A Scalable Approach to SDN Control Plane Management: High Utilization Comes With Low Latency

Victoria Huang, Gang Chen, Peng Zhang, Hao Li, Chengchen Hu, Tian Pan, and Qiang Fu

***Abstract*—One major research challenge for Software-Defined Networking is to properly deploy and efficiently utilize multiple controllers to improve resource utilization and maintain high network performance. While addressing this Controller Placement Problem (CPP), many existing studies overlooked the importance and influence of the Controller Scheduling Problem (CSP) with the central focus on proper distribution of requests from all switches among all controllers. In this paper, we define a new Controller Placement and Scheduling Problem (CPSP), emphasizing on the necessity and importance of tackling both CPP and CSP simultaneously in a coherent framework. To solve CPSP, we must seek a combination of solutions to both problems. Particularly, CSP is addressed based on a given solution to CPP and a Gradient-Descent-based (GD-based) scheduling algorithm is developed to optimize the probabilistic distribution of requests among all controllers. Built on the GD-based approach for controller scheduling, a Clustering-based Genetic Algorithm with Cooperative Clusters (CGA-CC) is further proposed to address CPP. In comparison to the majority of heuristic methods developed in the past, CGA-CC has two unique strengths. Specifically, it partitions a large network to substantially reduce the search space of the Genetic Algorithm (GA), resulting in fast identification of high-quality CPP solutions. Moreover, a greedy load re-distribution mechanism is developed to handle unexpected demand variations by dynamically forwarding bursting requests to neighboring sub-networks. Extensive simulations showed that our algorithms can significantly outperform several existing algorithms, including a recently proposed approach called Multi-controller Selection and Placement Algorithm (MSPA), in terms of both response time and controller utilization.**

***Index Terms*—Software-defined networking, distributed controller architectures, controller placement, controller scheduling.**

Manuscript received April 17, 2019; revised August 12, 2019 and December 7, 2019; accepted January 31, 2020. Date of publication February 11, 2020; date of current version June 10, 2020. The work of Peng Zhang was supported by the National Natural Science Foundation of China (NSFC) under Grant 61772412. The work of Hao Li was supported by NSFC under Grant 61702407. The work of Tian Pan was supported in part by the National Key Research and Development Program of China under Grant 2019YFB1802600, and in part by NSFC Grant 61702049. The associate editor coordinating the review of this article and approving it for publication was P. Chemouil. *(Corresponding author: Victoria Huang.)*

Victoria Huang, Gang Chen, and Qiang Fu are with the School of Engineering and Computer Science, Victoria University of Wellington, Wellington 6011, New Zealand (e-mail: guiying.huang@ecs.vuw.ac.nz; aaron.chen@ecs.vuw.ac.nz; qiang.fu@ecs.vuw.ac.nz).

Peng Zhang, Hao Li, and Chengchen Hu are with the Department of Computer Science and Technology, Xi'an Jiaotong University, Xi'an 710049, China (e-mail: p-zhang@xjtu.edu.cn; hao.li@xjtu.edu.cn; chengchenhu@xjtu.edu.cn).

Tian Pan is with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China (e-mail: pan@bupt.edu.cn).

Digital Object Identifier 10.1109/TNSM.2020.2973222

I. Introduction

AS AN emerging networking paradigm, Software-Defined Networking (SDN) is notable for extracting the network control from the distributed data plane to form a logically centralized control plane [1], [2]. With its support of centralized network management and rapid deployment of new network policies, SDN has been widely applied to many real-world networks (e.g., Google B4 [3]). To enhance network scalability and reliability, the control plane is commonly implemented based on a distributed controller architecture (e.g., ONOS [4] and Onix [5]) where multiple controllers can be deployed. In response to the industry requirement, in the recent literature, many researchers have studied the problem of deploying a large number of controllers, up to 2500 as reported in [6].

With the wide adoption of distributed controller architectures, the *Controller Placement Problem (CPP)* becomes a critical research issue. Although over-provisioning can be considered as a possible strategy to solve CPP, solely applying over-provisioning is not economical and effective for the day-to-day operation of many real networks due to the network traffic dynamics. As defined by Heller *et al.* [7], CPP has the goal to identify both the number and locations of controllers in any given network so as to achieve some important objectives, such as minimizing propagation latency [7], [8], [9] and improving resource utilization [10]. For instance, a suitable number of controllers must be placed in proximity to demanding switches in accordance with the dynamic fluctuations of traffic workload. This is critical to the overall network reliability and performance, especially for wide-area networks [6], [8], [11].

However, even with appropriate placement of controllers, we may still run into the risk of poor network performance if the requests cannot be properly distributed among controllers. In this paper, the *Controller Scheduling Problem (CSP)* is defined as the problem of optimizing the distribution of requests from all switches among all controllers so as to achieve certain objectives, such as load balancing and minimizing request response time. Without solving

CSP properly, controllers can easily experience high workload and, as a result, the response time of a controller can increase significantly to 15 times of its normal value under light workload, as reported in [12] and our simulation study in Section VI-A. Such a controller will become sensitive to its workload changes. A slight increase in request arrival rate can easily cause the controller to be overloaded, resulting in high variation in response time.

Motivated by the above understanding, Yao *et al.* [13] introduced the capacitated K-center problem where the controller workload is treated explicitly as a controller placement constraint. Several research works following similar ideas can also be found in the literature [6], [8], [10].

Although considerable efforts have been made to address CPP, the importance of CSP has always been underestimated and sometimes ignored [7], [14]. Only a few recent works have explicitly quantified the impact of CSP on the performance of the control plane [8], [15]. Most of these studies assume a fixed switch-controller connection. In other words, the requests from one switch can only be processed by one binded controller, restricting the opportunity to properly distribute workload across all controllers. This restriction among previous approaches may cause poor network performance and degrade resource utilization as reported in [16].

From a practical point of view, CPP and CSP should be simultaneously investigated. In this paper, we present a new problem definition that explicitly captures the strong inter-dependencies between CPP and CSP. To differentiate from the CPP studied before, we name our new problem the *Controller Placement and Scheduling Problem (CPSP)*. We assume that each switch can flexibly dispatch requests to any deployed controllers, providing flexible and effective controller scheduling. However, because we need to consider controller scheduling at the request level instead of the switch level, our CSP becomes more complicated. Apart from that, CPSP highlights the necessity and importance of simultaneously addressing both CPP and CSP within the same framework. Motivated by this understanding, when we tackle the placement problem, the distribution of switch requests among controllers is simultaneously optimized in our work.

To precisely measure the influence of CSP on network performance in a realistic manner, the controller processing latency is explicitly modeled through a mathematical queuing model in this paper. Based on the model, CPSP is subsequently formulated as a constrained optimization problem with the aim to improve the control plane utilization while satisfying the constraint on low network response time. CPSP can be dissected into two strongly coupled sub-problems, i.e., CPP and CSP. A full solution to CPSP is hence obtained as a combination of solutions to both CPP and CSP.

In consideration of the fact that the quality of a solution to CPP depends on the workload distribution among controllers (i.e., the solution to CSP), CSP was investigated first in our work. Based on the assumption that a number of controllers have already been deployed in multiple locations in a network, CSP aims to reduce the average response time of request processing to a reasonably low level. Rather than directly optimizing the distribution of every request across multiple controllers which is unfeasible in large networks, we choose to optimize the probabilistic distributions for any switch to dispatch their requests to all available controllers. In line with this idea, we further develop a Gradient-Descent-based (GD) scheduling algorithm to achieve a good balance between scheduling performance and problem scalability.

Driven by our GD-based scheduling algorithm, a new algorithm called Clustering-based Genetic Algorithm with Cooperative Clusters (CGA-CC) is further proposed to tackle CPP. Since CPP is NP-hard [7], [13], we consider Genetic Algorithm (GA) as a cost-effective method for identifying near-optimal controller placement solutions. When we apply GA to solving CPP, we need to evaluate the network performance (in terms of control plane utilization and the average request response time) with respect to any controller placement solution. For this purpose, we will use the workload distribution identified by our GD-based approach. However, direct application of GA is not ideal for large networks since the corresponding search space is too large for GA to handle effectively. In order to solve CPP scalably, CGA-CC is proposed in this study by utilizing a general purpose clustering algorithm to split the network into non-overlapping sub-networks. Subsequently, GA is applied to tackling CPP in each sub-network. Within a sub-network, each switch is forced to use local controllers most of the time. Thus, long-distance switch-controller communication is prevented. However, there are special cases when long-distance communication is deemed helpful. For example, when all controllers in a sub-network are overwhelmed by unexpected traffic surge, the only hope for the network to continue function efficiently is to seek help from remote controllers. Motivated by this understanding, we develop a greedy algorithm to strategically forward bursting requests in one sub-network to selected controllers in neighboring sub-networks. In this way, we can effectively cope with unexpected demand variations, ensuring good workload balance across all sub-networks and improving the robustness of our approach.

To evaluate the effectiveness of our GD-based scheduling algorithm and CGA-CC, simulation studies have been performed in a range of controller settings and network topologies provided by Sprint [17], a global Tier 1 Internet Service Provider (ISP). The results show that, compared with weighted round-robin and other popular scheduling algorithms, our GD-based scheduling algorithm can effectively reduce the response time by up to 37.5% while keeping the control plane throughput at a high level. Meanwhile CGA-CC significantly outperforms K-center [7] and the recently-proposed Multi-controller Selection and Placement Algorithm (MSPA) [8] in terms of both response time and control plane utilization under various network settings. Moreover, in comparison to GA, CGA-CC clearly performs better in large-scale networks by substantially reducing the computation cost while maintaining low response time and high control plane utilization.

II. Related Work

When CPP was first proposed [7], it was framed as a problem of identifying the number and locations of controllers

without considering CSP, which can easily overload controllers due to sudden increases of network traffic. Aimed at preventing controllers from being overloaded, controller load balancing has always been a popular topic in the research community. For example, a number of works [12], [16], [18] have been proposed lately to improve controller load-balancing whenever uneven load distributions occur.

Recent literature clearly acknowledges the importance of CSP [8], [10], [11], [13], [19]. For example, Yao *et al.* [13] addressed the CPP while taking the controller capacity as a constraint. However, most of the previous approaches for solving CPP relied on simple solutions to CSP. In Yao *et al.*'s work [13], they required that requests from switches never exceed the capacity of their associated controllers without considering the workload distribution among controllers. Therefore, some controllers are more vulnerable to poor utilization due to the underloaded issue. In comparison, we aim to make the best use of controllers by maximizing the controller utilization while maintaining low network response time. Although CPP and CSP have been jointly addressed in [20], the authors only focused on the propagation latency between controllers and switches while the processing latency was completely neglected despite of its practical importance. To address this limitation, the processing latency is explicitly modeled in our work through a queuing model. Follow-up studies [11], [19] subsequently argued that the workload among controllers should always be balanced. This idea is useful when all controllers are evenly deployed in the network with identical capacity. However, in a network where controllers with different capacities are located unevenly, dispatching more requests to nearby low-capacity controllers without overloading them can be an ideal option.

Moreover, most of the existing approaches for CPP are designed for binding-based controller architectures (e.g., ONOS [4] and Onix [5]) that require every switch to always contact its binded controller [16]. One limitation of binding-based architectures is the difficulty for the control plane to quickly adapt to traffic load variations. Although it can be alleviated by the switch migration mechanism [18] to re-associate switches from overloaded controllers to under-utilized ones, the migration itself is complex and time-consuming. Furthermore, its balancing granularity only limits to the switch level.

To tackle the above issues, more sophisticated approaches have been proposed recently. Our previous work [15] investigated CPP together with CSP, aiming to simultaneously optimize the controller utilization and response time with the help of a GA-GD-combined algorithm. However, the minimization of response time is not always necessary in practice (e.g., the benefits of optimizing the response time is negligible when it is sufficiently low). Furthermore, the direct application of GA cannot effectively handle CPP in large networks.

Recently, a Multi-controller Selection and Placement Algorithm (MSPA) [8] was proposed to address CSP together with CPP. Specifically, to minimize the maximum end-to-end latency between controllers and switches, MSPA partitioned the full network into a given number of sub-networks. To make the request response time model accurate and realistic, MSPA considered the queuing latency in controllers which is captured by an M/M/c queuing model. In particular, a centralized scheduler which can be easily implemented as Virtual Network Function (VNF) components is established for distributing requests received from any switch to arbitrary controllers within a sub-network, which can effectively avoid the switch migration issue. In their work, the queuing model assumes that within a sub-network, all requests must go through the scheduler before reaching any controllers. Apparently, the location of the scheduler must be carefully selected, which presents another challenge for CPP. Furthermore, only one single queue is maintained by the scheduler and the scheduler makes dispatching decisions solely based on the controller workload without considering propagation latency, potentially increasing the response time if the request is forwarded to a remote controller. Experimental studies of MSPA will be conducted in Section VI-F.

Despite the limitations, MSPA sheds new light on our research. In this paper, we adopt a recently proposed multi-controller architecture [16] where multiple lightweight schedulers can be easily deployed in the network. Meanwhile each controller manages its own processing queue, greatly simplifying the operation of distributed schedulers. In Section VI-F, empirical comparisons with MSPA will be performed to further demonstrate the potential advantage of our algorithm.

III. Problem Formulation

In this section, we will present a model of the network environment and formulate CPSP.

A. Network Environment

In this paper, we considered a large enterprise global communication backbone network $G = <V, E, D>$ where V is a set of M nodes and E is a set of bidirectional physical links between nodes. The distance function $D : V \times V \rightarrow \mathbb{R}$ defines the one-way propagation latency between any pair of nodes. Each $v \in V$ is a switching node (one switch or a switch group) and is responsible for handling all backbone related communication requests generated by local switches. Requests generated by v follow a Poisson distribution at the rate of λ_v.[1]

In addition to the data plane, controllers in the control plane can be deployed at any node in the network [7], [8], [13]. Depending on where a controller is deployed, the controller can have varied capacities. This is because controllers are not always supported by the same type of server machines at different locations [21]. We hence use α_v to measure the maximum number of requests processable by the controller located at node v within one second. A full solution to CPP can subsequently be represented as a binary vector x where

[1]We assume that requests generated by every local switch follow Poisson distributions. Mathematically, when requests from multiple independent Poisson sources are combined, the aggregated requests to be handled by each node in the backbone network still follow Poisson distribution.

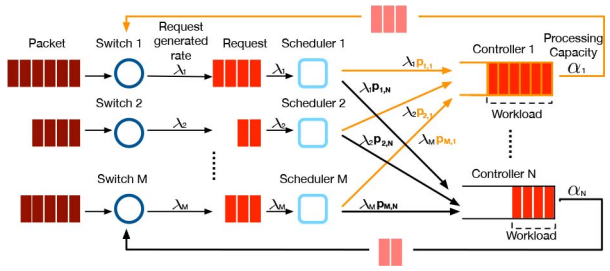


Fig. 1. Request processing procedure.

each element of x is denoted as

$$x_v = \begin{cases} 1, & \text{if } v \text{ is selected} \\ 0, & \text{otherwise} \end{cases}$$

Therefore, the number of deployed controllers is $N = \|x\|_1$. Given x, the set of N deployed controllers can be easily identified and denoted as $\widetilde{V}$.

Unlike [7], [11], [13], we assume that each switch can flexibly dispatch requests to any deployed controllers with the help of a locally installed scheduler as shown in Fig. 1. This assumption can be easily supported as an extension of existing distributed controller architectures (e.g., ONOS and Onix) as demonstrated by the prototype network developed in [16]. Since the network we considered is a large enterprise global communication backbone, frequent changes in network topology are unlikely to happen. Apart from that, since traffic generated by each node is aggregated from local networking devices, the traffic variation is supposed to be smooth. Thus, similar to many existing studies [22], [23], weak/eventual consistency is sufficient in our work.

Note that introducing schedulers into existing SDN architectures will not affect the clear logic distinction between the control and data planes. Specifically, from an architectural and operational point of view, the newly-introduced schedulers are transparent to both controllers and switches because no switch modification is required. Moreover, schedulers will make request dispatching decisions completely independently and in a fully decentralized manner. Therefore, they incur low communication and computation cost and can be easily implemented as light-weight VNF components, hence guaranteeing the scalability of the scheduling plane.

Although we assume that the schedulers are deployed on the data plane, we did not rule out the possibility of placing them in other network locations. In fact, the use of VNF gives ISPs the flexibility about where the schedulers can/should be placed based on their understanding of the network dynamics. Such flexibility is supported by both MSPA and the newly proposed approach in this paper.

After processing a request, a response will be sent from the controller back to the switch. The time interval measured by the switch between sending a request and receiving a response is defined as the request response time.

To measure the performance of any solution x to CPP, we must find the corresponding solution to CSP as well. Given the job of scheduling R requests within a specific time period, the complexity of CSP is $\mathcal{O}(N^R)$ and is intractable in practice. To solve the problem scalably and efficiently, we decide to distribute requests to every controller according to some pre-calculated probabilities. In this way, we can dramatically reduce the problem complexity from $\mathcal{O}(N^R)$ to $\mathcal{O}(MN)$.

Let P be an $M \times N$ matrix which is always guaranteed to exist. $P_{v,v'}$ is the probability for switch v to dispatch any new requests to controller at node v' for processing. As illustrated in Fig. 1, guided by P, we can highly efficiently distribute requests from any switches to any controllers, achieving real-time requirements for fast request processing. To ensure P is well-defined, the sum of each row of P must equal 1:

$$\sum_{v' \in \widetilde{V}} P_{v,v'} = 1, \quad \forall v \in V$$

Note that our CSP provides the flexibility and flow-level granularity of controller scheduling. It is also compatible with existing binding-based network architectures. This is easily realized with additional constraints on P to force switches to send all their requests to binded controllers. We did not explore this scenario for the sake of high scheduling flexibility and fine scheduling granularity.

B. Problem Formulation

As showed in Fig. 1, we can model each controller as an independent M/M/1 queue [24]. According to Little's Law, the long-term average number of customers in a stationary system can be calculated as the product of the arrival rate and the average customer waiting time. On the other hand, the mean queue length in a stationary system equals to the arrival rate divided by the difference between the processing rate and the arrival rate. Then the average waiting time can be calculated as the inverse of the difference between the processing rate and the arrival rate [25]. Similarly, considering each request as a customer, the average processing latency of the controller at node v' can be calculated

$$\tau_{v'} = \frac{1}{\alpha_{v'} - \sum_{v \in V} \lambda_v P_{v,v'}}$$

Note that in an SDN network, events can be divided into topology-altered and topology-unaltered events. To maintain a global synchronized view of the full network topology, only topology-altered events (e.g., switches go up and down) should be synchronized among controllers. As we mentioned in Section III-A, frequent changes in network topology are unlikely to happen [26], [27]. Thus, the number of topology-related events is limited. Nevertheless, to measure the synchronization impact on the network, the worst case scenario is considered here, i.e., each controller directly communicates with all other controllers regularly for synchronization and each time it takes up to N rounds of communications for all controllers to synchronize their local views. Thus, the synchronization cost for the worst case is proportional to N^2. Specifically, we model the synchronization cost F_{syn} for each controller as the product of N^2 and the synchronization factor γ that captures the number of topology-altered events generated within every simulated second of the Sprint network.

$$F_{syn} = \gamma N^2 \tag{1}$$

Note that the synchronization cost can be considered as part of the controller's workload. Hence the controller processing latency can be refined to become:

$$\tau_{v'} = \frac{1}{\alpha_{v'} - \sum_{v \in V} \lambda_v P_{v,v'} - F_{syn}}$$

The response time generally consists of five components: transmission latency, propagation latency, switch processing latency, scheduler processing latency, and controller processing latency. In a high-speed network (e.g., a backbone network considered in this paper), the transmission latency is trivial. Apart from that, with recent advancement of hardware technology, high-performance switches have become prevalent (e.g., NoviSwitch and EZchip), resulting in negligible switch processing latency [28], resulting in negligible switch processing latency. Note that schedulers are solely responsible for request dispatching according to the pre-determined probability distribution, incurring negligible computation overhead (simply toss a coin for every request to be dispatched). Nevertheless, to make our problem formulation accurate and realistic, we use ϵ to jointly capture the transmission latency, switch processing latency and scheduler processing latency.

Thus, the average request response time of the controller at node v' is calculated by averaging the request response time over all requests sent by all switches to the controller. The response time of each request includes the controller processing latency, the round-trip propagation latency, and ϵ:

$$t_{v'} = \tau_{v'} + 2 \cdot \frac{\sum_{v \in V} D\big(v, v'\big) \lambda_v P_{v,v'}}{\sum_{v \in V} \lambda_v P_{v,v'}} + \epsilon \tag{2}$$

The average response time of the network is calculated by averaging the response time of all requests processed by all controllers:

$$t_{avg} = \frac{\sum_{v' \in \widetilde{V}} t_{v'} \big(\sum_{v \in V} \lambda_v P_{v,v'}\big)}{\sum_{v \in V} \lambda_v} \tag{3}$$

Meanwhile, given a solution x to CPP, the control plane utilization is calculated as the proportion of controller capacities used for request processing in the control plane:

$$u = \frac{\sum_{v \in V} \lambda_v}{\sum_{v' \in \widetilde{V}} \alpha_{v'}} \tag{4}$$

In consideration of realistic requirements from network operators [6], [29], we formulate our problem with the main goal of minimizing the network operation cost, which is realized through maximizing the control plane utilization. Accordingly, CPSP aims to find the controller placement x and workload distribution P so as to maximize u without deteriorating the network performance measured by the average response time t_{avg} in (3). This requirement gives rise to the constrained optimization problem below:

$$\max_{x,P} \quad u(x) \tag{5}$$

$$\text{s.t.} \quad t_{avg}(x, P) \leq t_{th} \tag{6}$$

$$\sum_{v \in V} \lambda_v P_{v,v'} \leq \beta\big(\alpha_{v'} - F_{syn}\big), \quad \forall v' \in \widetilde{V} \tag{7}$$

$$\sum_{v' \in \widetilde{V}} P_{v,v'} = 1, \quad \forall v \in V \tag{8}$$

$$0 \leq P_{v,v'} \leq 1, \quad \forall v \in V, \quad \forall v' \in \widetilde{V} \tag{9}$$

Since the improvement of control plane utilization should not come at the expense of network performance, (6) guarantees that response time should remain below a threshold t_{th}. In particular, the latency value for a network depends on several factors, such as the network geographic coverage, access speeds, and the class of service associated with the data [30]. According to [31], Sprint's committed network round trip backbone delay is 55 ms within North America while the delay increases up to 105 ms within Asia. (7) ensures that the workload of each controller never exceeds its capacity. Moreover, a proportion of capacity must be reserved for each controller, as determined by a decay factor β, in order to withstand unexpected traffic bursts. For example, in a network where controller workload varies significantly and quickly, β should be set to a relatively high level. Thus, β enables network operators to impose high-level control over the long-term network operation, as required by most production networks [24]. Finally, both (8) and (9) guarantee that P is well-defined. Note that no assumptions about the type of networks are made in the problem formulation to ensure its generality.

Our formulation of CPSP (5)-(9) produces two subproblems. The first one is to identify the number and locations of controllers (i.e., CPP) while the second one requires us to determine the workload distribution among deployed controllers (i.e., CSP). Accordingly, our solution to CPSP is made up of two inter-dependent parts: one for CPP and the other one for CSP. In consideration of the fact that the quality of a solution to CPP depends on the corresponding solution to CSP, we will first investigate different approaches for solving CSP. A study on various methods for solving CPP will be presented subsequently.

IV. Controller Scheduling Algorithms

In this section, we assume that a solution x to CPP is given in advance. In other words, the CSP is solved based on all deployed controllers. Our task for CSP is to effectively schedule the requests generated by all switches among the deployed controllers to reduce the response time so as to satisfy (6). Accordingly, CSP can be formulated as:

$$\begin{aligned} \underset{P}{\text{solve}} \quad & t_{avg}(P) \leq t_{th} \\ \text{s.t.} \quad & (7),\ (8),\ (9) \end{aligned} \tag{10}$$

In order to achieve low average request response time, a switch will send most of its requests to nearest controllers rather than remote controllers as verified by our simulation studies in Section VI-B. To solve (10), several approaches will be studied, ranging from simple heuristics to a newly proposed approach that periodically calculates the suitable distribution probability P through a gradient-descent-based (GD) technique. We will also analyze the pros and cons of adopting each approach.

A. Weighted Round-Robin Scheduling

Speaking of heuristics, random and round-robin are two widely used scheduling methods. However, they are expected to only perform well when all controllers have identical capacities and are located close to each other, which may not always be true in reality. Thus, Weighted Round-Robin (WRR) scheduling is considered to be more flexible, since it can handle imbalanced controller workload and different controller capacities more effectively.

Similar to many existing research works [32], $P_{v,v'}$ can be made proportional to the controller capacity, as shown below:

$$P_{v,v'} \propto \big(\alpha_{v'} - F_{syn}\big), \quad \forall v \in V \tag{11}$$

Such a Capacity-based WRR (CWRR) can effectively handle the situation when controllers differ in capacities but have similar propagation latencies. However, scheduling requests solely based on capacity information may not be sufficient, especially in a large-scale network where propagation latency contributes significantly to response time. This understanding is further verified by our simulation studies in Section VI-B.

Motivated by this, $P_{v,v'}$ can also be determined as:

$$P_{v,v'} \propto \frac{\alpha_{v'} - F_{syn}}{D(v, v')} \tag{12}$$

This technique is utilized by the Capacity-Delay-based WRR (CDWRR). With CDWRR, the preference of choosing a controller depends not only on its processing capacity but also on its propagation latency with a switch. Thus, a switch has the tendency to send its requests to powerful and close controllers so as to reduce the response time.

It should be noted that $P_{v,v'}$ can be calculated independently by each switch. Since the propagation latency varies for different pairs of controllers and switches, each switch dispatches its requests to the same controller with different probabilities.

B. Gradient-Descent-Based Scheduling

According to (10), t_{avg} can be reduced through adjusting *P*. To further handle the constraints (7)-(9), we adopt the most commonly used penalty function method [33]. Specifically, (10) is transformed into a constraint-free optimization problem by introducing the term in (13) to penalize any constraint violation [33].

$$\begin{aligned} F_{con}(P) = \mu_1 \sum_{v' \in \widetilde{V}} \min\left\{0, \beta\big(\alpha_{v'} - F_{syn}\big) - \sum_{v \in V} \lambda_v P_{v,v'}\right\} \\ + \mu_2 \sum_{v \in V} \left| 1 - \sum_{v' \in \widetilde{V}} P_{v,v'} \right| \\ + \mu_3 \sum_{v \in V} \sum_{v' \in \widetilde{V}} \min\big\{0, P_{v,v'}\big\} \end{aligned} \tag{13}$$

where μ_1, μ_2, and μ_3 are penalty parameters.

Consequently, (10) becomes:

$$\min_P F_{csp}(P) = \min_P \{t_{avg}(P) + F_{con}(P)\}$$

To solve this optimization problem, we develop a GD-based scheduling algorithm. In each iteration, the gradient of $F_{csp}(P)$ with respect to *P* is calculated to guide the search of *P*. Instead of updating *P* using a constant learning rate *l*, *l* is gradually reduced from a high value l_H to a low value l_L based on the following equation for the algorithm to converge reliably:

$$l = l_H - \frac{(l_H - l_L)}{N_I} \cdot i$$

where N_I is the maximum number of iterations and *i* represents the current number of iterations. We can update *P* as:

$$P_{i+1} \leftarrow P_i - \frac{\partial F_{csp}}{\partial P} \cdot l \tag{14}$$

so that the optimization process can gradually shift from constraint satisfaction to $t_{avg}(P)$ minimization.

The whole process repeats until the stopping criterion (i.e., $t_{avg}(P) \leq t_{th}$) is reached. It is important to note that the algorithm can be executed efficiently in practice with the help of advanced learning/optimization tools such as Theano or TensorFlow. During our simulation, we noticed that our algorithm converges quickly (i.e., around one second). With the support of high-performance computers in industry, the convergence time can be further reduced. Moreover, one second of delay is only required when all the controllers currently being used by a switch become unavailable at the same time, which is unlikely to happen. Even such a rare situation happens, the delay will only affect new flows. Any ongoing flows will not experience the delay. Hence, in terms of user experience, end users may not even experience the one second of delay at all, in addition to being acceptably short. Thus, the GD running time can be safely ignored in practice.

Once *P* that solves (10) is found, it will guide request distribution for all switches.

V. Controller Placement Algorithms

Facilitated by a method for solving CSP, we can now address CPP for the purpose of improving controller utilization. As we explained previously, it is unfeasible to find the optimal solution to CPP in a large network. Existing approaches attempted to simplify CPP [7], [13] by assuming that either the number of required controllers is pre-determined or all controllers have identical capacities so that the required controller number can be easily calculated. In this work, we instead study a more realistic scenario where the number of controllers is unknown in advance and controllers with varying capacities can be deployed in the network. To address this CPP, different approaches will be explored.

A. K-Center

K-center is currently one of the most well-known CPP strategies [7] which aims to deploy *k* controllers so as to minimize the worst-case propagation latency from any switch to its closest controller. However, determining a suitable value for *k* is a challenging task. In this work, we introduce an extra

constraint to guarantee that the total capacity of selected controllers discounted by β in (7) must be larger than the total request arrival rate.

This modified K-center approach will be further compared with other methods in Sections VI-C and VI-D. Obviously K-center does not explicitly handle the CSP. Given two controllers with similar propagation latencies but significantly different capacities, K-center will choose the controller with less propagation latency despite its low capacity, resulting in increased average network response time. Furthermore, K-center overlooks the control plane utilization.

B. Genetic Algorithm

Recently, Evolutionary Computation (EC) techniques have been widely exploited to effectively solve various NP-hard problems [34], [35], [36]. The promising results reported in the literature inspired us to tackle CPP (5) through an EC method. Among all EC techniques, we prefer Genetic Algorithm (GA) for two main reasons. (1) The solution x to CPP in the form of a fixed-length binary array can be easily implemented by standard chromosomes in GA. (2) Many previous research clearly demonstrated that GA has been widely exploited to effectively solve constrained optimization problems [34], [35], [36]. However, existing studies [34], [35], [36] developed memetic algorithms which combine evolutionary algorithm (e.g., GA) for new solution exploration and local search (e.g., greedy strategy) for existing solution improvement. Different from these works, GD in our algorithm is purely for fitness evaluation instead of improving any existing solutions.

In this study, we follow the GA framework introduced in [37]. We first transform the CPP objective (5)-(9) into a constraint-free optimization problem:

$$\max_{x,P} F_{cpp}(x) = \max_{x,P} \Big\{ u(x) - \hat{F}_{con}(x,P) \Big\} \tag{15}$$

where

$$\hat{F}_{con}(x,P) = F_{con}(P) + \mu_4 \min\{0, t_{th} - t_{avg}(P)\}$$

In line with the CPP objective in (15), the fitness value of any solution x can be calculated as $F_{cpp}(x)$. Given a candidate solution x to CPP, the control plane utilization $u(x)$ can be directly obtained via (4). Additionally, the solution P to (10) will be obtained from the GD-based scheduling algorithm. Consequently, $F_{cpp}(x)$ can be easily calculated.

After evaluating the fitness of all solutions in the current population, a new population is created by performing genetic operators (e.g., crossover and mutation) on the current population. The whole process repeats over multiple generations until the maximum number of generations is reached. In terms of scalability, the fitness evaluation of all candidate solutions in a GA population can be performed in parallel.

C. Clustering-Based Genetic Algorithm

In order to effectively solve CPP in large networks, both the population size and the generation number in GA need to be extremely large, significantly prolonging the algorithm running time. To reduce the complexity of the search space in CPP, we adopt the famous divide-and-conquer strategy [38] and develop the Clustering-based Genetic Algorithm (CGA) in this paper. In particular, network partitioning is first applied to group network nodes (switches) into k sub-networks according to their mutual distance. Afterwards, GA is utilized to solve CPP in each sub-network. This approach allows us to effectively control the maximum propagation latency within each sub-network. Furthermore, the search space complexity of GA can be significantly reduced. For example, in a network with M nodes, the complexity of the search space is $\mathcal{O}(2^M)$. After network partitioning, each sub-network has on average $\frac{M}{k}$ nodes. In this case, the complexity of the search space is reduced to $\mathcal{O}(k2^{\frac{M}{k}})$. With network partitioning, no schedulers will be overwhelmed by requests generated within each sub-network that they belong to.

Following this idea, the goal of network partitioning is to divide a network $G = <V, E, D>$ into k non-overlapping sub-networks $G_i =< V_i, E_i, D > (i = 1, \ldots, k)$ so that the intra-sub-network latency can be minimized:

$$\min_{G_1,\ldots,G_k} \sum_{i=1}^{k} \sum_{v \in V_i} D(v, c_i) \tag{16}$$

$$\text{s.t.} \bigcup_{i=1}^{k} V_i = V, \quad \forall i \tag{17}$$

$$V_i \bigcap V_j = \phi, \quad \forall i \neq j \tag{18}$$

where $c_i \in C$ is the center of sub-network G_i. (17) and (18) ensure complete network partitioning. After partitioning, the requests generated within a sub-network will be processed most of the time by controllers deployed in the same sub-network.

Inspired by the previous research [8], the Clustering-based Network Partition Algorithm (CNPA) is adopted in this paper to tackle the network partitioning problem. CNPA first initializes C as an empty set. In other words, CNPA considers the whole network as a sub-network and starts by choosing a node with minimal total propagation latency given in (19) as the initial center c which will be added into C:

$$c = \operatorname*{argmin}_{v \in V_i} \sum_{v' \in V_i} D(v, v') \tag{19}$$

Afterwards, the node that has the highest propagation latency with existing centers C is chosen from remaining nodes as a new center c' and subsequently added into C.

$$c' = \operatorname*{argmax}_{v \in V} \sum_{c \in C} D(v, c) \tag{20}$$

After determining the first two centers according to (19) and (20) respectively, other network nodes will be grouped into two sub-networks based on their respective propagation latency from the two centers. Note that after all nodes are assigned, the centers will be recalculated based on (19) and updated in C. After that, if the number of sub-networks is less than k, a new center will be selected using (20). The whole process repeats until k sub-networks have been obtained. The effectiveness of CNPA has been widely demonstrated [39].

The whole process of CGA is summarized as follows: Specifically, CGA uses CNPA to partition the network into k sub-networks. For each sub-network G_i, GA will be utilized to find the CPP solution x_i. Because x_i for each sub-network is independent with each other, multiple GA runs can be performed in parallel, ensuring the overall scalability of CGA.

Note that a controller may miss the most up-to-date global network information because of weak consistency. However, it does not mean that the local information maintained by each controller is not useful. Especially with the help of network partitioning, a controller can easily obtain the latest information about its sub-network which is sufficient to make routing decisions within its sub-network.

D. Clustering-Based Genetic Algorithm With Cooperative Clusters

Despite of CGA's benefits of preventing long-distance switch-controller communication, when the traffic within a sub-network suddenly and substantially increases, existing controllers in the sub-network cannot handle the extra workload, which may significantly slow down the control plane. Hence, in the event of high workload in a sub-network, it is a common practice to seek help from remote controllers.

Specifically to cope with the traffic burst, we can adopt either a proactive or reactive approach. For the proactive approach, a fine-tuned sophisticated traffic prediction model is required to accurately predict the future traffic trend based on historical data. Apart from that, the proactive approach requires us to develop complicated techniques for planning and uncertainty handling. This is beyond the scope of the controller placement problem in this work. Instead, a simple-yet-effective reactive offloading scheme is developed here.

CGA with Cooperative Clusters (CGA-CC) extends CGA to facilitate workload sharing between adjacent sub-networks. For this purpose, we must decide which adjacent controllers should process burst requests. Driven by this requirement, a greedy but efficient approach with two steps have been developed. The first step is to identify the controller candidates to handle the extra requests. In this paper, we take advantage of the piggybacking mechanism. Inspired by [40], a no-forward flag plus extra bits for workload information can be piggybacked in the Type of Service (ToS) field (or option field) in the IP header of every controller response packet. In this way, candidate controllers can be identified effortlessly.

The second step of our greedy approach is controller selection. We first rank the candidate controllers based on their latency to the overloaded sub-network G_j, which is defined as the total propagation latency between the controller node and all nodes in G_j. Controllers with low latency will be added into the deployed controller set of G_j until the total controller capacity matches the processing demand of bursting requests.

Upon obtaining the controller set, we proceed to distribute the requests among chosen controllers by solving the corresponding CSP through the GD-based scheduling algorithm. To avoid overloading any controllers, we only use the remaining capacities of these controllers as the input to the scheduling algorithm.

Compared to a proactive approach, our approach does not rely on any sophisticated prediction model and the overall workload-offloading process is simple and effective. It also provides operational flexibility by allowing the network operators to control the threshold for kicking start the workload-offloading process.

Note that in rare cases when controllers in neighboring sub-networks must lend a helping hand to the overloaded sub-network, those helping controllers can receive updates from the sub-network under question more frequently than usual. Due to this reason, CGA-CC eliminates the necessity for strong network-wide controller consistency and incurs no extra synchronization cost.

VI. EVALUATION

Given that no physical WAN (the central focus of this paper) testbed is currently accessible to us, simulation is adopted as the main evaluation tool due to its flexibility, feasibility, and validity. This section presents the performance evaluation of our proposed algorithms. In particular, after introducing the simulation setting, the effectiveness of the GD-based scheduling approach is demonstrated in Section VI-B and the performance of GA, CGA and CGA-CC is shown in Sections VI-C–VI-E. We also compare CGA-CC with MSPA in Section VI-F. The simulation setup[2] and our algorithm implementation[3] have been uploaded to Github.

A. Evaluation Setup

Simulator: The simulation starts with an idle network (all controllers are idle and no packets are transmitted in the network). Immediately after the simulation starts, the requests are generated randomly based on a pre-determined fixed request arrival rate for each network node. The simulation runs for 240 seconds. We observed in our simulation study that after 10 seconds of simulation, the throughput of the network reaches a stable level. Thus, 240 seconds is considered to be sufficiently long for the network to enter and stay in a stationary condition. Whenever a packet is received at the data plane, the following packet dispatching flow is simulated: (1) The switch forwards the requests to its scheduler. Each request is associated with a timestamp T_{gnr} indicating its generation time from the switch. (2) Upon receiving requests, the scheduler dispatches them to controllers chosen by the scheduling algorithm. (3) The controller sends a response back to the scheduler after it finishes processing a request. (4) The scheduler directs the responses back to the switch. (5) At time T_{rcv} when a response is received by the switch, the response time of the request under question will be calculated as $T_{rcv} - T_{gnr}$. The whole simulation keeps running until the simulation time reaches 240 seconds. More details can be found in our simulation code.[2]

Topology: All simulations will be conducted using the topology information provided by Sprint [17] including the propagation latency, network links, and the numbers of switch

[2]https://github.com/VictoriaWong/sdn-simulator

[3]https://github.com/VictoriaWong/SDN-Controller-Placement

nodes. The sizes of the networks are 14 nodes with 23 links (Asia Sprint network), 15 nodes with 22 links (Europe Sprint network), and 82 nodes with 1056 links (i.e., global Sprint network) respectively, which are comparable to the widely adopted Internet2 OS3E topology (34 nodes with 42 links) in the existing literature [7].

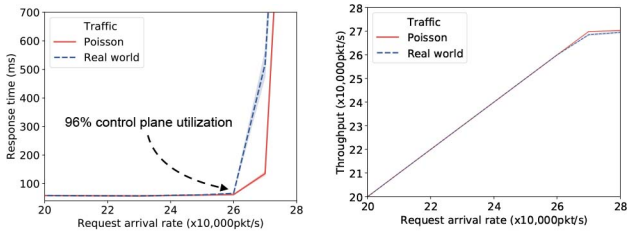
Each node is responsible for handling all backbone related communication requests generated by local switches. Furthermore, a broad spectrum of request arrival rates has been considered in our simulation study. For example, to demonstrate the performance of different CPP approaches, the request arrival rate ranges from 220 k pkt/s to 620 k pkt/s in the Asia Sprint Network, see Fig. 4. The propagation latency between any two nodes is calculated by Dijkstra's algorithm [41]. A set of heterogeneous controllers with capacities ranging from 60 k to 120 k pkt/s is considered.

Apart from that, the control plane traffic consists of (C1) the inter-controller traffic and (C2) the switch-controller traffic. As pointed out by existing studies [13], C2 is generally regarded as the most significant part of the control plane workload. Following our problem formulation in Section III, the impact of C2 is taken care of by the synchronization cost in (1).

Parameter setting: During our simulation, we tried different parameter settings for both GA and GD. For example, the population size for GA is tested from 10 times to 50 times of the total number of nodes in the network. The maximum generation number for GA is tested from 100 to 500. The number of iterations run by GD is tested from 10 to 50. But we did not notice significant performance differences. The results reported in this section are based on the following setting: GA uses a population size of 10 times of the total number of network nodes and evolves for up to 100 generations. For each generation, the mutation and crossover rates are 0.1 and 1 respectively. Roulette wheel selection with elitism is used for selection. For CSP, GD is performed for 10 iterations, the decay factor β is set to 0.85, and the synchronization factor γ is set to 0.1 following existing studies [24].

Trace: Note that using traffic trace datasets is not flexible because they were captured at a certain arrival rate while different arrival rates are required to evaluate the performance of our algorithms. To address this issue, a network traffic generator was implemented in this work to simulate the real world traffic as described in Section III. To justify the performance of our traffic generator, we compared the network performances obtained by using the traffic it generated and the real-world network traces [42].

As demonstrated in Fig. 2, the corresponding throughput achieved is indistinguishable regardless of whether real-world or artificial traffic is used. In addition, their response time is indistinguishable when the control plane utilization is below 96%. After that, significant difference can be spotted. In particular, when the arrival rate reaches 270 k pkt/s, the response time of real-world traffic shows larger fluctuation (400 ms - 600 ms) compared to simulated traffic (100 ms - 150 ms). Note that our problem formulation presented in Section III-B carefully preserves a small portion of capacity on each controller as determined by β. Thus, as long as a controller's workload does not exceed a certain threshold, we can confirm that any performance deviation introduced by the use of our traffic generator in the simulation studies is marginal.



(a) Response time (b) Throughput

Fig. 2. Performance comparison with real-world traffic and traffic generated by Poisson distribution.

TABLE I
AVERAGE RESPONSE TIME (MS) IN EUROPE AND ASIA SPRINT NETWORK

Arrival rate (x10k pkt/s)		15	19	23	27
Fig. 3(a)	CWRR	34.47 ± 0.05	34.60 ± 0.06	34.96 ± 0.04	147.28 ± 33.52
	CDWRR	25.30 ± 0.03	26.14 ± 0.02	4654.18 ± 105.02	\
	GD	**25.18 ± 0.02**	**25.76 ± 0.03**	**28.48 ± 0.07**	144.78 ± 28.12
Fig. 3(b)	CWRR	28.37 ± 0.05	28.55 ± 0.04	29.12 ± 0.05	149.26 ± 34.05
	CDWRR	20.46 ± 0.03	21.25 ± 0.05	6936.75 ± 190.05	\
	GD	**20.35± 0.02**	**20.96± 0.05**	**21.79± 0.06**	145.26± 35.05
Fig. 3(c)	CWRR	87.12 ± 0.05	87.17 ± 0.02	87.38 ± 0.01	138.37 ± 31.48
	CDWRR	**52.86 ± 0.03**	54.30 ± 0.35	2704.84 ± 18.46	\
	GD	53.96 ± 0.02	**53.54±0.01**	**57.66±0.03**	138.37± 31.73
Fig. 3(d)	CWRR	55.02 ± 0.07	55.14 ± 0.06	55.41 ± 0.04	115.90 ± 17.12
	CDWRR	**37.78 ± 0.03**	42.38 ± 1.12	9686.49 ± 65.38	\
	GD	39.21 ± 0.04	**39.24 ± 0.05**	**38.53± 0.05**	108.9 ± 15.15

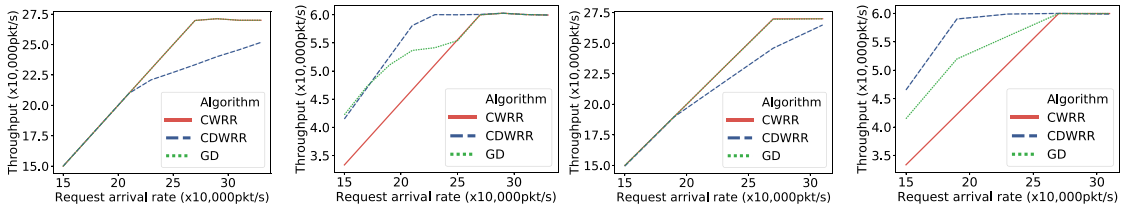
B. Effectiveness of GD for CSP

We first compare the performance of our proposed GD-based approach with CWRR and CDWRR. The simulations are conducted on two networks (i.e., Europe and Asia Sprint Networks) with different geographic coverage. In particular, the largest one-way propagation latency in Europe is 59 ms while it is 202 ms in Asia. Thus, the advantage of using GD is expected to be more significant in Asia. For both networks, 3 controllers with capacities 60 k, 90 k, and 120 k pkt/s respectively are deployed using K-center.

1) Overall Network Performance: From Fig. 3 and Table I, we can clearly notice that GD achieved the lowest response time and highest throughput on both topologies.

Specifically, Table I shows that in Europe network, the response time of both GD and CDWRR slightly increases from 25 ms to 26 ms in accordance with increasing incoming traffic when the request arrival rate is less than 190 k pkt/s due to low controller utilization. We also notice that the response time of both GD and CDWRR is 26% lower than CWRR, which agrees well with our expectation that distributing requests solely relying on the controller capacities is inappropriate.

Meanwhile, the response time of CDWRR soars up as the request arrival rate reaches 230 k pkt/s. In comparison, the sharp increase in the response time of both GD and CWRR does not occur until the arrival rate exceeds 270 k pkt/s. Obviously, the sudden growth in response time is due to the heavily loaded controllers. Note that the amount of requests sent to a controller in CWRR is only proportional to its capacity, which effectively prevents overloading any controller at an



(a) Overall performance (b) Individual performance (c) Overall performance (d) Individual performance

Fig. 3. Performance comparison of different scheduling algorithms for solving the controller scheduling problem in two different networks. (a) and (c) show the overall network performance in Europe and Asia Sprint Network respectively. (b) and (d) are the network performance of one controller in Europe and Asia Sprint Network respectively.

early stage. GD guarantees that all controllers are not overloaded by fulfilling the constraint in (7) when solving the optimization problem. Thus, the response time of both GD and CWRR remains low until the arrival rate increases to 270 k pkt/s. On the other hand, when CDWRR is used, more requests will be sent to nearby controllers, quickly overloading them.

2) Individual Controller Performance: To verify whether the sharp increase in the response time of CDWRR comes from overloading a controller, we measure the average response time and throughput of the controller with low capacity of 60 k pkt/s. It can be seen from Fig. 3(b) that at the arrival rate of 230 k pkt/s, the throughput of CDWRR almost reaches 60 k pkt/s, implying that the controller is fully-loaded. Simultaneously, a dramatical growth in response time of CDWRR can also be noticed from Table I. Comparatively, only moderate increase in response time has been witnessed for both GD and CWRR in Table I. Similar results have also been obtained in the Asia Sprint Network as depicted in Fig. 3(c) and 3(d).

3) Asia vs. Europe Sprint Network: It is important to note that the gap in response time between GD and CWRR in Asia (30 ms in Table I) is significantly larger than the gap in Europe (9 ms) as we expected. Particularly, with the help of GD, the response time in Asia is successfully reduced from 87 ms to less than 60 ms (37.5% lower). In comparison, only 26% reduction in response time is achieved by GD in Europe. This is because the weights (11) used in CWRR only consider the controller capacity. Thus, CWRR is more suitable in a network with similar or negligible propagation latency. However, in a large-scale network (e.g., Asia), the disadvantage of only considering controller capacity becomes significant. Alternatively, GD can jointly and systematically consider multiple factors including response time, propagation latency, and controller capacity. This explains why GD can solve CSP with the highest effectiveness. Due to GD's clear performance advantage over other scheduling methods, for the remaining simulation studies, we will consistently use GD to schedule request processing in the control plane.

C. Effectiveness of GA for CPP

To demonstrate the effectiveness of GA, we conduct a set of simulations on Asia Sprint Network using 4 different controller settings as shown in Table II. The controller capacities are set to either 60 k or 90 k pkt/s in each setting. To ease the discussion, we start with all controllers with the same capacities of 60 k pkt/s as shown in setting 1 in Table II. Then we gradually upgrade some controllers to larger capacities (90 k pkt/s). In order to evenly allocate controllers with different capacities, the network is divided into 4 regions enclosed in every red circle drawn in Fig. 5. One controller within each region has the upgraded capacity of 90 k pkt/s in setting 2. Sequentially, two controllers within each region are chosen and upgraded in setting 3. Finally all controllers are upgraded in setting 4.

TABLE II
CONTROLLER SETTINGS IN ASIA AND GLOBAL NETWORKS

Region		Total	Number of Controllers							
			Setting 1		Setting 2		Setting 3		Setting 4	
			90K	60K	90K	60K	90K	60K	90K	60K
Asia	India region	5	0	5	1	4	2	3	5	0
	Singapore region	3	0	3	1	2	2	1	3	0
	Hong Kong region	3	0	3	1	2	2	1	3	0
	Japan region	3	0	3	1	2	2	1	3	0
Global		82	0	82	18	64	42	40	\	\

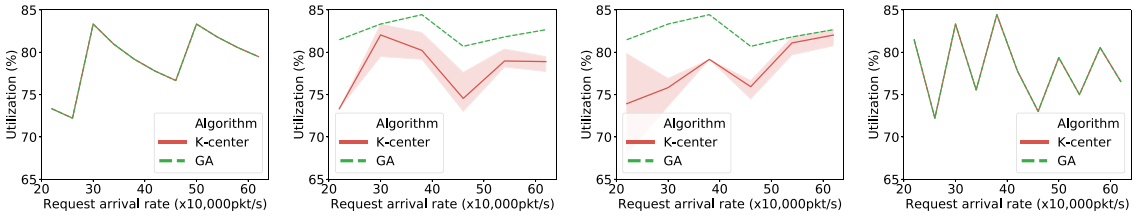
TABLE III
CONTROL PLANE AVERAGE RESPONSE TIME (MS) WITH DIFFERENT CONTROLLER SETTINGS AND REQUEST ARRIVAL RATES IN ASIA SPRINT NETWORK

Arrival rate (x10k pkt/s)		22	38	54
Setting 1	K-center	79.03 ± 3.99	76.52 ± 0.65	73.46 ± 0.27
	GA	**56.55 ± 0.27**	**61.48 ± 0.12**	**65.32± 0.35**
Setting 2	K-center	78.94 ± 4.06	76.39 ± 0.83	72.54 ± 0.22
	GA	**57.66 ± 0.11**	**59.42± 0.25**	**63.01± 0.43**
Setting 3	K-center	83.72 ± 4.55	78.85 ± 1.88	76.34 ± 3.12
	GA	**58.65 ± 0.33**	**61.54 ± 0.36**	**64.0± 0.25**
Setting 4	K-center	85.39 ± 5.77	79.12 ± 4.00	76.18 ± 0.67
	GA	**55.79 ± 0.22**	**56.92 ± 0.17**	**61.01± 0.15**

We compare the throughput achieved by K-center and GA and the results show no significant difference regardless of the controller settings and arrival rates. For the remaining simulation studies, if the throughputs achieved by different placement algorithms are identical, the corresponding results will be omitted.

1) GA on Settings With Identical Controllers: We measure the control plane utilization and response time in network settings where all controllers have identical capacities (setting 1 and setting 4 in Table II). In Fig. 4(a) and Fig. 4(d), both GA and K-center achieve the same control plane utilization. Nevertheless, GA outperforms K-center in terms of response time as shown in Table III.

In general, given the same traffic demand (i.e., request arrival rate), the minimal required control plane capacities will be the same. Since all controllers have identical capacities in setting 1 and setting 4, given the same traffic demand, it agrees



(a) Setting 1 (b) Setting 2 (c) Setting 3 (d) Setting 4

Fig. 4. Performance comparison between K-center and GA for solving the CPP using different controller settings in Asia Sprint Network. (a)-(d) are the network performance with controller setting 1 to 4 in Table II respectively.

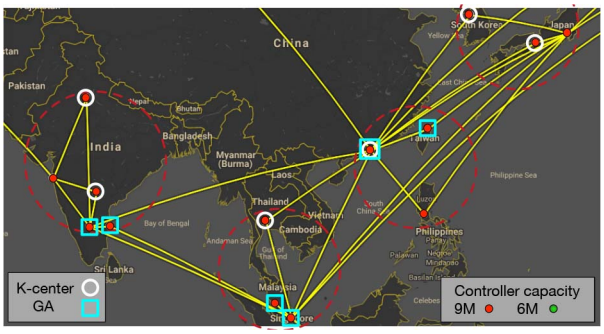


Fig. 5. Controller placement in Asia Sprint Network with controller setting 4 in Table II at the arrival rate of 420 k pkt/s.

TABLE IV
CONTROL PLANE AVERAGE RESPONSE TIME (MS) WITH DIFFERENT CONTROLLER SETTINGS AND REQUEST ARRIVAL RATES IN GLOBAL SPRINT NETWORK

Arrival rate (x10k pkt/s)		100	200	300
Setting 1	K-center	174.05 ± 5.45	153.36 ± 3.93	140.87 ± 2.57
	GA	120.9 ± 0.37	119.13 ± 0.44	114.51 ± 0.28
	CGA	**90.02± 0.16**	**75.92 ± 0.34**	**35.83 ± 0.22**
Setting 2	K-center	175.26 ± 8.03	158.18 ± 3.05	145.43 ± 4.83
	GA	119.63 ± 0.21	120.02 ± 0.59	109.92 ± 0.33
	CGA	**84.59± 0.29**	**79.81± 0.30**	**37.73 ±0.31**
Setting 3	K-center	174.09 ± 2.97	160.92 ± 1.52	145.26 ± 2.76
	GA	115.8 ± 0.21	121.56 ± 0.40	110.31 ± 0.37
	CGA	**85.16 ± 0.17**	**83.37±0.21**	**38.12±0.35**

with our expectation that both GA and K-center decided to deploy the same number of controllers. In other words, their control plane utilization will be identical.

Although the utilization is indistinguishable, GA outperforms K-center in response time. To investigate the cause, we visualize the placement results of both GA and K-center in setting 4 at the arrival rate of 420 k pkt/s, which is shown in Fig. 5. We find that K-center tends to select controllers scattered on the periphery of the network while GA prefers controllers located in the centers with multiple links connected to other nodes. This is mainly because K-center allocates controllers to minimize the worst case propagation latency and controllers at the network boundary are likely to have longer propagation latency. However, due to their remote locations, requests sent to these controllers have to travel a long distance, which inevitably increases the response time. On the other hand, GA strategically deploys controllers to intersection locations where multiple links join together in the network. As a result, most of the requests can be sent directly to the controllers without traveling through other network nodes. Thus, even though the same number of controllers is deployed by GA and K-center, GA can effectively lower the response time by up to 25%.

2) GA on Settings With Different Controllers: As shown in Fig. 4(b), Fig. 4(c) and Table III, when controllers have varied capacities (i.e., setting 2 and setting 3 in Table II), GA can handle CPP more effectively than K-center in terms of either utilization or response time as we expected.

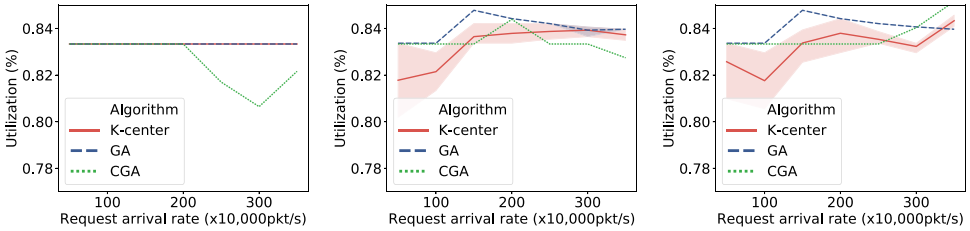
D. Effectiveness of CGA for CPP

To further increase the difficulty of CPP, a larger network (i.e., the global Sprint Network [17]) is adopted. Similar to Section VI-C, two optional controller capacities (60 k and 90 k pkt/s) are adopted during the simulation. The network will be first divided into 2 sub-networks when the arrival rate remains below 2000 k pkt/s. With further increase in arrival rate, the number of sub-networks will be doubled to 4.

Fig. 6 and Table IV show the results of all competing placement algorithms with three different controller settings summarized in Table II. From Fig. 6 and Table IV, we can see that CGA can achieve much lower response time than GA and K-center without substantially lowering the control plane utilization. In the rest of this subsection, we will examine the performance with respect to every network setting in detail.

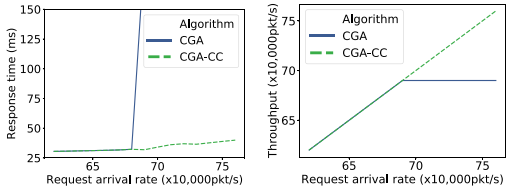
1) CGA on Settings With Identical Controllers: Similar to Fig. 4(a) and Fig. 4(d), we notice that in Fig. 6(a), both K-center and GA achieved the same utilization due to identical controller capacities. Apart from that, GA can reduce the response time by up to 30% compared with K-center. Meanwhile, the utilization of CGA is slightly lower than GA when the request arrival rate is at 250 k pkt/s. This is because the whole network is now divided into 4 rather than 2 sub-networks. Apparently, more sub-networks demand for more controllers. Nevertheless, CGA managed to drastically reduce the response time by 50% (from 80 ms to 40 ms).

2) CGA on Settings With Different Controllers: When controllers have different capacities (i.e., setting 2 and 3 in Table II), CGA becomes more competitive in terms of both utilization and response time. For example, compared to GA, the response time of CGA is significantly reduced by up to 66%. Meanwhile, the gap in utilization between GA and CGA is narrowed down to 1% in Fig. 6(b). Another interesting phenomenon is that in Fig. 6(c), the utilization of CGA keeps increasing and even outperforms GA at the request arrival rate



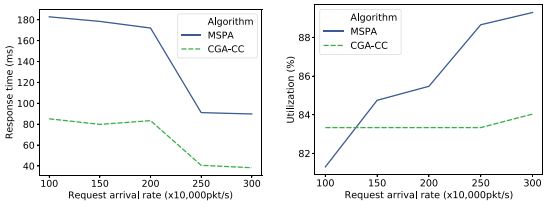
(a) Setting 1 (b) Setting 2 (c) Setting 3

Fig. 6. Performance comparison between K-center, GA, and Clustering-based GA for solving CPP using different controller settings in Global Sprint Network. (a)-(c) are the network performance with controller setting 1 to 3 in Table II respectively.



(a) Response time (b) Throughput

Fig. 7. Performance comparison with burst traffic.



(a) Response time (b) Utilization

Fig. 8. Performance comparison without burst traffic using Setting 3 in Table II with different request arrival rates.

of 3,500 k pkt/s. Our results show that CGA enjoys higher chance of identifying better solutions in large search spaces.

E. Effectiveness of CGA-CC for CPP

In the simulations discussed previously, we focus mainly on CGA because no bursting requests occur in the simulated networks. In this situation, workload sharing across different sub-networks does not help. In this Subsection, however, we want to evaluate the usefulness of supporting collaborative sub-networks through CGA-CC.

To demonstrate the effectiveness of CGA-CC on coping with unexpected traffic burst, we deploy a fixed collection of controllers in the Europe network with setting 3 in Table II and constantly increase the request arrival rate. When the arrival rate reaches the control plane capacity, we expect that some requests will be forwarded to controllers in neighboring sub-networks to avoid overloading the control plane.

As depicted in Fig. 7, the response time of CGA stays below 32 ms and its throughput shows a steady growth before the arrival rate reaches 680 k pkt/s. After that, a notable jump can be observed in CGA response time.

On the contrary, the response time of CGA-CC is consistent with CGA when the arrival rate is less than 680 k pkt/s since the workload is still below the control plane capacity. After that, the gap in both response time and throughput between CGA and CGA-CC widens in accordance with the increasing arrival rate. It is mainly because CGA-CC strategically offloads the requests to nearby controllers to avoid overloading the control plane. Although CGA-CC selects controllers with the lowest propagation latency, the controllers are still located outside the sub-network, which introduces a considerable propagation latency in response time. Thus, an upward trend (but substantially less severe than in CGA) in CGA-CC response time can be spotted from Fig. 7(a).

F. Comparison With MSPA

To further demonstrate the effectiveness of CGA-CC, we compare it with a recently proposed realistic and robust algorithm called Multi-controller Selection and Placement Algorithm (MSPA) [8].

For a fair comparison, we adopt the same number of sub-networks in the network partitioning step. Given the partitioning result, MSPA calculates the number of controllers needed in each sub-network based on a given threshold $\hat{t}_{th}$ on controller processing time and deploys the controllers using CNPA. We evaluated a range of $\hat{t}_{th}$ from 0.01 ms to 5 ms and the results reported in this section are based on the $\hat{t}_{th}$ value with the best performance (0.5 ms).

Apart from this, MSPA requires to establish a central scheduler in each sub-network, as we explained in Section II. However, the authors in MSPA did not explain their method of selecting the location for the scheduler. To find the "best" scheduler location, we tried all possible locations within each sub-network and reported the best performance.

1) CGA-CC vs. MSPA Without Burst Traffic: Fig. 8 demonstrates the performance comparison between CGA-CC and MSPA. Specifically, in comparison to MSPA, CGA-CC can significantly reduce the average request response time. For example, when the request arrival rate is 1000 k pkt/s, the response time of MSPA is 182.78 $\pm$ 0.06 ms while CGA-CC is significantly smaller than 100 ms. This is mainly for two reasons. First, within each sub-network, MSPA sends all requests generated from the switches to a single scheduler first instead of controllers directly, which inevitably increases the propagation latency. Second, the M/M/c queuing model adopted in MSPA considers neither the capacity differences between controllers nor the distance between the scheduler and controllers. Thus, requests can be sent to controllers with low capacity and long propagation latency. This can be verified in Figure 9.

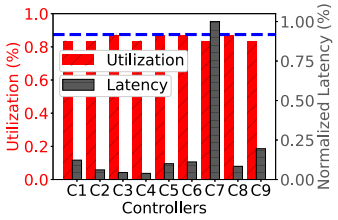
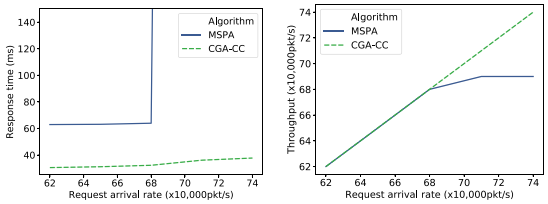


Fig. 9. Controller utilization among all selected controllers in Europe network and their normalized propagation latency with the scheduler in MSPA.



(a) Response time (b) Throughput

Fig. 10. Performance comparison with burst traffic using the fixed controller placement from Europe network.

Although CGA-CC outperforms MSPA in terms of average response time, we can see that MSPA achieves higher control plane utilization when the arrival rate is higher than 1500 k pkt/s. In comparison, CGA-CC maintains a stable control plane utilization at around 84%. This is mainly because of the controller capacity constraint in our problem formulation (7) which ensures that a proportion of capacity must be reserved for the control plane to withstand unexpected traffic bursts.

2) CGA-CC vs. MSPA With Burst Traffic: We also evaluate the performance of MSPA with burst traffic. As shown in Fig. 10, the performance of MSPA follows a similar trend as CGA in Fig. 7 which is easily understandable since both MSPA and CGA do not share workload across sub-networks.

VII. Conclusion

In this paper, we introduce a new controller placement and scheduling problem (CPSP) that explicitly strengthens the importance of solving both the controller placement problem (CPP) and the controller scheduling problem (CSP) coherently within the same framework. CPSP is mathematically described as a constrained optimization problem aiming to optimize control plane utilization while simultaneously guaranteeing low network response time. Therefore, to tackle CPSP, we must seek a combination of solutions to both CPP and CSP.

To address CSP scalably and effectively, we focus on optimizing the probabilities for request distribution over all controllers. A gradient-descent-based (GD) scheduling algorithm was subsequently developed to balance the trade-off between scheduling performance and problem scalability.

In line with the solution of CSP, a Clustering-based Genetic Algorithm with Cooperative Clusters (CGA-CC) was proposed to address CPP. In particular, to reduce the search space of GA, CGA-CC split the network into non-overlapping sub-networks so that GA can effectively deploy controllers within each sub-network. Moreover, to alleviate the impact of unexpected bursting requests in any sub-network, a greedy algorithm was developed to strategically offload indigestible requests to adjacent sub-networks.

Extensive simulations have been conducted based on real-world topologies and traffic. The results showed that our algorithms can significantly outperform several existing algorithms, including a recently proposed approach called Multi-controller Selection and Placement Algorithm (MSPA). Note that it is interesting to further evaluate the performance of our algorithms on a real-world testbed. This is an important direction for our future research.

References

[1] W. Kellerer, P. Kalmbach, A. Blenk, A. Basta, M. Reisslein, and S. Schmid, "Adaptable and data-driven softwarized networks: Review, opportunities, and challenges," *Proc. IEEE*, vol. 107, no. 4, pp. 711–731, Apr. 2019.

[2] S. Kuklinski and P. Chemouil, "Network management challenges in software-defined networks," *IEICE Trans. Commun.*, vol. 97, no. 1, pp. 2–9, 2014.

[3] S. Jain *et al.*, "B4: Experience with a globally-deployed software defined WAN," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 3–14, 2013.

[4] P. Berde *et al.*, "ONOS: Towards an open, distributed SDN OS," in *Proc. ACM HotSDN*, 2014, pp. 1–6.

[5] T. Koponen *et al.*, "ONIX: A distributed control platform for large-scale production networks." in *Proc. USENIX Symp. Oper. Syst. Design Implement.*, vol. 10, 2010, pp. 1–6.

[6] M. T. I. U. Huque, W. Si, G. Jourjon, and V. Gramoli, "Large-scale dynamic controller placement," *IEEE Trans. Netw. Service Manag.*, vol. 14, no. 1, pp. 63–76, Mar. 2017.

[7] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in *Proc. ACM HotSDN*, 2012, pp. 7–12.

[8] G. Wang, Y. Zhao, J. Huang, and Y. Wu, "An effective approach to controller placement in software defined wide area networks," *IEEE Trans. Netw. Service Manag.*, vol. 15, no. 1, pp. 344–355, Mar. 2018.

[9] P. Vizarreta, C. M. Machuca, and W. Kellerer, "Controller placement strategies for a resilient SDN control plane," in *Proc. IEEE 8th Int. Workshop Resilient Netw. Design Model. (RNDM)*, 2016, pp. 253–259.

[10] M. T. I. U. Huque, G. Jourjon, and V. Gramoli, "Revisiting the controller placement problem," in *Proc. IEEE 40th Conf. Local Comput. Netw.*, 2015, pp. 450–453.

[11] S. Lange *et al.*, "Heuristic approaches to the controller placement problem in large scale SDN networks," *IEEE Trans. Netw. Service Manag.*, vol. 12, no. 1, pp. 4–17, Mar. 2015.

[12] J. Cui, Q. Lu, H. Zhong, M. Tian, and L. Liu, "A load-balancing mechanism for distributed SDN control plane using response time," *IEEE Trans. Netw. Service Manag.*, vol. 15, no. 4, pp. 1197–1206, Dec. 2018.

[13] 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.

[14] M. Obadia, M. Bouet, J.-L. Rougier, and L. Iannone, "A greedy approach for minimizing SDN control overhead," in *Proc. IEEE Conf. Netw. Softw.*, 2015, pp. 1–5.

[15] V. Huang, G. Chen, Q. Fu, and E. Wen, "Optimizing controller placement for software-defined networks," in *Proc. IFIP/IEEE Integr. Netw. Service Manag.*, 2019, pp. 224–232.

[16] V. Huang, Q. Fu, G. Chen, E. Wen, and J. Hart, "BLAC: A bindingless architecture for distributed SDN controllers," in *Proc. IEEE Conf. Local Comput. Netw.*, 2017, pp. 146–154.

[17] *Sprint Network*. Accessed: Jan. 15, 2019. [Online]. Available: https://www.sprint.net/performance/

[18] A. A. Dixit, F. Hao, S. Mukherjee, T. V. Lakshman, and R. R. Kompella, "ElastiCon: An elastic distributed SDN controller," in *Proc. ACM/IEEE Archit. Netw. Commun. Syst.*, 2014, pp. 17–28.

[19] D. Hock, M. Hartmann, S. Gebert, M. Jarschel, T. Zinner, and P. Tran-Gia, "Pareto-optimal resilient controller placement in SDN-based core networks," in *Proc. IEEE Int. Teletraffic Congr.*, 2013, pp. 1–9.

[20] M. F. Bari *et al.*, "Dynamic controller provisioning in software defined networks," in *Proc. IEEE Conf. Netw. Service Manag.*, 2013, pp. 18–25.

[21] *Data Center Facility of the Future Is a Hybrid*. Accessed: Dec. 12, 2012. [Online]. Available: https://searchcio.techtarget.com/feature/Data-center-facility-of-the-future-is-a-hybrid
[22] E. Sakic, F. Sardis, J. W. Guck, and W. Kellerer, "Towards adaptive state consistency in distributed SDN control plane," in *Proc. IEEE Int. Conf. Commun.*, 2017, pp. 1–7.
[23] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, "Logically centralized? State distribution trade-offs in software defined networks," in *Proc. ACM HotSDN*, 2012, pp. 1–6.
[24] 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.
[25] D. Gross, J. F. Shortle, J. M. Thompson, and C. M. Harris, *Fundamentals of Queuing Theory*, 4th ed. Hoboken, NJ, USA: Wiley, 2008, pp. 10–12.
[26] F. A. Kuipers, H. Wang, and P. Van Mieghem, "The stability of paths in a dynamic network," in *Proc. ACM Conf. Emerg. Netw. Exp. Technol.*, 2005, pp. 105–114.
[27] R. V. Oliveira, B. Zhang, and L. Zhang, "Observing the evolution of Internet as topology," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 313–324, 2007.
[28] H. T. Dang, D. Sciascia, M. Canini, F. Pedone, and R. Soule, "Netpaxos: Consensus at network speed," in *Proc. 1st ACM SIGCOMM Symp. Softw. Defined Netw. Res.*, 2015, pp. 1–7.
[29] X. Ye, G. Cheng, and X. Luo, "Maximizing SDN control resource utilization via switch migration," *Comput. Netw.*, vol. 126, pp. 69–80, Oct. 2017.
[30] J. Martin and A. Nilsson, "On service level agreements for IP networks," in *Proc. IEEE INFOCOM*, vol. 2, 2002, pp. 855–863.
[31] *Sprint SLA Performance*. Accessed: Nov. 10, 2019. [Online]. Available: https://www.sprint.net/sla_performance.php?network=sl
[32] N. Ghani, A. Shami, C. Assi, and M. Y. A. Raja, "Intra-ONU bandwidth scheduling in Ethernet passive optical networks," *IEEE Commun. Lett.*, vol. 8, no. 11, pp. 683–685, Nov. 2004.
[33] T. P. Runarsson and X. Yao, "Stochastic ranking for constrained evolutionary optimization," *IEEE Trans. Evol. Comput.*, vol. 4, no. 3, pp. 284–294, Sep. 2000.
[34] W. Higino, A. A. Chaves, and V. V. de Melo, "Biased random-key genetic algorithm applied to the vehicle routing problem with private fleet and common carrier," in *Proc. IEEE Congr. Evol. Comput.*, 2018, pp. 1–8.
[35] A. Arab and A. Alfi, "An adaptive gradient descent-based local search in memetic algorithm applied to optimal controller design," *Inf. Sci.*, vol. 299, pp. 117–142, Apr. 2015.
[36] M. Ghosh, S. Begum, R. Sarkar, D. Chakraborty, and U. Maulik, "Recursive memetic algorithm for gene selection in microarray data," *Expert Syst. Appl.*, vol. 116, pp. 172–185, Feb. 2019.
[37] M. Srinivas and L. M. Patnaik, "Genetic algorithms: A survey," *Computer*, vol. 27, no. 6, pp. 17–26, 1994.
[38] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. Cambridge, MA, USA: MIT Press, 2009.
[39] I. Palomares, L. Martinez, and F. Herrera, "A consensus model to detect and manage noncooperative behaviors in large-scale group decision making," *IEEE Trans. Fuzzy Syst.*, vol. 22, no. 3, pp. 516–530, Jun. 2014.
[40] Q. Fu *et al.*, "Taming the wild: A scalable anycast-based CDN architecture (T-SAC)," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 12, pp. 2757–2774, Dec. 2018.
[41] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.
[42] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. ACM Internet Meas. Conf.*, 2010, pp. 267–280.

Victoria Huang received the M.Eng. degree from Sun Yat-sen University, China, in 2016. She is currently pursuing the Ph.D. degree with the School of Engineering and Computer Science, Victoria University of Wellington, New Zealand. Her research interests include network architecture, resource allocation/scheduling, software-defined networking, evolutionary computation algorithms, and reinforcement learning.

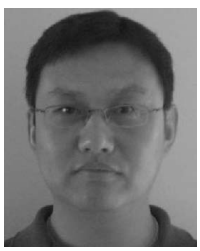
Gang Chen received the Ph.D. degree from Nanyang Technological University, Singapore, in 2006. He is currently a Senior Lecturer with the School of Engineering and Computer Science, Victoria University of Wellington, New Zealand. His research interests include reinforcement learning algorithms for multiagent, and evolutionary computation algorithms and their application to scheduling problems and resource management in networked systems.

Peng Zhang received the Ph.D. degree in computer science from Tsinghua University in 2013. He is currently an Associate Professor with the Department of Computer Science and Technology, Xi'an Jiaotong University, Xi'an, China. His research interests include network verification, network measurement, and software-defined networking.

Hao Li received the Ph.D. degree in computer science from Xi'an Jiaotong University, China, in 2016, where he is currently an Assistant Professor with the Department of Computer Science and Technology. His main research interests include computer networking systems, network measurement and monitoring, and software-defined networking.

Chengchen Hu received the Ph.D. degree from Tsinghua University, China, in 2008. He joined the Department of Computer Science and Technology, Xi'an Jiaotong University (XJTU), China, where he has been served as a Professor since 2016. Since 2017, he has been on leave from XJTU and became the Principal Engineer and the Director leading Xilinx Research Labs Asia Pacific, Singapore. This work was done when he was with XJTU. His main research interests include network measurement, cloud data center networking, and software-defined networking.

Tian Pan received the Ph.D. degree from the Department of Computer Science and Technology, Tsinghua University, Beijing, China, in 2015. He was a Postdoctoral Researcher with the Beijing University of Posts and Telecommunications from 2015 to 2017, where he has been an Assistant Professor since 2017. His research interests include router architecture, software-defined networking, programmable data plane, satellite networks, and machine learning for network applications.



Qiang Fu received the Ph.D. degree in telecommunications engineering from the University of Queensland, Australia. He is currently a Senior Lecturer in network engineering with the School of Engineering and Computer Science, Victoria University of Wellington, New Zealand. His research interests include Internet architecture and protocols, wireless and mobile systems, and network measurement and security.