = \gamma_R = 0.5$) and re-evaluate the algorithms in Fig. 7. The former observations *(i)* and *(ii)* still apply. As a proper response to the higher priority, `OPT(0/1/2)` pushes the two reconfiguration costs down, which is not revealed for `FHC(1/2)`. For the overall cost, the performance of `FHC(1/2)` is similar to that of `OPT(0/1/2)` when the reconfiguration priority is lower. Nevertheless, `FHC(1/2)` can be worse than `STA` at high reconfiguration priority, because `STA` has zero reconfiguration cost.

Fig. 8 shows the detailed overall cost values for Abilene topology ($\Pi = 4$) for all time slots in $T$. We can observe that for low flow densities, the curves of `OPT(0/1)` and `FHC(0/1)` intertwine and there is no absolute winner. `CNPA`, however, always delivers the worst overall cost values, on average 2x that of the other algorithms, and in the worst case, 4x. When the flow density becomes larger, the trend of `CNPA` does not change. It can be seen that The difference between `OPT(0)` and `FHC(0/1)` is marginal, and they three are slightly better than `OPT(1)`.

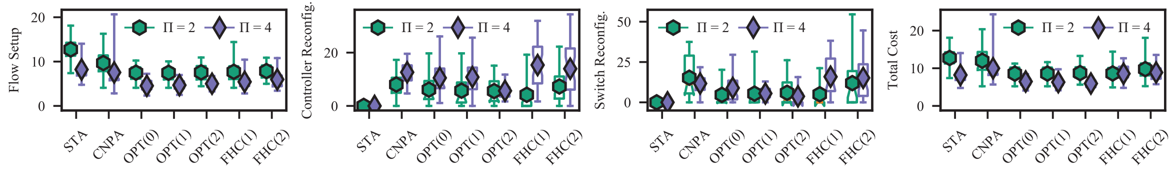
**Controller Load.** We create Fig. 9 and plot the total load and the load balance factor (defined in Sec IV-D) for two different topologies. The result of Abilene is depicted in Fig. 9a and Fig. 9b. The total incurred load in terms of the

[2]`OPT(0/1/2)` represents all the three cases of `OPT(0)`, `OPT(1)` and `OPT(2)`. Same for the notation of other algorithms.



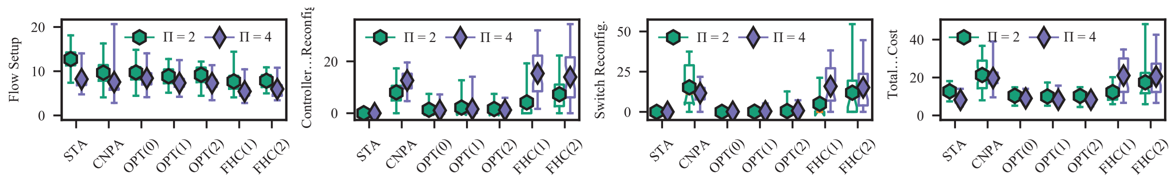
(a) Average flow setup time [ms] (b) Controller reconfiguration [ms] (c) Switch reconfiguration [ms] (d) Runtime [s]

Fig. 5. Three cost factors, i.e, $C_F$, $C_M$ and $C_R$, and algorithm runtime (over 30 runs) obtained of topology Abilene with different parameters $T_i$ and $R$. The other two parameters are fixed: $T_s = 1$ and $\alpha = 0.95$. Number of controllers Π is 2, and the lookahead window size $\omega$ is 1.



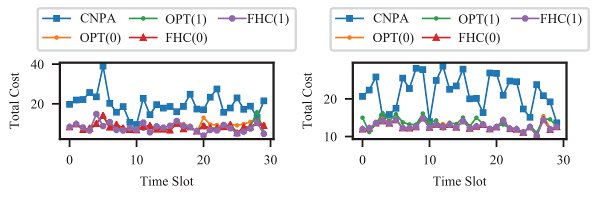
(a) Average flow setup time [ms] (b) Controller reconfiguration [ms] (c) Switch reconfiguration [ms] (d) Total cost

Fig. 6. Three cost factors, i.e, $C_F$, $C_M$ and $C_R$, and the compound cost with coefficient $\gamma_F$, $\gamma_M$, $\gamma_R$ equals to 1, 0.1 and 0.1 respectively (over 30 runs) obtained of topology Abilene with different algorithms. Flow density equals 0.05.



(a) Average flow setup time [ms] (b) Controller reconfiguration [ms] (c) Switch reconfiguration [ms] (d) Total cost

Fig. 7. Three cost factors, i.e, $C_F$, $C_M$ and $C_R$, and the compound cost with coefficient $\gamma_F$, $\gamma_M$, $\gamma_R$ equals to 1, 0.5 and 0.5 respectively (over 30 runs) obtained of topology Abilene with different algorithms. Flow density equals 0.05.



(a) Low flow density (b) High flow density

Fig. 8. Total cost with coefficient $\gamma_F$, $\gamma_M$, $\gamma_R$ equals to 1, 0.5 and 0.5 respectively obtained of topology Abilene with different algorithms. $\pi^{(t)} = 4$.

number of flow setup requests increases when the number of controllers increases. `FHC(0/1/2)` achieve smaller total load compared to `STA` and `CNPA`, and the load balance factor is nearly the same for all algorithms. Fig. 9c and Fig. 9d show the evaluation for AttMpls, with which we have the same observation for load balance factor. Additionally, it can be observed that the improvement of total load of `FHC(0/1/2)` becomes marginal, since load balancing is not an explicit optimization objective in our model.

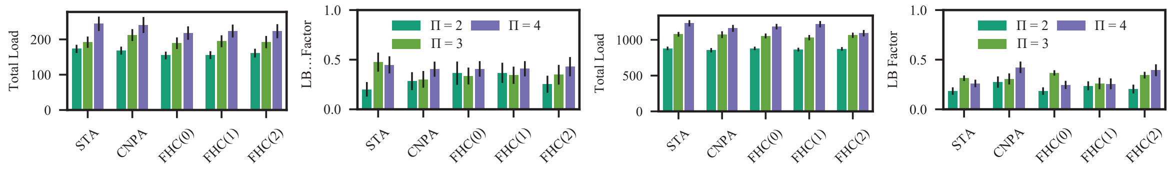
**RHC vs. FHC.** Because of its robustness with future dynamic input, `RHC` is theoretically superior to `FHC`. However, when it comes to particular optimization problems, the advantage of `RHC` can be marginal, which is the case for our problem. Fig. 10 compares the induced cost of `RHC` and `FHC` by showing the difference between the respective cost factors. The performance of the two lookahead control schemes are nearly the same: boxplots distribute symmetrically with means at zero. We conclude that both can make "robust" decisions here. When we consider the computation time, `FHC` obviously defeats `RHC`, since `FHC` does not need to solve the cost optimization for each $t$.

**Takeaway.** Our proposed algorithms `RHC` and `FHC` show their advantage of a fair trade-off between flow setup and control plane reconfiguration costs, compared with the state-of-the-art algorithms `STA` and `CNPA`. We would prefer `FHC` in this work because of its short runtime.

### C. *How long we should look into the future?*

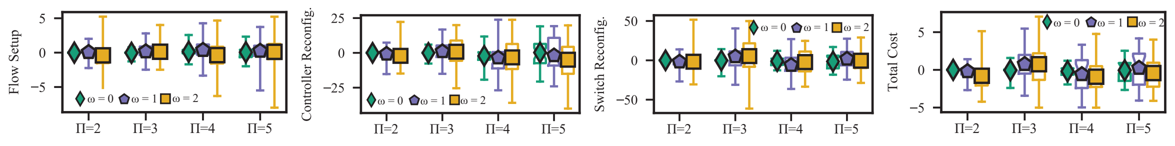
This is an interesting question for all types of lookahead control schemes. Intuitively, larger lookahead window size leads to lower overall cost, because it can achieve a better trade-off between operational cost and reconfiguration cost by considering more time slots. However, this intuition is not reflected in our evaluation results. Fig. 11 compares the total cost of different lookahead window sizes for different topologies and number of controllers (Π from 2 to 5). We observe that the total cost increases slightly with a larger window size, a phenomenon we have already seen in Fig. 6d and 7d. This is due to the positive correlation between the complexity of the online problem and the window size. When the complexity increases, our proposed algorithm converges slower and is likely to output a less optimal solution. Nevertheless, when we look into the three cost coefficients, we notice that operational cost is the only one that actually decreases (Fig. 6a and 7a).

**Takeaway.** With the same parameter sets for SA, larger lookahead window size $\omega$ leads to worse objective function, while the runtime is reduced. In this case, we suggest to employ different algorithm parameter combinations (e.g., higher



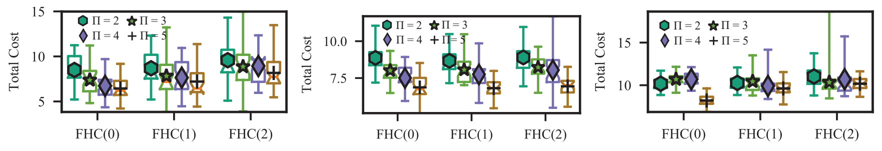
(a) Total Load (Abilene) (b) Load Balance Factor (Abilene) (c) Total Load (AttMpls) (d) Load Balance Factor (AttMpls)

Fig. 9. Total incurred load (in terms of the number of flow setup requests) and load balance factor with $\gamma_F$, $\gamma_M$, $\gamma_R$ equals to 1, 0.1 and 0.1 respectively (over 30 runs) obtained of two different topologies with different algorithms. Flow density equals 0.05.



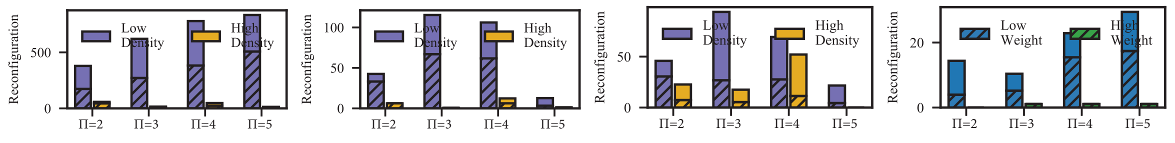
(a) Average flow setup time (b) Controller reconfiguration (c) Switch reconfigurtion (d) Total cost

Fig. 10. Performance comparison of RHC and FHC with different lookahead window size $\omega$: 0, 1 and 2. Cost coefficient $\gamma_F$, $\gamma_M$, $\gamma_R$ equals to 1, 0.1 and 0.1 respectively. Results obtained from Abilene topology over 30 runs with flow density of 0.05.



(a) Abilene (b) AttMpls (c) OS3E

Fig. 11. Total cost with coefficient $\gamma_F$, $\gamma_M$, $\gamma_R$ equals to 1, 0.1 and 0.1 comparing different lookahead window sizes $\omega$ from 0 to 2, different number of controllers Π, and different topologies. Results obtained using `FHC` algorithm over 30 runs with flow density equals 0.05.



(a) Abilene (b) AttMpls (c) OS3E (d) Abilene (Realistic Traffic)

Fig. 12. Reconfiguration cost of different traffic distribution with cost coefficient $\gamma_F$, $\gamma_M$, $\gamma_R$ equals to 1, 0.1 and 0.1 respectively (over 30 runs) obtained from different topologies for (a) to (c). In (d), we compare $\gamma_F = 1$, $\gamma_M = 0.1$, $\gamma_R = 0.1$ (low weight) and $\gamma_F = 1$, $\gamma_M = 0.5$, $\gamma_R = 0.5$ (high weight).

initial temperature $T_i$ and lower stop temperature $T_s$) to extend the stochastic search procedure.

## D. Do we need frequent reconfigurations?

The frequency of control plane reconfiguration impacts the induced cost, and it is not always required to have many reconfigurations. Fig. 12a to 12c plot the reconfiguration costs for the three topologies with random traffic. We compare different numbers of controllers and types of traffic. The lower (shaded) and upper part of each bar represents the reconfiguration cost of controller and switch, respectively. In general, we can observe that when the traffic distribution is more evenly distributed (i.e., high flow density), the intention of the reconfiguration becomes less obvious. The control plane can stay unchanged for most flow profiles with good flow setup performance. For Abilene and low flow density, more controllers lead to higher reconfiguration cost. Nevertheless, the trend becomes different for the other two topologies, where the cost reaches the maximum at $\Pi = 3$ and drops afterwards.

For the case of random traffic, there is no spatial or temporal correlation among the flows of different source and destination pairs. Therefore, it represents the worst case in terms of controller and switch reconfiguration cost. In order to reveal the realistic situation, we apply the traffic introduced in Sec. VI-B on Abilene topology and plot the reconfiguration cost in Fig. 12d. Compared with the reconfiguration cost of 500 for random traffic, realistic traffic only triggers a cost of 20 on average. The two bars for each Π represent low and high weights for reconfiguration cost, respectively. The results confirm our previous conclusion: with high weighting factors, our proposed algorithm can push down the value of the particular cost coefficient.

**Takeaway.** By using prediction techniques, we can analyze the correlation within the traffic. If the correlation is strong, we can stick to a single static controller placement throughout all time slots without losing much performance optimality. If the correlation is low, we would expect frequent reconfigurations to maintain the flow setup performance. Meanwhile, tuning the weight parameters ($\gamma_F$, $\gamma_M$ and $\gamma_R$) can help to suppress unnecessary reconfigurations.

## E. How flexible is the control plane?

We evaluate the flexibility (introduced in Sec. IV-D3) as follows. First, we define a performance threshold (average flow setup time) and a migration threshold (total migration time). We then iterate through all flow profiles, and for each flow profile, we check whether the old placement can satisfy the flow profile in terms of the performance threshold. If yes,

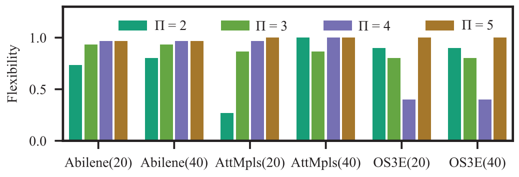


Fig. 13. Flexibility measure comparison of different migration time constraints and different topologies. Solutions obtained with `FHC(1)`.

the old placement stays without triggering reconfiguration, which is counted as a successful adaptation. Otherwise, a control plane reconfiguration is inevitable, and the induced reconfiguration time is evaluated. We count it as a successful adaptation only if the reconfiguration time is smaller than the migration threshold. Otherwise, we keep the old placement and count it as a failed adaptation.

Fig. 13 compares the flexibility for different topologies, different numbers of controllers and migration time thresholds (numbers in brackets). We have the following observations. *(i)* When migration time threshold increases, the flexibility value also increases for each scenario (same topology and number of controllers). *(ii)* In most of the cases, the more controllers we have, the more flexible the control plane is. *(iii)* For OS3E, the flexibility value of $k = 4$ is lower than all the other $k$s, due to the randomness of the evaluated flow profiles. As future work, we plan to extend the flexibility analysis and consider more types of traffic distribution.

**Takeaway.** We need a standard approach to quantify the flexibility of networks and compare different system design choices [6]. With numerical results, we show that the number of controllers can affect the flexibility of the control plane.

## VIII. Conclusion

Traffic variability exists broadly in modern WAN [2], [3] and data center networks [47], [48], which can affect the performance and reliability of transmitting high bandwidth among thousands of end-hosts and VMs. In this regard, SDN leverages the idea of a centralized control plane by allocating resources to varying traffic flows in a more efficient manner. To maintain an acceptable flow setup time, control plane should adapt itself with controller migration and switch reassignment. Control plane adaptation, however, is costly because of induced temporary instability, additional signaling, and performance degradation. This paper focuses on the mathematical formulation of the problem of managing a dynamic control plane (i.e., DCPP) to achieve a fair trade-off between flow setup and adaptation cost.

We model the end-to-end flow setup time in the SD-WAN scenario and formulate DCPP as a multi-period optimization problem, considering both operational and adaptation costs. Because of its intractability, we leverage the scheme of lookahead control and propose efficient online algorithms. Our comprehensive evaluation shows that the proposed algorithms can achieve up to 30% average flow setup time reduction and 20% reconfiguration cost reduction, compared with the state-of-the-art algorithm. We also observe that there is a trade-off between the operational and the adaptation cost. By comparing different design choices, we conclude that more controllers can increase the flexibility of the dynamic control plane.

## Acknowledgment

This work is part of a project that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No 647158 - FlexNets). The authors alone are responsible for the content of the paper.

## References

[1] A. Dixit *et al.*, "ElastiCon: an elastic distributed SDN controller," in *Proceedings of the ACM Symposium on Architectures for Networking and Communications Systems (ANCS)*. IEEE, 2014, pp. 17–27.
[2] S. Jain *et al.*, "B4: Experience with a globally-deployed software defined WAN," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 3–14, 2013.
[3] R. Ahmed and R. Boutaba, "Design considerations for managing wide area software defined networks," *IEEE Communications Magazine*, vol. 52, no. 7, pp. 116–123, 2014.
[4] M. Casado *et al.*, "Fabric: a retrospective on evolving SDN," in *Proceedings of the ACM Workshop on Hot Topics in SDN*. ACM, 2012, pp. 85–90.
[5] Y. Guo *et al.*, "Traffic engineering in SDN/OSPF hybrid network," in *Proceedings of the IEEE International Conference on Network Protocols (ICNP)*. IEEE, 2014, pp. 563–568.
[6] W. Kellerer *et al.*, "How to measure network flexibility? A proposal for evaluating softwarized networks," *IEEE Communications Magazine*, 2018.
[7] M. He *et al.*, "Flexibility in softwarized networks: Classifications and research challenges," *IEEE Comm. Surveys & Tutorials*, 2019.
[8] ——, "How flexible is dynamic SDN control plane?" in *Proceedings of the IEEE INFOCOM Workshops*. IEEE, 2017, pp. 689–694.
[9] Y. Zhou *et al.*, "Elastic switch migration for control plane load balancing in SDN," *IEEE Access*, vol. 6, pp. 3909–3919, 2018.
[10] S. Agarwal *et al.*, "Traffic engineering in software defined networks," in *Proceedings of the IEEE INFOCOM*. IEEE, 2013, pp. 2211–2219.
[11] M. He *et al.*, "Modeling flow setup time for controller placement in SDN: Evaluation for dynamic flows," in *Proceedings of the IEEE International Conference on Communications (ICC)*. IEEE, 2017, pp. 1–7.
[12] A. S. Muqaddas *et al.*, "Inter-controller traffic to support consistency in ONOS clusters," *IEEE Transactions on Network and Service Management*, vol. 14, no. 4, pp. 1018–1031, 2017.
[13] A. K. Singh and S. Srivastava, "A survey and classification of controller placement problem in SDN," *Wiley International Journal of Network Management*, vol. 28, no. 3, 2018.
[14] B. Heller *et al.*, "The controller placement problem," in *Proceedings of the ACM Workshop on Hot Topics in SDN*. ACM, 2012, pp. 7–12.
[15] S. Lange *et al.*, "Heuristic approaches to the controller placement problem in large scale SDN networks," *IEEE Transactions on Network and Service Management*, vol. 12, no. 1, pp. 4–17, 2015.
[16] G. Wang *et al.*, "An effective approach to controller placement in software defined wide area networks," *IEEE Transactions on Network and Service Management*, vol. 15, no. 1, pp. 344–355, 2018.
[17] B. P. R. Killi and S. V. Rao, "Capacitated next controller placement in software defined networks," *IEEE Transactions on Network and Service Management*, vol. 14, no. 3, pp. 514–527, 2017.
[18] T. Wang *et al.*, "An efficient online algorithm for dynamic SDN controller assignment in data center networks," *IEEE/ACM Transactions on Networking*, vol. 25, no. 5, pp. 2788–2801, 2017.
[19] Y. Wang *et al.*, "Resource allocation for reliable communication between controllers and switches in SDN," *Springer Journal of Network and Systems Management*, pp. 1–27, 2018.
[20] F. Bannour *et al.*, "Distributed SDN control: Survey, taxonomy, and challenges," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 1, pp. 333–354, 2017.
[21] P. Berde *et al.*, "ONOS: towards an open, distributed SDN OS," in *Proceedings of the third workshop on Hot topics in Software Defined Networking*. ACM, 2014, pp. 1–6.

[22] J. Medved *et al.*, “Opendaylight: Towards a model-driven SDN controller architecture,” in *Proceedings of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*. IEEE, 2014, pp. 1–6.

[23] S. Hassas Yeganeh and Y. Ganjali, “Kandoo: a framework for efficient and scalable offloading of control applications,” in *Proceedings of the ACM Workshop on Hot Topics in SDN*. ACM, 2012, pp. 19–24.

[24] K.-K. Yap *et al.*, “Taking the edge off with espresso: Scale, reliability and programmability for global internet peering,” in *Proceedings of ACM SIGCOMM*. ACM, 2017, pp. 432–445.

[25] H. Liu *et al.*, “Performance and energy modeling for live migration of virtual machines,” in *Proceedings of the International Symposium on High Performance Distributed Computing*. ACM, 2011, pp. 171–182.

[26] H. B. McMahan *et al.*, “Ad click prediction: a view from the trenches,” in *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining*. ACM, 2013, pp. 1222–1230.

[27] A. Nucci *et al.*, “The problem of synthetically generating IP traffic matrices: initial recommendations,” *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 3, pp. 19–32, 2005.

[28] Y. Lv *et al.*, “Traffic flow prediction with big data: A deep learning approach,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 2, pp. 865–873, 2015.

[29] R. Vinayakumar *et al.*, “Applying deep learning approaches for network traffic prediction,” in *Proceedings of the International Conference on Advances in Computing, Communications and Informatics*. IEEE, 2017, pp. 2353–2358.

[30] G. Wang *et al.*, “The controller placement problem in software defined networking: a survey,” *IEEE Network*, vol. 31, no. 5, pp. 21–27, 2017.

[31] M. F. Bari *et al.*, “Dynamic controller provisioning in software defined networks,” in *Proceedings of the International Conference on Network and Service Management (CNSM)*. IEEE, 2013, pp. 18–25.

[32] M. T. I. ul Huque *et al.*, “Large-scale dynamic controller placement,” *IEEE Transactions on Network and Service Management*, vol. 14, no. 1, pp. 63–76, 2017.

[33] G. Cheng *et al.*, “Toward a scalable SDN control mechanism via switch migration,” *IEEE China Communications*, vol. 14, no. 1, pp. 111–123, 2017.

[34] M. Tanha *et al.*, “Capacity-aware and delay-guaranteed resilient controller placement for Software-Defined WANs,” *IEEE Transactions on Network and Service Management*, 2018.

[35] D. Kusic *et al.*, “Power and performance management of virtualized computing environments via lookahead control,” *Springer Cluster computing*, vol. 12, no. 1, pp. 1–15, 2009.

[36] W. Kwon and A. Pearson, “A modified quadratic cost problem and feedback stabilization of a linear system,” *IEEE Transactions on Automatic Control*, vol. 22, no. 5, pp. 838–842, 1977.

[37] M. Lin *et al.*, “Online algorithms for geographical load balancing,” in *Proceedings of the International Green Computing Conference*. IEEE, 2012, pp. 1–10.

[38] ——, “Dynamic right-sizing for power-proportional data centers,” *IEEE/ACM Transactions on Networking*, vol. 21, no. 5, pp. 1378–1391, 2013.

[39] M. Jarschel *et al.*, “Modeling and performance evaluation of an OpenFlow architecture,” in *Proceedings of the 23rd International Teletraffic Congress (ITC)*. IEEE, 2011, pp. 1–7.

[40] K. Mahmood *et al.*, “Modelling of OpenFlow-based software-defined networks: the multiple node case,” *IET Networks*, vol. 4, no. 5, pp. 278–284, 2015.

[41] M. Ghaznavi *et al.*, “Elastic virtual network function placement,” in *Proceedings of the IEEE International Conference on Cloud Networking (CloudNet)*. IEEE, 2015, pp. 255–260.

[42] C. Liang *et al.*, “Scalable and crash-tolerant load balancing based on switch migration for multiple openflow controllers,” in *Proceedings of the International Symposium on Computing and Networking*. IEEE, 2014, pp. 171–177.

[43] S. Knight *et al.*, “The Internet topology zoo,” *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, 2011.

[44] A. Basta *et al.*, “SDN and NFV dynamic operation of LTE EPC gateways for time-varying traffic patterns,” in *Proceedings of the International Conference on Mobile Networks and Management*. Springer, 2014, pp. 63–76.

[45] S. Orlowski *et al.*, “SNDlib 1.0: Survivable network design library,” *Wiely Networks: An International Journal*, vol. 55, no. 3, pp. 276–286, 2010.

[46] National population totals and components of change: 2010-2017. [Online]. Available: https://www.census.gov/data/tables/2017/demo/popest/nation-total.html

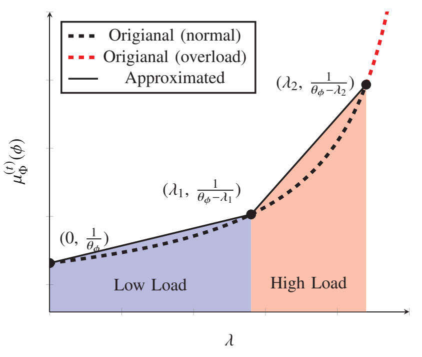


Fig. 14. Piecewise linear approximation of the average controller sojourn time.

[47] F. Liu *et al.*, “eba: Efficient bandwidth guarantee under traffic variability in datacenters,” *IEEE/ACM Transactions on Networking*, vol. 25, no. 1, pp. 506–519, 2017.

[48] J. Guo *et al.*, “Pricing intra-datacenter networks with over-committed bandwidth guarantee,” in *Proceedings of the USENIX Annual Technical Conference*, 2017, pp. 69–81.

## Appendix A
## Proof of Theorem 1

*Proof:* The three anchor points are $(0, \frac{1}{\theta_\phi})$, $(\lambda_1, \frac{1}{\theta_\phi - \lambda_1})$ and $(\lambda_2, \frac{1}{\theta_\phi - \lambda_2})$. The two line pieces are expressed as follows.

$$f_1(\lambda) = \frac{1}{(\theta_\phi - \lambda_1)\theta_\phi}\lambda + \frac{1}{\theta_\phi}, \quad 0 \le \lambda < \lambda_1 \tag{24}$$

$$f_2(\lambda) = \frac{1}{(\theta_\phi - \lambda_1)(\theta_\phi - \lambda_2)}\lambda - \frac{\lambda_1 + \lambda_2 - \theta_\phi}{(\theta_\phi - \lambda_1)(\theta_\phi - \lambda_2)}, \quad \lambda_1 \le \lambda < \lambda_2 \tag{25}$$

In order to get the largest gap between the approximation and the real curves, we substrate $\frac{1}{\theta_\phi - \lambda}$ from (24) and (25), and calculate the first order derivative, e.g.,

$$\begin{aligned}(\Delta f_1(\lambda))' &= \left[\frac{1}{(\theta_\phi - \lambda_1)\theta_\phi}\lambda + \frac{1}{\theta_\phi} - \frac{1}{\theta_\phi - \lambda}\right]' \\ &= \frac{1}{\theta_\phi(\theta_\phi - \lambda_1)} - \frac{1}{(\theta_\phi - \lambda)^2}\end{aligned} \tag{26}$$

Let the derivative equal zero, we then have

$$\begin{aligned}\theta_\phi(\theta_\phi - \lambda_1) &= (\theta_\phi - \lambda_l)^2 \\ \lambda_l &= \theta_\phi \pm \sqrt{\theta_\phi^2 - \theta_\phi\lambda_1}\end{aligned} \tag{27}$$

We take the minus and do the same thing to (25)

$$\lambda_r = \theta_\phi \pm \sqrt{\theta_\phi^2 - \theta_\phi(\lambda_1 + \lambda_2) + \lambda_1\lambda_2} \tag{28}$$

$\max(\lambda_l, \lambda_r)$ returns the largest gap. In the worst-case, all flows need to issue flow setup requests in each control domain, and therefore we have $\max(\lambda_l, \lambda_r) \cdot \Pi$ for the approximation curve on top of the real one. ■