Achieving Effective Resource Management in Distributed SDN Controller Architectures

by

Guiying Huang

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Doctor of Philosophy
in Computer Science.

Victoria University of Wellington
2020

Abstract

As an emerging computer networking paradigm, Software-Defined Networking (SDN) empowers network operators with simplified network configuration and centralized network management. Recently, distributed controller architectures have become a notable invention where multiple controllers are jointly deployed in the network for request processing. One major research challenge for distributed controller architectures is to effectively manage the controller resources including allocating sufficient controllers to the suitable network locations and making the best use of the given controller resources.

In general, existing approaches for managing the controller resources in the literature can be classified into three main directions. Designing new controller architectures belongs to the first direction, where the focus is on enabling workload shifting among controllers using switch migration. Designing controller placement algorithms to identify the number and locations of controllers is the second direction. Given the controller placement solution, the third direction is controller scheduling which aims to make the best use of the shared controllers by properly distributing requests among them.

However, existing approaches have three major limitations. First, existing controller architectures feature a switch-controller binding which restricts the requests generated by a switch to only be processed by a predefined controller. Since each switch comes with different workload and the workload can be time-variant, the binding renders the bound controller susceptible to either being overloaded or underloaded. Second, existing placement algorithms have consistently underestimated the importance

of controller scheduling. Due to the NP-hardness of the placement problem, Genetic Algorithm (GA) is a promising candidate. However, as a population-based approach, GA can be computationally expensive. Especially in a large network, the corresponding search space becomes too large for GA to handle effectively. Third, existing approaches for controller scheduling are mostly designed under the switch-controller binding constraint. When the scheduling is performed at a per-request level, the scheduling complexity increases significantly, rendering the efficiency and effectiveness of existing algorithms questionable. Apart from that, existing studies mainly focus on manually designing request dispatching policy which strongly relies on domain knowledge and involves a time-consuming fine-tuning process.

The overall goal of this thesis is to effectively manage the controller resources in distributed SDN controller architectures. To address the three major limitations, three research objectives are established. First, this thesis aims to propose a new controller architecture to enable flexible controller placement and scheduling. Second, the thesis focuses on effectively and scalably identifying suitable controller placement while jointly taking the controller scheduling problem into consideration. Third, the thesis seeks to incorporate machine learning techniques in the request dispatching policy design to automatically learn adaptive and effective policies.

To achieve the first objective, this thesis proposes a new BindingLess Architecture for distributed Controllers (BLAC) which features bindingless association between switches and controllers. With the newly introduced scheduling layer, requests can be transparently and flexibly dispatched among multiple controllers without invoking the time-consuming and complicated switch migration. Experiments conducted in this thesis show that BLAC significantly reduces the average response time and improves the throughput compared to existing SDN architectures.

To achieve the second objective, this thesis proposes a Clustering-based Genetic Algorithm with Cooperative Clusters (CGA-CC) to tackle

the controller placement problem. Particularly, CGA-CC partitions a large network into non-overlapping sub-networks to substantially reduce the search space of GA. Within each sub-network, GA is applied to identifying the placement solution. The quality of any given placement solution is evaluated by a gradient-descent-based scheduling algorithm which is developed to optimize the probability distribution of requests among all controllers. Moreover, a greedy load re-distribution mechanism is developed to handle unexpected demand variations by dynamically forwarding indigestible requests to adjacent sub-networks. Extensive simulations show that our algorithms can significantly outperform several existing and state-of-the-art algorithms and is more robust in handling unexpected traffic bursts.

To achieve the third objective, this thesis proposes a Multi-Agent (MA) deep-reinforcement-learning-based approach with the aim to automatically learn adaptive, effective, and efficient policies used by each switch. In particular, a new adaptive policy representation is proposed to support networks with a changing number of controllers. To enable the training of an adaptive policy, a new policy gradient calculation technique is developed. Then the policy design problem is formulated as an MA Markov Decision Processing and a new MA training algorithm is proposed. The results show that the policy designed by our algorithm can easily adapt to networks with a changing number of controllers. Moreover, our policy can achieve significantly better performance compared with existing policies including the man-made policy (e.g., weighted round-robin), the model-based policy (e.g., the gradient-descent-based scheduling algorithm), and policies designed by other reinforcement learning algorithms (e.g., the proximal policy optimization algorithm).

List of Publications

The publications completed during my PhD period are listed below in chronological order

1. **VICTORIA HUANG**, QIANG FU, GANG CHEN, ELLIOTT WEN, AND JONATHAN HART. BLAC: A BindingLess Architecture for Distributed SDN Controllers. In *Proceedings of the 42nd IEEE Conference on Local Computer Networks (LCN 2017)*, October 09–12, 2017, Singapore. IEEE Press. Pages 146–154.

2. **VICTORIA HUANG**, GANG CHEN, QIANG FU, AND ELLIOTT WEN. Optimizing Controller Placement for Software-Defined Networks. In *Proceedings of 2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, April 08–12, 2019, Washington DC, USA. IEEE Press. Pages 224–232.

3. **VICTORIA HUANG**, GANG CHEN, AND QIANG FU. Effective Scheduling Function Design in SDN Through Deep Reinforcement Learning. In *Proceedings of 2019 IEEE International Conference on Communications (ICC)*, May 20–24, 2019, Shanghai, China. IEEE Press. Pages 1–7.

4. **VICTORIA HUANG**, GANG CHEN, PENG ZHANG, HAO LI, CHENGCHEN HU, TIAN PAN AND QIANG FU. A Scalable Approach to SDN Control Plane Management: High Utilization Comes with Low Latency. In *IEEE Transactions on Network and Service Management (TNSM)*, Volumn 17, Issue 2, June, 2020, Pages 682–695.

5. **VICTORIA HUANG**, GANG CHEN, QIANG FU. Multi-Agent Deep Reinforcement Learning for Request Dispatching in Multi-Controller Software-Defined Networking. In *IEEE Transactions on Parallel and Distributed Systems (TPDS)*. (Under review)

Acknowledgments

I would like to express my sincere gratitude to those who have provided me the invaluable support and guidance during my Ph.D. study.

My deepest gratitude goes to my supervisors, Dr. Aaron Chen and Dr. Qiang Fu. Dr. Aaron Chen has provided me substantial support during my Ph.D. journal. He has spent dedicated time and efforts to train my research skills and provided constructive and challenging feedback to improve my research work. Without him, completing this thesis would have been much harder or even impossible. Dr. Qiang Fu is a source of valuable advice and novel ideas. The discussions with him always bring me to a deeper understanding of my research.

In addition, I want to thank my dear friends for their support. Special thanks to my best friends Yiyin Chin and Liyan Pan, who have always unconditionally encouraged me and supported me under any circumstances. Another big thank goes to Jakob Pfender for his constant encouragement and delicious meals. I also wish to thank my friends Jordan Ansell, Elvie Annabelle M, and Kouji Shotiwuth for the board games that helped me relax from my stressful Ph.D. life. I am also grateful to my lab colleagues Muru Raj Odiathevar, Yiming Peng, and Elliott Wen for the fruitful research discussions and knowledge sharing. In addition, I wish to thank my friends in the Evolutionary Computation Research Group (ECRG) and Evolutionary Computation for Combinatorial Optimization Research Group (ECCO) for creating an active research environment. I also want to thank my thesis examiners Assoc. Prof. Ian Welch, Dr. Richard Nelson,

and Assoc. Prof. Maode Ma for their feedbacks, which help me to further improve the quality of my thesis.

Furthermore, I am grateful for the Victoria Doctoral Scholarship, the University Research Fund (220603/1929), and Thesis Submission Scholarship provided by Victoria University of Wellington for their financial support.

Last but not least, I wish to thank my family who has always been there for me, offering their unconditional love, encouragement, and support throughout my life.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

BLAC A BindingLess Architecture for Distributed SDN Controllers.

CCPP Capacitated Controller Placement Problem.

CGA-CC Clustering-based Genetic Algorithm with Cooperative Clusters.

CP Controller Placement.

CPP Controller Placement Problem.

CPSP Controller Placement and Scheduling Problem.

CS Controller Scheduling.

CSP Controller Scheduling Problem.

DNN Deep Neural Network.

DRL Deep Reinforcement Learning.

EA Evolutionary Algorithm.

EC Evolutionary Computation.

GA Genetic Algorithm.

GAE Generalized Advantage Estimation.

GD Gradient Descent.

ISP Internet Service Provider.

MA Multi-Agent.

MA-DRL Multi-Agent Deep Reinforcement Learning.

MA-MDP Multi-Agent Markov Decision Process.

MA-PPO Multi-Agent Proximal Policy Optimization.

MDP Markov Decision Process.

ML Machine Learning.

NFV Network Function Virtualization.

NIB Network Information Base.

NN Neural Network.

PDS Policy Direct Search.

PGS Policy Gradient Search.

RDPD Request Dispatching Policy Design.

RL Reinforcement Learning.

RM Resource Management.

SA Single-Agent.

SA-DRL Single-Agent Deep Reinforcement Learning.

SA-MDP Single-Agent Markov Decision Process.

SDN Software-Defined Networking.

SI Swarm Intelligence.

TCAM Ternary Content Addressable Memory.

TE Traffic Engineering.

TI Training Iteration.

UCPP Uncapacitated Controller Placement Problem.

VIS Value function Indirect Search.

VM Virtual Machine.

VNF Virtual Network Function.

WAN Wide Area Network.

Chapter 1

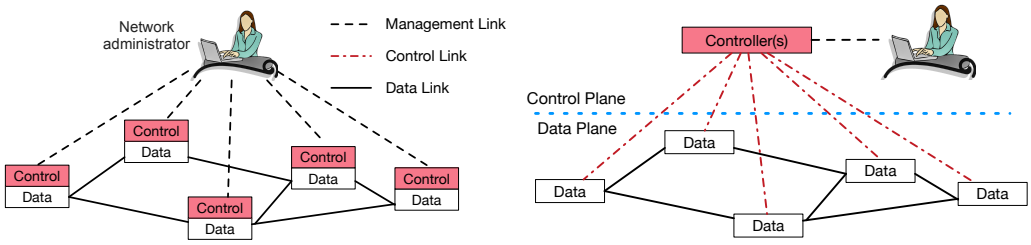
Introduction

This chapter provides a general introduction to the thesis. Specifically, it starts with describing the controller resource management problem in Software-Defined Networking and outlines the motivations. Aiming at addressing the controller resource management problem, the research goals and objectives are proposed, followed by a summary of the major contributions. A brief discussion of the thesis organization concludes this chapter.

1.1 Problem Statement

According to recent statistics in 2019 [7, 23], more than half of the global population uses computer networks and the number of users is still growing steadily [3]. Undoubtedly, computer networks have become indispensable to people in their daily lives.

In computer networks, network devices (e.g., routers and switches) are connected to each other and perform packet forwarding based on routing decisions made by routing algorithms. Everything that network devices do can be classified as being in a particular plane. Generally, there are two planes: the control plane and the data plane. The control plane is in charge of making packet forwarding decisions which guide the packet



(a) Traditional Network (b) SDN

Figure 1.1: Traditional network VS SDN.

forwarding in the data plane. Given the decision, the data plane transfers the received packet from an input link interface to the appropriate output link interface within the network device.

In traditional networks, the control plane and the data plane are highly integrated and embedded in individual network devices as shown in Figure 1.1(a). Each network device is designed with vendor-specific interfaces and predefined protocols, which makes network configuration heavily human-centered with minor autonomous behavior [156]. In some extreme cases, network operators may need to manually and individually configure each device using a limited set of low-level device configuration commands in a command line interface environment. Thus, managing traditional networks is a time-consuming and error-prone task [16, 18, 93, 214].

Software-Defined Networking (SDN) revolutionized computer networking by introducing a new paradigm that decouples the control plane from the data plane, as shown in Figure 1.1(b) [90]. The control plane, which is equipped with one or multiple controllers, serves as the network brain and controls the behaviors of the data plane. As a result, the data plane is "brainless" but highly efficient. Because SDN supports centralized network management and rapid deployment of new network policies, it has been widely applied to many real-world networks (e.g., Google B4 [140] and NTT Com's SD-WAN [9]).

Traditionally, the control plane is equipped with one single controller (e.g., Beacon [84] and Floodlight [5]), which may suffer from responsiveness, resilience, and scalability issues [127, 148, 161, 214]. The conflict between relatively poor controller performance and high network demands has motivated researchers to design distributed controller architectures (e.g., ONOS [45] and Onix [158]) where multiple controllers are physically spread across the control plane to manage different sub-regions of switches. However, these controllers still logically form a centralized control plane.

Regardless of the benefits provided, the introduction of distributed controller architectures also gives rise to new research problems, such as consistency, reliability, and scalability [127, 148, 161, 214]. In particular, since SDN features a logically centralized control plane, it requires each physically distributed controller to maintain a consistent global network view by constantly sharing their local network information. As the network scales up, more controllers will be deployed with more information exchanges, which increases the synchronization overhead [281]. Apart from that, the information shared across the network can be easily intercepted by the attackers, inevitably posing a privacy threat to network management [161, 216].

Among all the research problems, scalability is one of the most critical challenges in distributed controller architectures [129, 148]. As a widely investigated topic in system architecture designs [105, 113, 199], scalability is generally defined as the capability of the system to maintain its functionality and performance in the event of growing demand/workload. Similarly, in the SDN context, scalability is widely understood as the ability of a distributed controller architecture to handle more requests with the increased network scale while simultaneously satisfying network performance requirements (e.g., network response time and throughput) [128, 148].

According to the scalability definition, it is clear that effectively and

efficiently managing the control plane resources[1] is the key to enhancing the scalability. Resource Management (or allocation, scheduling) is a popular topic in various systems. Although the basic notion is intuitive, the term Resource Management (RM) has different definitions in different areas. For example, RM in cloud computing refers to the procedure of distributing the cloud resources (e.g., computing, network, and storage resources) to a set of applications so as to satisfy both consumers' and providers' demand as well as adapting to changes in the availability of resources [52, 190]. In operating systems, RM considers the allocation of system resources (e.g., memory, CPU, and network bandwidth) to different threads, processes, and applications to achieve certain objectives, such as high system throughput, fairness, and quality of service [110].

Similarly, from the perspective of distributed controller architectures, RM is the process of allocating control plane resources to the network as well as performing request dispatching to available shared resources. In other words, it involves decision making with respect to three questions:

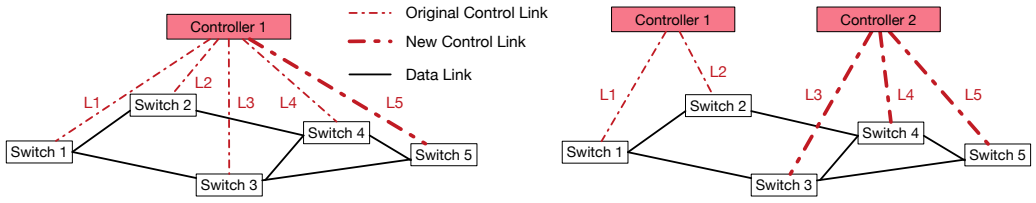
(Q1) *How many* controllers should be allocated to a given network with given controller capacity settings?

(Q2) *Where* should the controllers be placed in the network?

(Q3) *How* to schedule the given controller resources? In other words, how to perform request dispatching from switches to shared controllers?

Particularly, identifying a suitable number of controllers (Q1) and their locations (Q2) in accordance with the dynamic fluctuations of traffic workload is defined as the CPP [118]. On the other hand, deciding how requests are distributed from all switches among controllers (Q3) so as to make the best use of the given controller resource is defined as the Controller Scheduling Problem (CSP).

[1] Due to the fact that the main and basic functionality of SDN control plane is to process the requests initiated from switches by running the routing algorithms [128], the control plane resources we consider in this thesis are measured by the control plane's capacity in terms of the number of requests it can process in each time unit.



(a) New switch (Switch 5) (b) New controller (Controller 2)

Figure 1.2: Examples of switch-controller binding.

1.2 Motivation

Solving the RM problem in distributed SDN controller architectures is challenging [128, 148, 295]. In general, existing studies can be classified into two main categories: (1) introducing new system architecture designs and (2) proposing new RM algorithms. Following the two categories, the challenging issues of the RM problem are discussed from both the architectural and algorithm design aspects respectively.

1.2.1 Architectural Challenges

Since RM is performed on top of a distributed controller architecture, a well-designed architecture that enables transparent, efficient, and fine-grained RM operations is critical. In SDN, establishing a network connection (e.g., a TCP/IP connection) between a switch and a controller is required before any communication can take place [14], e.g., the red dash-dotted lines in Figure 1.2.

Existing controller architectures feature a switch-controller *binding*. Specifically, each switch is bound to a controller, which restricts the requests generated by a switch to only be processed by its bound controller [14, 138]. As shown in Figure 1.2(b), requests from Switch 3 can only be sent to Controller 2. Thus, when performing Controller Scheduling (CS), instead of partially scheduling/distributing the requests from a

heavily loaded controller to an underloaded controller, all requests generated by the same switch must be forwarded to a new controller. In other words, CS is limited to a coarse switch level, which restricts the capability of properly distributing workload across all controllers and increases the risk of overloading some controllers if the newly bound switch accumulates lots of pending requests.

Moreover, since a switch is bound to a controller, whenever the Controller Placement (CP) is changed (regardless of the number of controllers or their locations), switch-controller bindings need to be re-established. For example, in Figure 1.2, 3 switches (i.e., Switch 3, 4, and 5) are rebound from their original controller (Controller 1 in Figure 1.2(a)) to the newly added controller (Controller 2 in Figure 1.2(b)). During the rebinding process, requests arriving at the switches cannot be processed by the controllers, which inevitably causes increased response time or even introduces network interruption.

In view of the above limitations, designing a new distributed controller architecture to enable effective and efficient RM becomes necessary. However, designing a distributed controller architecture for RM is challenging since it has to satisfy the following design principles:

- *Scalability*: The new architecture should not introduce any potential bottlenecks when it scales up. Also, it should support flexible controller number changes to meet various network demands.

- *Transparency*: Both CP and CS should be performed in a transparent manner without interruption of switch-controller re-binding.

- *Compatibility*: The new architecture should be compatible with existing SDN designs without violating the control-data plane separation. Also, the new architecture should not introduce any hardware or special software modification to existing switches.

- *Extensibility*: To enable continued development, the new architecture

should be designed to easily and flexibly integrate any new functionality. For example, new RM algorithms can be easily implemented and deployed on the new architecture using provided interfaces.

- *Separation of Concerns*: The architecture should be separated into different modules and each module is responsible for a dedicated function. For example, interactions with switches (e.g., connection establishment and statistics collection) should be handled in a module that is separated/independent from modules related to controller operations. With separation of concerns, the architecture can be easily maintained and quickly adapt to changes (e.g., new network protocols).

1.2.2 Algorithm Design Challenges

Even without the switch-controller binding constraint, managing the controller resources is still difficult due to the following reasons:

1.2.2.1 Challenges for the CPP

NP-hardness of the CPP: According to existing literature [50, 118, 291], solely finding the locations of a given number of controllers without considering their capacity constraint is a vertex k-center problem [122] or a k-median problem [81], subject to the optimization goal. Both k-center and k-median problems are already NP-hard. The CPP we considered in this thesis needs to identify not only the number of controllers with different capacities but also their locations, which is clearly also NP-hard since the k-center or k-median problem can be considered as a special case of our problem.

Problem Formulation: When deciding the CP, many studies [118, 143, 153, 165, 303] completely ignore the impact of the CSP. However, even with an appropriate CP, we may still run into the risk of poor network performance if the requests cannot be properly distributed among all controllers.

In such a situation, the controller processing time can dominate the total request response time [269, 291]. Although follow-up studies explicitly recognized the importance of the CSP when dealing with the CPP, they tend to oversimplify the CSP, e.g., by assuming each switch contributes an equal amount of workload to the control plane [123, 180]. Thus, how to explicitly and precisely quantify the impact of the CSP on the CPP needs to be further investigated.

Algorithm Design: Different algorithms were proposed to solve the CPP. Specifically, exact methods were adopted in [118, 123, 206, 258] which guarantee the optimality of the solution. However, their optimality guarantee comes at the price of high computational costs. As an alternative, heuristic methods have been widely used to address the CPP [66, 297] to balance the trade-off between computation time and solution quality [92, 139, 141, 151, 152, 165, 180, 235].

In literature [212, 213, 233], Evolutionary Computation (EC) is often exploited to find near-optimal solutions to NP-hard problems. The promising results reported previously inspired us to tackle the CPP through an EC method. However, as a population-based approach, EC requires a large number of performance evaluations of randomly generated CP candidates until a satisfying solution is obtained, which can be computationally expensive. Especially in a large network, the corresponding search space becomes too large for an EC method to handle effectively. Although network clustering has been widely used to reduce the search space in the CPP by dividing the network into independent clusters, existing research treated each clustered sub-network as being completely independent. When a cluster encounters traffic bursts that cannot be handled by existing controllers, the control plane will be significantly slowed down. Therefore, further research must be performed to tackle both the CPP and the CSP simultaneously in a coherent framework and to effectively handle the traffic bursts.

1.2.2.2 Challenges for the CSP

NP-hardness of the CSP: Generally speaking, the CSP can be solved at two granularity levels: switch level and request level. Let us consider a network with M switches and N controllers. R requests generated by M switches within a specific period need to be dispatched to N controllers. When CS is performed at the switch level (i.e., with the switch-controller binding constraint), solving the CSP means finding the switch-controller mapping which has a complexity of $\mathcal{O}(N^M)$. On the contrary, when performing CS at the request level, requests from one switch are no longer restricted to be handled by only one controller. Instead, they can be flexibly distributed and processed among multiple controllers. However, this flexibility comes at a cost of increasing the CSP complexity from $\mathcal{O}(N^M)$ to $\mathcal{O}(N^R)$ given $M \ll R$. Although solving the CSP at the switch level seems more feasible, it is still NP-hard according to existing studies [56, 278, 280, 281].

Algorithm Design: To address the CSP, different algorithms have been proposed, ranging from exact methods [206, 258] to heuristics [38, 56, 66, 74, 80, 278, 307, 308]. However, these algorithms were mostly designed for switch level CS, i.e., the CS decision is made for each switch. In comparison, when CS is performed at a per-request level, its complexity increases significantly, rendering the efficiency and effectiveness of existing algorithms questionable (e.g., dynamic programming in [206] and simulated annealing in [38]). Recently, Deep Reinforcement Learning (DRL) has demonstrated its potential on tackling challenging scheduling tasks [67, 135, 185, 194]. However, the scheduling policies trained by existing DRL approaches only target networks with a fixed number of controllers. In reality, the number of controllers can change in order to meet the varying traffic demand, which renders the trained policies inapplicable. In line with the fixed policy design, existing DRL algorithms were developed to train policies with a fixed number of controllers. Therefore, they cannot cope with the training of an adaptive policy.

Problem Formulation: When applying DRL, existing studies typically designed a policy to be used by a centralized agent with access to up-to-date global information. Such a centralized approach suffers from several limitations. First of all, this approach is prone to scalability issues [300] and a single point of failure. Secondly, the use of global information may not be practical since obtaining timely global information over the entire SDN network can cause substantial communication overhead and sometimes may not even be available. Thirdly, even though the previous two issues can be alleviated by deploying multiple agents in the network using the same policy, the trained policy cannot cope with inter-agent interference and outdated/local network information. This results in poor and unpredictable network performance [188].

1.3 Goals

The ultimate goal of this thesis is to effectively manage the controller resources in distributed SDN controller architectures by developing a comprehensive solution to address the challenges from both the architectural and algorithm design aspects. To achieve this overall goal, the following three research objectives have been established to guide this research.

(1) *Designing a new bindingless distributed controller architecture to enable flexible CP and CS.*

Given the limitations of switch-controller binding, this thesis will design a new controller architecture featuring bindingless switch-controller association. With the help of the new architecture, CP and CS should be performed flexibly and transparently without introducing any modification to existing SDN switches. In particular, the bindingless design aims to avoid switch-controller rebinding whenever the CP is changed. For the CSP, the new architecture should enable flexible and fine-grained CS, e.g., a per-request level. More-

over, it should also allow newly designed RM algorithms to be easily implemented and deployed.

(2) *Developing a new EC-based approach armed with Gradient Descent (GD) optimization for effective and scalable controller placement.*

Existing studies tackled the CPP without explicitly considering the impact of request distribution among controllers. Particularly, the request distribution among all controllers is usually oversimplified to be identical or even completely ignored in some extreme cases. The influence of request distribution on the CPP performance will be investigated in this thesis. Considering the NP-hardness of the CPP, a new EC-based algorithm will be proposed. In addition, to cope with the large search space, network clustering will be adopted to solve the CPP in a scalable manner. However, instead of isolating each cluster, cluster cooperation mechanisms will be investigated to handle traffic bursts.

(3) *Developing a new DRL-based method to automatically design effective, efficient and adaptive policies to guide request dispatching for all switches.*

It is typical to represent a policy as a Deep Neural Network (DNN) [240, 242]. However, this policy representation fails to function well in a network with changing numbers of controllers. To address this issue, a new adaptive policy representation will be designed for efficient request dispatching over an arbitrary number of controllers. In line with the new policy representation, new training algorithms will be proposed to enable effective and efficient training of an adaptive policy. Moreover, considering that the global network information may not be always available to the policy, the CSP will be investigated in a Multi-Agent Deep Reinforcement Learning (MA-DRL) manner where the dispatching decision is made with local information.

1.4 Major Contributions

This thesis makes the following major contributions:

(1) *The thesis develops a new BindingLess Architecture for distributed Controllers (BLAC), enabling flexible controller resource management.*

By introducing a scheduling layer into existing SDN architectures, CP and CS can be performed in a flexible and transparent way without imposing the switch-controller binding constraint. New scheduling algorithms can be designed and utilized by the scheduling layer to cope with the change of communication demand in the network, making our architecture more flexible and adaptive.

With all the benefits provided, our new architecture is compatible with existing SDN designs without introducing any hardware or special software modification to existing switches. To demonstrate its real-world applicability, a prototype of BLAC is implemented based on ONOS, a widely-used real-world open source SDN controller architecture. Experiments have been carried out to demonstrate its efficacy. The results show that our design outperforms the binding-based controller architectures in terms of both system throughput and response time.

Part of this contribution has been published in:

VICTORIA HUANG, QIANG FU, GANG CHEN, ELLIOTT WEN, AND JONATHAN HART. BLAC: A BindingLess Architecture for Distributed SDN Controllers. In *Proceedings of the 42nd IEEE Conference on Local Computer Networks (LCN 2017)*, October 09–12, 2017, Singapore. IEEE Press. Pages 146–154.

(2) *This thesis proposes a new algorithm called Clustering-based Genetic Algorithm with Cooperative Clusters (CGA-CC) to tackle the CPP.*

In particular, we present a new problem definition named the Controller Placement and Scheduling Problem (CPSP) that explicitly cap-

tures the strong inter-dependencies between the CPP and the CSP. The CPSP highlights the necessity and importance of simultaneously addressing both the CPP and the CSP within the same problem framework. A full solution to the CPSP is hence obtained as a combination of solutions to both the CPP and the CSP.

To address the CSP, a GD-based scheduling algorithm is developed to reduce the average response time of request processing to a reasonably low level. Driven by our GD-based scheduling algorithm, CGA-CC is proposed to utilize a general purpose clustering algorithm to split the network into non-overlapping sub-networks. Subsequently, GA is applied to address the CPP in each sub-network. Moreover, to enable close cooperation among sub-networks, we further introduce 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 and ensure good workload balance across all sub-networks.

Simulation studies conducted in this thesis show that GD-based scheduling algorithm can effectively reduce the response time while maintaining high control plane throughput. On the other hand, CGA-CC effectively improved the resource utilization of the control plane without sacrificing response time, in comparison to the widely-used K-center approach and the state-of-the-art algorithm.

Parts of this contribution have been published in:

VICTORIA HUANG, GANG CHEN, QIANG FU, AND ELLIOTT WEN. Optimizing Controller Placement for Software-Defined Networks. In *Proceedings of 2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, April 08–12, 2019, Washington DC, USA. IEEE Press. Pages 224–232.

VICTORIA HUANG, GANG CHEN, PENG ZHANG, HAO LI,

CHENGCHEN HU, TIAN PAN AND QIANG FU. A Scalable Approach to SDN Control Plane Management: High Utilization Comes with Low Latency. In *IEEE Transactions on Network and Service Management (TNSM)*, Volumn 17, Issue 2, June, 2020, Pages 682–695.

(3) *To effectively utilize the multi-controller resources in SDN, a new DRL-based approach called Multi-Agent Proximal Policy Optimization (MA-PPO), together with a new adaptive policy representation, is proposed.*

Specifically, we first propose a new DNN-based policy representation that can be applied to networks with changing numbers of controllers. To demonstrate the effectiveness of the new policy design, the policy design problem is formulated under the Single-Agent Deep Reinforcement Learning (SA-DRL) setting. Specifically, the policy is trained and used by a centralized agent with access to up-to-date global network information. In line with the new policy, a new Single-Agent (SA) training approach is developed armed with the new policy gradient calculation technique.

Considering that the SA-DRL formulation is prone to scalability issues as we discuss in Subsection 1.2.2.2, an MA-DRL approach is proposed which reformulates the problem as a Multi-Agent Markov Decision Process (MA-MDP) where each switch makes request dispatching decisions solely based on its local network information. With the new formulation, a new training algorithm called MA-PPO is proposed, enabling multiple agents to simultaneously learn their local policies.

Our simulation results show that the policies trained using our SA-DRL and MA-DRL approaches can achieve significantly better performance compared with man-made policies as well as policies learned via other RL algorithms and can adapt to networks with a changing number of controllers.

Parts of this contribution have been published in:

VICTORIA HUANG, GANG CHEN, AND QIANG FU. Effective Scheduling Function Design in SDN Through Deep Reinforcement Learning. In *Proceedings of 2019 IEEE International Conference on Communications (ICC)*, May 20–24, 2019, Shanghai, China. IEEE Press. Pages 1–7.

VICTORIA HUANG, GANG CHEN, QIANG FU. Multi-Agent Deep Reinforcement Learning for Request Dispatching in Multi-Controller Software-Defined Networking. In *IEEE Transactions on Parallel and Distributed Systems (TPDS)*. (Under review)

1.5 Organization of the Thesis

The remainder of the thesis is structured as follows. Chapter 2 provides the background knowledge and surveys related work. The main contributions of the thesis are presented in Chapters 3 to 5. Each chapter addresses one research objective. Chapter 6 concludes the thesis.

Chapter 2 provides basic concepts and related work that form the background and motivate our research. It starts with introducing basic concepts of SDN and optimization techniques. Guided by our research goal, related work on SDN controller architectures, CP methods, and CS algorithms are also reviewed in this chapter.

Chapter 3 tackles the RM problem from an architectural perspective. In particular, it proposes a new distributed controller architecture featuring switch-controller bindingless association by introducing a scheduling layer. A prototype is built and experiments are conducted to demonstrate its effectiveness.

With the flexibility provided by the new architecture proposed in Chapter 3, Chapter 4 addresses the RM problem, particularly the CPP, from an algorithm design perspective. A queuing model and a new problem formulation are introduced to explicitly quantify and strengthen the importance of solving both the CPP and the CSP coherently within the

same problem framework. New algorithms (e.g., CGA, CGA-CC) are introduced and extensive simulations are conducted to compare the proposed algorithms with the widely-used and state-of-the-art algorithms.

Given the controllers selected by CGA-CC which is developed in Chapter 4, Chapter 5 solves the CSP by proposing new DRL-based approaches to automatically learn policies. Specifically, a new policy representation is designed to adapt to a changing number of controllers. To enable the training of the adaptive policy, new training algorithms are investigated. An extensive comparison between the proposed approaches and a range of widely used CS methods is performed.

Chapter 6 summarizes the thesis and outlines our major contributions. Different potential research directions for future work are also highlighted and discussed in this chapter.

Chapter 2

Literature Review

This chapter provides basic concepts and related work that form the background and motivate our research. This chapter starts with introducing basic concepts and terminologies in SDN. Then we provide a brief introduction to different techniques for solving optimization problems, ranging from exact methods to machine learning algorithms. Details about reinforcement learning concepts and algorithms are also provided. Guided by the motivations in Section 1.2, SDN controller architectures, controller placement methods, and controller scheduling algorithms are three main approaches to solve the RM problem in distributed SDN. Literature reviews on these three approaches are presented respectively.

2.1 Software-Defined Networking

With the rapidly growing number of mobile devices (e.g., cell phones, tablets, and laptops) and the increasing popularity of cloud computing, computer networks are undoubtedly playing a key role in communication [225].

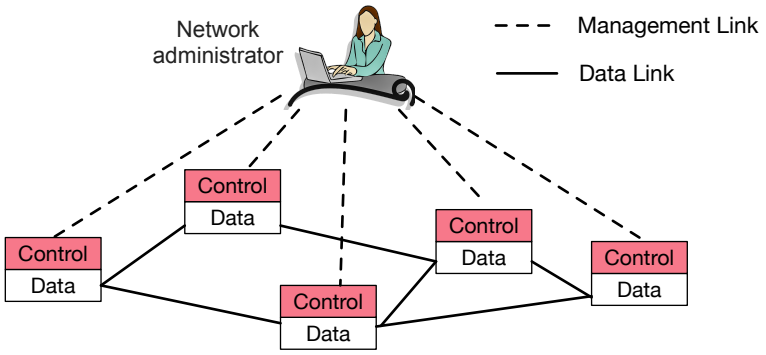


Figure 2.1: Traditional network.

2.1.1 Traditional Networks vs. SDN

In general, a computer network consists of two layers of functionality: the control plane and the data plane (also called the infrastructure layer). Specifically, the control plane is responsible for the configuration of network devices and making packet forwarding decisions. Guided by the decisions, the data plane, which corresponds to the physical network infrastructure (e.g., switches and links), performs network data forwarding at the hardware level [246].

In traditional networks, as shown in Figure 2.1, the control plane and data plane are tightly coupled and resided in the same network device. Although this integrated design is widely adopted, there are several limitations in traditional networks:

- Complicated and time-consuming network configuration:
  - *Distributed control plane:* In traditional networks, network configuration update is heavily human-centered with minor autonomous behavior [156]. In some extreme cases, network operators may need to manually and individually configure each device using a limited set of low-level device configuration commands in a command line interface environment, which is a time-consuming and error-prone process [155].

  - *An increasing number of network devices:* A network can be huge in size. Even a network hosted by medium-size organizations (e.g., a campus network) can consist of hundreds and up to thousands of devices [155]. Moreover, to accommodate the rapid growth of communications, more network devices will be added over time. The growing number of network devices will inevitably exacerbate the configuration complexity and potentially introduce higher chances of human errors [35].
  - *Heterogeneous network devices:* Computer networks are made of a large number of heterogeneous network devices (e.g., switches, routers, and middle-boxes). Managing and configuring these proprietary and dedicated hardware requires the use of a limited set of commands, vendor-specific languages, and configuration tools through specific interfaces and protocols. Obviously, the heterogeneity of network devices increases the configuration difficulty and requires a high level of domain expertise [35].

- Time-consuming deployment of new ideas: Even though new network technologies or new protocols are developed, it takes years from design to standardization before the deployment, making it difficult to implement innovative concepts [288].

In view of the aforementioned problems, a new network solution is required. Software-Defined Networking (SDN), an emerging networking paradigm, is currently attracting substantial attention from both academia and industry. SDN is notable for its separation from the data plane and the control plane. Specifically, SDN decouples the control plane from the network devices and forms an external entity called the SDN controller which is the "network brain". The controller is a software platform that manages network resources and provides high-level abstractions and APIs for network applications to manage, monitor, interact with, and program

network devices [45, 161]. With its decoupling nature, SDN is gaining increasing popularity due to the following benefits it provided:

- *Easy configuration:* With the centralized control plane, network configuration can be done in a centralized manner. Moreover, the network can be easily programmed using the high-level vendor-neutral abstractions and APIs provided by the controllers instead of using the low-level vendor-specific commands.
- *Fast Innovation:* Since the control plane is separated from the data plane in SDN, network administrators and researchers could implement their innovative solutions in a form of software applications without accessing low-level commands. Apart from that, new protocols and network services could be deployed easily without changing/upgrading the underlying hardware, speeding up network innovation and new network designs.
- *Low Cost:* SDN could reduce the cost of network devices since the switches in SDN only perform packet forwarding operations, which is relatively simple compared with dedicated hardware (e.g., Cisco ASA 5545-X and A10 Thunder 1040 [24]) in traditional networks with embedded control function [127, 161]. Besides, it also lowers the network operational and management expenses without depending heavily on highly skilled administrators who are adept at configuration and management of vendor-specific network devices [246].

2.1.2 SDN Framework

As shown in Figure 2.2, an SDN architecture comprises an infrastructure layer and a control layer. These two layers communicate with each other via predefined interfaces.

- *Infrastructure layer:* The infrastructure layer in SDN consists of a set of hardware-based (e.g., switches and routers) or software-based

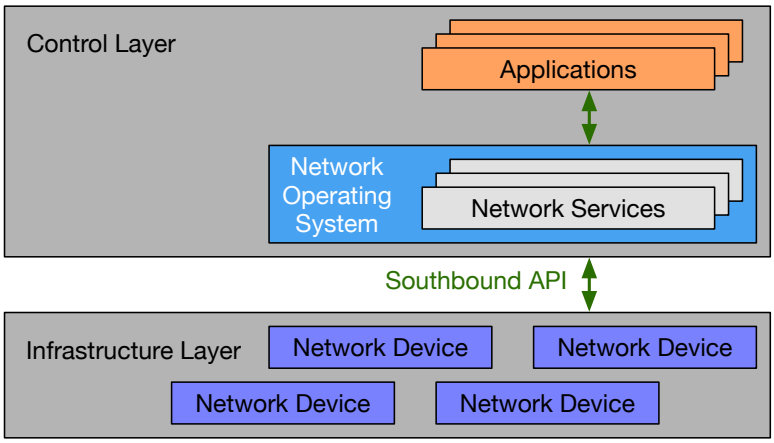


Figure 2.2: Three-layer framework of a SDN architecture, adapted from Figure 1 in [289].

(e.g., Open vSwitch [222]) devices interconnected via wired cables or wireless channels. Different from traditional networks, network devices in SDN only perform data forwarding based on a set of well-defined instructions (e.g., flow rules).

Specifically, an OpenFlow-based switch, as depicted in Figure 2.3, contains two major components: a secure OpenFlow channel and flow tables with multiple flow rules/entries stored in Ternary Content Addressable Memory (TCAM). As shown in Figure 2.3, each flow rule consists of three parts: (1) a match field used to match incoming packets based on information found in the packet header (e.g., ip address and MAC address); (2) a set of actions to be executed on matching packets (e.g., drop or forward to certain ports); and (3) counters for collecting network statistics.

When a new packet arrives, the switch compares the packet header fields with the matching fields in the flow table. If a matching rule is found, the packet will be handled with the action provided by the matching rule. Otherwise, the packet will be processed based on the *table-miss* flow rule, which may include dropping the packet, passing

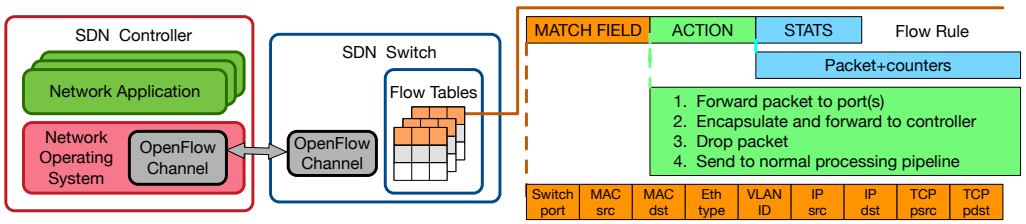


Figure 2.3: OpenFlow-enabled SDN devices, adapted from Figure 7 in [161].

it to a subsequent table or forwarding it to the controller over the OpenFlow channel.

- *Southbound interfaces:* Southbound interfaces are open interfaces used as the connection between the infrastructure layer and the control layer, which is a crucial part of separating the data forwarding and network control. Nowadays, different southbound interfaces are proposed, such as OVSDB [11] and OpenFlow [13].

  So far, OpenFlow is the most widely deployed open southbound communication protocol for SDN [161]. OpenFlow offers three information sources sent from switches to controllers: (1) event-based messages to notify controllers with network state changes (e.g., a link or port changes its status) (2) *packet-in* messages when packets do not match the flow rules. (3) network statistics collected by the switches. These information sources are crucial to provide flow-level information to the controllers.

- *Control layer:* The control plane in SDN is commonly referred to as the controllers, which provides abstractions and management of the underlying devices (i.e., switches) through well-defined southbound interfaces (e.g., OpenFlow). Various design and architectural choices for the control plane have been proposed, which will be discussed in Section 2.3. The control plane provides an ideal platform to run network applications. These applications implement the con-

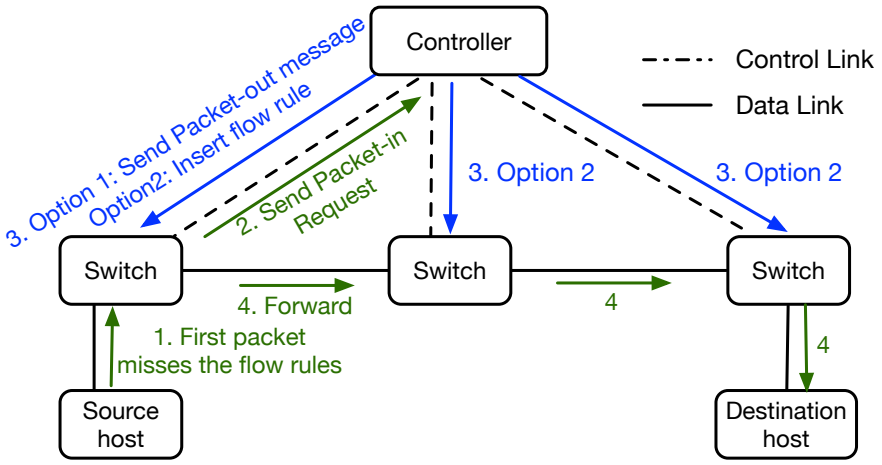


Figure 2.4: Reactive mode.

trol logic to control the behaviors of the network devices. In particular, these applications take the input (e.g., network topology and network states) provided by the controllers to perform various functionalities, such as traffic engineering, security enhancement, measurement, and monitoring.

The main function of an SDN controller is to process *Packet-in* requests generated from switches and setup flow rules for switches [128, 295] either reactively or proactively [91]. In the reactive mode, as shown in Figure 2.4, a *Packet-in* request is sent from the switch to a controller whenever a table miss occurs. Upon receiving the request, the controller calculates a forwarding path for the new packet and sends either a *Packet-out* message to the switch or flow rules to be installed in related switches. The timespan from when a *Packet-in* request is sent by the switch to when the corresponding *Packet-out* response is received by the switch is defined as the *response time*. In comparison, controllers in the proactive mode install flow rules beforehand, which reduces the chance of new packets triggering a table miss. Therefore, packets can be processed immedi-

ately using the matched rules without waiting for the response from the controller, completely avoiding the switch-controller communication and flow setup time. Similar to existing works [47, 48], this thesis focuses on reactive mode due to several reasons. First, reactive mode provides an upper bound for the amount of controller workload compared to the fully proactive mode and the hybrid proactive and reactive mode. Thus, it helps the network administrator to understand the maximum capacity of the control plane [48]. Second, compared to the proactive mode, the reactive mode allows controllers to react to unexpected network events in a timely manner [91]. Third, the reactive mode provides a more efficient way of using the TCAM memory.

With these decoupling layers and open interfaces, network devices are highly efficient in packet forwarding and can be dynamically programmed via well-defined southbound interfaces; a global network view can be abstracted and maintained by the control plane; the network logic can be easily implemented by the applications and deployed by the control plane.

2.1.3 Challenges in SDN

Given the promises of improved performance and simplified management, SDN is still in its infancy. Many challenging issues about SDN have not been well addressed:

- *Security:* Cyber-attacks are gaining increasing global concerns due to their extensive damage. As an emerging network paradigm, SDN can be utilized to enhance network security by simultaneously exploiting its programmability and the centralized network view [244]. Nevertheless, these advantages offered by SDN can also result in se-

curity vulnerabilities. For example, DoS[2] attacks can be launched by flooding the switches with large amounts of spoofed flow arrivals with non-repetitive random header patterns. This may eventually overwhelm the control plane with the generated flow requests [62, 89]. One solution is to improve the control plane's scalability to avoid resource exhaustion [62].

- *Global network view maintenance:* Distributed controller architectures enhance the scalability and reliability of the control plane by deploying multiple controllers in the control plane. To maintain a consistent global network view, the physically distributed controllers need to exchange their network information and update their individual network view based on the exchanged information. As the network scales up, more controllers will be deployed with more information exchanges, which increases the synchronization overhead. Thus, it is important to determine a suitable number of controllers to be deployed in the network so as to balance the trade-off between synchronization overhead and the network performance improvement (e.g., increase the throughput and reduce the response time).

- *Control plane scalability:* Though there exist numbers of studies about the scalability of control plane by enhancing the capacity of one controller or adding more controllers into the control plane, the scalability problem has not been well addressed. For example, even with enough control plane resources, long request response time can still occur due to (1) long controller processing time (e.g., controllers are overloaded due to the unbalanced workload distribution) and (2) long propagation latency (e.g., controllers are deployed in remote locations because of the inappropriate controller placement).

[2]A denial-of-service (DoS) attack is a cyber-attack in which attackers prevent legitimate users from using a network service or resources by overloading the systems with superfluous requests.

In this thesis, we will focus on the scalability issue of the SDN control plane. In the SDN context, scalability is widely understood as the control plane's ability to handle more requests with the increased number of network nodes and geographic coverage while simultaneously satisfying network performance requirements (e.g., network response time) [128, 148]. To improve the control plane scalability, it is critical to effectively and efficiently manage the control plane resources. That is, the available controller resources should be effectively utilized to optimize the network performance and controllers should be flexibly added or removed in a timely manner.

Note that the control plane resources are mainly consumed by four components: (1) *Local view construction* by collecting network information from switches; (2) *Global view maintenance* through communicating with other controllers; (3) *Packet-in requests processing*; (4) *Flow rule installation*. According to existing studies [128, 291, 295], processing *Packet-in* requests contributes the most significant part of a controller's workload. Therefore, this thesis focuses on the control plane's ability in requests processing.

2.2 Optimization Techniques

In the perspective of SDN distributed controller architectures, RM is the process of allocating control plane resources to the network as well as scheduling existing shared resources so as to optimize the network performance (e.g., response time and energy efficiency) under a number of constraints. Therefore, RM can be naturally formulated as an optimization problem and potentially solved by optimization techniques. In this section, a wide range of optimization techniques which have been used to solved RM problems are reviewed.

2.2.1 Exact Methods

Exact methods guarantee the optimality of the solutions. Two classical exact methods are dynamic programming [42, 43] and branch and bound algorithms [261].

Dynamic programming was first proposed to solve multistage decision processes [42, 43]. In particular, dynamic programming recursively divides the problem into a number of sequential stages. Then it starts with making a decision for the last stage and to work backwards to the earlier ones until it reaches the end. The sequence of decisions made at each stage forms the optimal solution for the underlying problem. Dynamic programming can also be applied to solve other problems (e.g., Integer Linear Problems (ILP)) if they can be formulated as a multistage process [229]. Dynamic programming is an exhaustive search method which avoids enumerating all possible solutions in the search space by pruning partial decision sequences that cannot lead to the optimal solution [229, 261].

Branch and bound algorithm is another popular enumerative approach [229, 261]. It explores the search space by dynamically building a tree with the full search space at its root node. The algorithm recursively splits/branches the search space into smaller subspaces/nodes. In each iteration, if candidate solutions in a node cannot improve the current best performance bound (e.g., lower bound for minimization and upper bound for maximization), the node will be completely discarded (i.e., it will not be searched again), which is called pruning. Otherwise, the current performance bound is updated and the associated node will be split. The algorithm terminates when there are no more nodes to branch or all nodes are eliminated.

An example is provided to illustrate the idea of using branch and bound algorithm to solve a two-dimensional maximization ILP $f(x_1, x_2)$. First of all, an upper bound z_0^R for the optimal solution can be obtained by solving a relaxed ILP (e.g., dropping the integer constraint). At the same time, the lower bound z_{ip} of the original problem is set to $-\infty$. Then we

create two mutually exclusive subproblems (e.g., f_1 and f_2) by introducing additional constraints on one of the two variables (e.g., x_1), which splits up the original search space. f_1 will be removed if it cannot be solved or its upper bound z_1^R is smaller than z_0^R. If the optimal solution of f_1 are integral and $z_1^R > z_{ip}$, z_{ip} will be replaced with z_1^R and f_1 will be removed. Otherwise, another constraint will be introduced to create two subproblems $f_{1,1}$ and $f_{1,2}$ under f_1. The algorithm continues until all nodes are removed and the current z_{ip} is the optimal solution.

Apart from dynamic program and branch and bound, there are other exact methods, e.g., A* search [229], branch and cut [32], constraint programming [29]. Despite the optimality guarantee, the main drawback of these methods is their effort to solve NP-hard problems grows exponentially with the problem size [229]. Therefore, all exact methods currently applied to NP-hard problems are limited to small-scale problems [229].

2.2.2 Model-driven Methods

Apart from exact methods, there are model-driven methods which have been widely used especially in operations research. In general, model-driven methods solve an optimization problem by constructing mathematical models to describe and analyze the underlying system, which generally provide optimal solutions or at least performance bounds. Model-driven methods can be classified based on the mathematical models that they use. Existing widely used model-driven methods are mostly based on queuing theory, Lyapunov optimization, and game theory.

Queuing theory was first proposed in [85], which is applied to model and analyze the behavior of processes that involve waiting lines/queues. To apply queuing theory, modeling the underlying system as a queuing model is critical. A queuing model is generally characterized by four basic elements: the arrival process of customers (e.g., inter-arrival times of customers), service mechanism (e.g., the number of servers, the duration,

and mode of service), queue discipline (e.g., first come first served or last come first served), and system capacity (i.e., the number of customers that can wait at a time in the queues). Once we manage to formulate the system as a queuing model, queuing theory can be applied to determine the system utilization and the average waiting time of each customer.

Another example is *Lyapunov optimization* which applies Lyapunov function[3] to queuing networks (e.g., wireless networks) with the aim to stabilize all network queues while simultaneously optimizing the time average of certain performance objectives (e.g., network throughput and energy cost) [195, 205, 211]. In Lyapunov optimization, Lyapunov drift[4] measures the changes of the queue states. Given a defined penalty function related to the performance objective, Lyapunov optimization can be used to calculate a performance bound on the sum of Lyapunov drift and the weighted penalty function. Typically, the weight parameter is used to balance the trade-off between the average queue size (i.e., the system stability) and the penalty function (i.e., performance objective).

Game theory has been widely used to study the interaction among decision makers/players. In general, game theory consists of two branches: cooperative game theory and non-cooperative game theory. Cooperative game theory analyzes what groups/coalitions agents form, the joint actions taken as coalitions and their corresponding payoffs. On the other hand, non-cooperative game theory focuses on predicting each player's individual strategy and payoff and find Nash equilibrium. Game theory has been successfully applied to various fields (e.g., supply chain management [169], transportation [25, 41], and computer networks [58]).

[3]Lyapunov function has been extensively used in control theory to prove system stability. In a queuing system, Lyapunov function is a non-negative function which defines a scalar measure of the state of all queues at a time slot. A typical Lyapunov function is defined as the sum of the squares of all queue sizes at a time slot.

[4]Lyapunov drift is defined as the difference between the Lyapunov function at two consecutive time slots.

2.2.3 Heuristic Optimization Methods

Due to the fact that computing optimal solutions is intractable for many optimization problems, heuristic optimization methods have been developed to search for acceptable solutions in a reasonable time. In general, heuristic optimization methods can be classified into *heuristics*, *approximation algorithms*, *meta-heuristics*, and *hype-heuristics*.

Heuristics are regarded as experience-based techniques which are designed to solve a specific problem [261]. In general, there are two types of heuristics: construction heuristics and improvement heuristics [49, 229]. Construction heuristics (e.g., greedy heuristic) generate solutions from scratch by iteratively adding components to an initially empty partial solution until a complete solution is obtained. In comparison, improvement heuristics (e.g., hill climbing), also called local search, start with a complete solution and iteratively try to improve the current solution by searching through its neighborhood. In terms of performance, construction heuristics are typically fast but the constructed solutions are usually inferior to the ones obtained by improvement heuristics. On the other hand, improvement heuristics can easily get trapped in local optima [229] and its performance highly depends on the initial solution.

Compared to heuristics, *approximation algorithms* aim to provide solutions with guaranteed quality within provable running time [229]. Thus, approximation algorithms are considered as an attempt to formalize heuristics by deriving a guarantee on the bound of the obtained solution from the global optimum [121]. For example, ϵ-Approximation algorithms guarantee that the maximal error between the generated solution and the optimum is ϵ [276]. Note that approximation algorithms are problem dependent which limits their applicability [261].

The term "*meta-heuristics*" was introduced by [103] and meta-heuristics define a high-level general methodology to guide the design of the underlying heuristics [49]. Different from heuristics and approximation algorithms, meta-heuristics usually make no or very few assumptions about

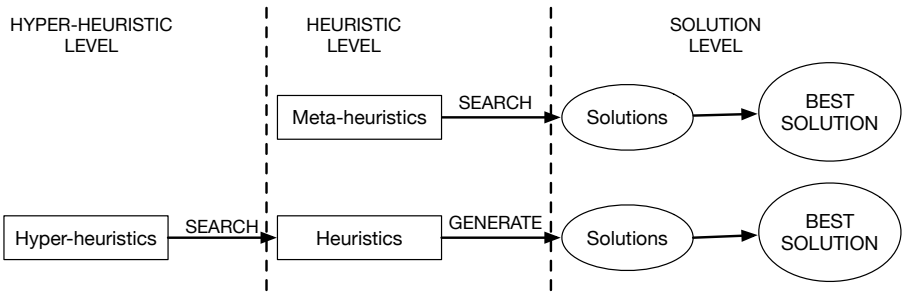


Figure 2.5: Meta-heuristics versus hyper-heuristics, adapted from Figure 3.3 in [40].

the problems being solved [229]. Due to their flexibility and intuitiveness, they have been applied to solve a wide range of complex real-life optimization problems in different domains, e.g., logistics [119], networking [137], transportation [34], and finance [198]. Meta-heuristics can be classified into single-point and population-based search based on the number of solutions searched at the same time. Single-point methods include local search-based meta-heuristics (e.g., tabu search [104] and simulated annealing [274]), which focus on modifying and improving a single solution. On the contrary, population-based methods (e.g., genetic algorithms [124] and ant colony [82]) maintain and improve a population of candidate solutions in every iteration of the algorithm. More details of specific meta-heuristics, particularly evolutionary computation, are provided in Subsection 2.2.4.

Recently, hyper-heuristics have emerged as a powerful approach for selecting or generating (combining, adapting) heuristics (or components of heuristics). Different from heuristics and meta-heuristics that directly search within the problem solution space, hyper-heuristics explore a heuristic search space to discover effective and adaptive heuristics, as shown in Figure 2.5 [40]. Note that different heuristics have different strengths and weaknesses. Therefore, a fundamental idea of hyper-heuristics is to use a set of known heuristics to either (1) find the com-

bination of existing heuristics or (2) generate new heuristics by combining basic components of existing heuristics to solve the problem. Such hyper-heuristics have been successfully applied to production scheduling [287], vehicle routing [251], and educational timetabling [253]. Most of the hyper-heuristic approaches involve a learning mechanism or meta-heuristics to assist the heuristic selection. Several learning methods such as Reinforcement Learning (RL) and Learning Classifier Systems (LCSs) [125] have been studied. Besides, evolutionary computation techniques such as genetic algorithms [124] and genetic programming [299] have been applied as meta-heuristics to direct the search process.

2.2.4 Evolutionary Computation

As a subfield of artificial intelligence and soft computing, EC is a family of global optimization algorithms inspired by nature. Under the EC framework, algorithms can be classified into two main streams: Evolutionary Algorithm (EA) and Swarm Intelligence (SI).

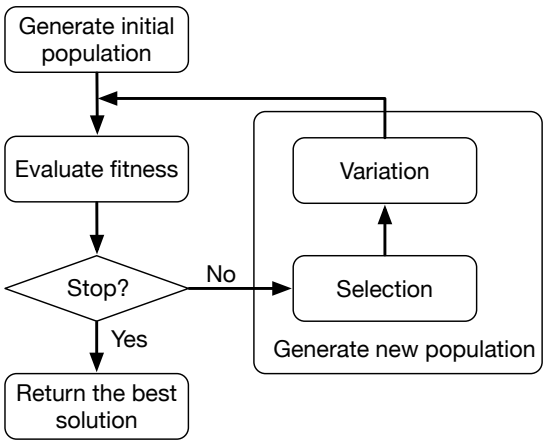


Figure 2.6: The principal diagram of evolutionary algorithms, adapted from Figure 1 in [83].

2.2.4.1 Evolutionary Algorithm

EAs are optimization methods derived from Darwin's theory of evolution by natural selection. They simulate the process of natural selection to find highly fit solutions to any given problems, as shown in Figure 2.6. Specifically, an initial population of potential solutions is randomly generated, which are often encoded in structures called *chromosomes*. During each iteration, called a *generation*, each individual of the population is evaluated by a predefined fitness function. After evaluation, the best performing individuals are selected as the basis for the next new generation that is produced via genetic operators such as *mutation* and *crossover*. This whole process repeats over many iterations until a sufficiently fit solution has been found [284]. The following are some popular EAs.

Genetic algorithms (GAs) are one of the earliest and most popular EAs which follow the EA diagram (i.e., Figure 2.6). In GA, each chromosome is usually represented as a fixed-length array of bits, integers, or real numbers and each chromosome is an encoded potential solution for the problem being solved. Despite its simplicity, GA has been successfully applied to solving many NP-hard problems [98, 111, 119, 198].

Genetic Programming (GP) is a variant of GA for symbolic regression where chromosomes are computer programs traditionally represented as tree structures with variable lengths. Due to its flexibility, GP is particularly suitable for problems in which the optimal underlying form of the solution is unknown [299].

Evolutionary Strategies (ES) was originally designed for numerical optimization [243]. An ES individual not only contains the solution of an optimization problem, but also the strategy parameters, especially in self-adaptive ESs [46]. The strategy parameters are used to control the statistical properties of the genetic operators (e.g., the mutation operator). In self-adaptive ES, the strategy parameters are evolved along with the solution during the evolution process.

2.2.4.2 Swarm Intelligence

SI algorithms are inspired by the collective behavior of natural and artificial systems where individual coordination is performed in a decentralized and self-organized manner. Typically, an SI system is composed of a population of simple and relatively homogeneous individuals that locally interact with each other and their environment following simple rules. Although no centralized/global coordinators are presented to guide the population's behaviors, the interaction enables the system to act in a coordinated way, e.g., converge to an optimum [149]. Two widely used SI algorithms are Particle Swarm Optimization (PSO) [150] and Ant Colony Optimization (ACO) [82].

2.2.5 Machine Learning

There has been a surge of interest in Machine Learning (ML) in recent years from both academia and industries. An ML algorithm generally includes a learning phase when the algorithm builds models (e.g., DNNs or decision trees) from the training data. In many ML algorithms, they also include a decision making phase when the well-trained model is used to make predictions on new data. Existing ML algorithms are widely classified into five categories: supervised, unsupervised, semi-supervised, reinforcement learning, and transfer learning.

2.2.5.1 Supervised Learning

Given a labeled training dataset (i.e., inputs and target outputs), a supervised learning algorithm is required to find the best mapping function from inputs to desired outputs so that the function can be used later to correctly predict the output for new/unseen inputs [160]. Supervised learning algorithms can be further divided into regression and classification. In particular, regression maps the inputs to a continuous output which is generally a quantity prediction (e.g., price or size). Well-known regression

algorithms include linear regression and logistic regression. On the contrary, classification generates discrete outputs called categories or labels (e.g., cat or dog). Some of the well-known classification algorithms are decision tree [234], support vector machines [275], and k-Nearest Neighbor [72].

2.2.5.2 Unsupervised Learning

Different from supervised learning, an unsupervised learning algorithm is given a set of unlabeled training dataset (i.e., inputs without target outputs). The algorithm aims to extract the patterns, structures among the training data by grouping training data with shared attributes. A central application of unsupervised learning is clustering and popular clustering algorithms include hierarchical clustering [145], k-Means clustering [147], and DBSCAN [86].

2.2.5.3 Semi-supervised Learning

Semi-supervised learning takes advantage of both supervised learning and unsupervised learning by learning from both labeled and unlabeled training data [57]. Semi-supervised learning can be classified as either inductive or transductive. Given both labeled and unlabeled training data, inductive learning learns a function that can predict the outputs for future unseen inputs (similar to supervised learning). On the other hand, transductive learning trains a good predictor on the unlabeled training data (similar to unsupervised learning). Typical semi-supervised learning algorithms are transductive support vector machine [144], expectation maximization [309], and pseudo labeling [168].

2.2.5.4 Reinforcement Learning

Instead of learning from a given training dataset, RL learns a mapping function during the interaction between an agent and the environment in

a trial and error process. In particular, an RL agent interacts with the unknown environment by performing a sequence of actions generated by the mapping function based on its observations. Depending on the actions, the environment generates a sequence of rewards to the agent. The goal of the agent is to learn the mapping function that can maximize the long-term cumulated reward. A detailed discussion is provided in Subsection 2.2.6.

2.2.5.5 Transfer Learning

One common assumption in many ML algorithms is that both training and testing data are sampled from the same distribution and same feature space [218]. When this assumption does not hold, most of the learned models exhibit unsatisfactory performance and need to be retrained from scratch with the newly collected training data. However, both data collection and model retraining can be expensive in many real-world applications. To address this issue, Transfer Learning (TL) was introduced which aims to extract the knowledge learned from source tasks in the source domain and apply it to improve the learning of the target task in the target domain [218]. Depending on different relationships between the source and target tasks and domains, existing TL can be widely classified into three categories: inductive TL, transductive TL, and unsupervised TL. In inductive TL, the target task is different from the source task regardless of the similarity between the target and source domains. Also, some labeled data in the target domain are provided. On the other hand, transductive TL tackles the same source and target tasks but in different domains. In unsupervised TL, the target task is different but related to the source task and no labeled data is available. Currently, TL has received increasing interests and has been widely applied to many domains, e.g., computer vision [99] and natural language processing [230].

2.2.6 Reinforcement Learning

RL is defined as a sequential decision making process where decisions/actions are made by an agent which is also a learner during its iterative interactions with an unknown environment [28]. Such a sequential decision making problem can be formulated as a Markov Decision Process (MDP).

2.2.6.1 MDP Formulation

Depending on the number of agents in the environment, there are Single-Agent Markov Decision Process (SA-MDP) where only one agent is involved during the training and MA-MDP with more than one agent.

An *SA-MDP* is usually described by a 6-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \mathcal{Z}, \mathcal{O} \rangle$. At each time step t, the overall environment is captured by a global state $s_t \in \mathcal{S}$. While interacting with the unknown environment, the agent receives a local observation $z_t \in \mathcal{Z}$. The relationship between z_t and s_t is determined by the observation function $z_t = \mathcal{O}(s_t)$. Based on its local observation, the agent issues an action $a_t \in \mathcal{A}$ chosen from its policy π which maps $\mathcal{Z}$ to a probability distribution over $\mathcal{A}$. After performing a_t, the agent receives a reward r_t given by the reward function $\mathcal{R}(s_t, a_t)$ and the environment enters the next state s_{t+1} decided by the state transition probabilities $\mathcal{P}(s_{t+1}|s_t, a_t)$. The agent's goal is to learn a policy π so as to maximize the *expected long-term cumulative rewards*:

$$J(\pi) = \mathbb{E}_{s_t, a_t \sim \pi} \left\{ \sum_{t=0}^{T} \gamma^t r_t(s_t, a_t) \right\} \tag{2.1}$$

where $\gamma \in [0, 1]$ is a discount factor that balances the trade-off between immediate and future rewards.

Similarly, an *MA-MDP* with N learning agents $\{A_i\}_{i=1}^N$ can be described by $\langle \mathcal{S}, \{\mathcal{A}^i\}_{i=1}^N, \mathcal{P}, \{r^i\}_{i=1}^N, \{\mathcal{Z}^i\}_{i=1}^N, \{\mathcal{O}^i\}_{i=1}^N \rangle$. At each time step t, every agent A_i receives its local observation $z_t^i = \mathcal{O}^i(s_t) \in \mathcal{Z}^i$ and the

multi-agent action $a_t = \{a_t^i\}_{i=1}^N$ is jointly formed from each agent's action $a_t^i \in \mathcal{A}^i$ chosen from its respective policy π^i. Apart from that, each agent also receives its own reward $r_t^i = \mathcal{R}^i(s_t, a_t)$. In a *fully cooperative* setting, all agents share the same goal which is to learn the policies $\{\pi^i\}_{i=1}^N$ so as to maximize (2.1) where $\pi_{\boldsymbol{\theta}} = \{\pi_{\boldsymbol{\theta}_i}\}_{i=1}^N$ and $r_t = \sum_{i=1}^N r_t^i$.

Based on whether the global state s_t is accessible to all agents, an MDP can be either *fully observable* or *partially observable*. For a fully observable MA-MDP, $z_t^i = s_t$ for all $i = 1, ..., N$ while in a partially observable environment, $z_t^i \neq s_t$ for any $i = 1, ..., N$. The same definition also holds for SA-MDP.

It should also be noted that to model a problem as an MDP, one essential requirement is that the environment needs to satisfy the *Markov property* [259]. In other words, the environment state s_t at time step t should compactly summarize the past without degrading the ability to predict the future [259]. Mathematically speaking, the probability of the environment giving the response $\{s_{t+1}, r_{t+1}\}$ at time step $t+1$ conditioned on both the past and present responses depends only on the present response

$$\begin{aligned} &P(s_{t+1} = s', r_{t+1} = r | s_t = s, a_t = a) \\ &= P(s_{t+1} = s', r_{t+1} = r | s_0, a_0, r_1, ..., s_{t-1}, a_{t-1}, r_t, s_t, a_t), \qquad \forall r, s', s, a \end{aligned} \tag{2.2}$$

where $s_0, a_0, r_1, ..., s_{t-1}, a_{t-1}, r_t, s_t, a_t$ are observed during the past $t+1$ time steps.

2.2.6.2 Basic Concepts

To provide a better understanding of RL, basic concepts in RL are introduced which includes value function and policy.

Value function: A value function is a function that maps any specific state s or state-action pair (s, a) to the expected long-term cumulative rewards when a particular policy π is followed. In general, there are *state*

value function $V^{\pi}(s)$ (i.e., V-function), *action value function* $Q^{\pi}(s,a)$ (i.e., Q-function), and *advantage function* $A^{\pi}(s,a)$ (i.e., A-function).

V-function measures the expected long-term cumulative rewards that can be received if agents start from any state s and follow policy π, which can be defined as below,

$$V^{\pi}(s) = \mathbb{E}_{s_t,a_t\sim\pi}\left\{\sum_{t=0}^{T}\gamma^t r_t|s_0 = s\right\} \tag{2.3}$$

Similarly, Q-function measures the expected long-term cumulative rewards when agents initiate from any state s and take an action a:

$$Q^{\pi}(s,a) = \mathbb{E}_{s_t,a_t\sim\pi}\left\{\sum_{t=0}^{T}\gamma^t r_t|s_0 = s, a_0 = a\right\} \tag{2.4}$$

The relation between V^{π} and $Q^{\pi}(s,a)$ is

$$V^{\pi}(s) = \int_{a\sim\pi}\pi(a|s)Q^{\pi}(s,a) \tag{2.5}$$

Given V^{π} and $Q^{\pi}(s,a)$, A-function measures how good an action a is compared to the average action for a specific state s:

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s) \tag{2.6}$$

Policy: Two types of policies are considered in RL, i.e., deterministic and stochastic. A deterministic policy maps the observations to actions, which is represented as

$$\pi : \mathcal{S} \rightarrow \mathcal{A} \tag{2.7}$$

In comparison, a stochastic policy outputs the distribution over all possible actions for a given state:

$$\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0,1] \tag{2.8}$$

where $\pi(a|s) \leq 0$ and $\int_{a\sim\pi}\pi(a|s) = 1$.

A policy can be represented in different ways, e.g., dynamic movement primitives or parametric functions [78]. Owing to the expressiveness and

trainability of neural networks, we mainly focus on policies approximated by neural network based parametric functions in this thesis. Specifically, a parametric policy is represented as $\pi_{\boldsymbol{\theta}}$ where $\boldsymbol{\theta}$ denotes the policy parameters.

2.2.6.3 Reinforcement Learning Algorithms

RL has been successfully applied to solving challenging problems, ranging from game playing [202, 250] to robotics [171]. The success achieved by RL has led to a significant increase in the number of RL algorithms. Existing RL algorithms can be generally classified into Value function Indirect Search (VIS) and Policy Direct Search (PDS).

Value Function Indirect Search: VIS learns the optimal value function which is used to extract the optimal policy by greedily selecting the action that maximizes the long-term rewards:

$$\pi(s_t) = \arg\max_{a \in \mathcal{A}} Q(s_t, a) \tag{2.9}$$

One of the most well-known VIS is *Q-learning* [283] which learns Q-function using the following update rule at any time step t:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a) - Q(s_t, a_t)]$$

where α is a hyper-parameter for learning rate.

In problems with small state and action spaces, value function can be easily represented as tables, which guarantees convergence to the optimal policy for finite MDPs [283]. Obviously, the tabular value function methods cannot scale in high-dimensional state and action spaces [78]. To address this issue, value function approximation approaches have been proposed which approximates the value function using a linear or nonlinear function. Existing popular VIS algorithms include Deep Q Network (DQN) [202] and Double Deep Q Network (DDQN) [273].

To tackle the high-dimensional state space problem in traditional Q-learning, *DQN* learns the Q-function approximated by a neural network

with parameters $\boldsymbol{\theta}$ via iteratively minimizing a loss function. Specifically, the loss function $\mathcal{L}_i(\boldsymbol{\theta}_i)$ at the ith iteration is defined as:

$$\mathcal{L}_i(\boldsymbol{\theta}_i) = \mathbb{E}_{s_t, a_t \sim \rho(\cdot)} \left[(y_i - Q(s_t, a_t; \boldsymbol{\theta}_i))^2 \right]$$

where $y_i = r_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a; \boldsymbol{\theta}_{i-1})$ and $\rho(s_t, a_t)$ is a probability distribution over s_t and a_t. To alleviate the issues of strongly correlated data[5] and non-stationary data distributions[6], DQN introduces an experience replay mechanism where data are randomly sampled from the experience repository to smooth the training distribution as well as breaking the strong correlations [202].

The maximization of the action space, i.e., (2.9), in both traditional Q-learning and DQN can easily lead to the overestimation of Q values [266]. To cope with this problem, *DDQN* [273] is proposed with the use of double Q-learning estimators [114]. In particular, a target Q network is used to estimate Q values. On the other hand, an online Q network is used for action selection and its parameters are updated through temporal difference.

Although literature has demonstrated the effectiveness of VIS [114, 202, 250, 283], there are several open research questions, e.g., many VIS algorithms suffer from the unstable learning process [208, 259]. The instability of the learning process in value function approximation approaches can result in value function divergence [33, 63, 114]. Although many techniques have been proposed to address the stability issue (e.g., experience replay [202] and dueling networks [282]), the learning stability still cannot be guaranteed. The policy learned by VIS is only implicitly represented and recreating the policy to achieve the learned long-term rewards can be challenging [221]. For example, when the action space is large, searching

[5]A common assumption in most deep learning algorithms is that the sampled data are independent. However, the sequence of observed data (e.g., states and actions) are typically strongly correlated.

[6]Deep learning algorithms generally assume the data are sampled from a fixed distribution. However, the data distribution in RL changes when the algorithm learns new behaviors.

for the action with maximum value function requires the time-consuming enumeration over all possible actions. The situation becomes even worse with a continuous action space.

Policy Direct Search: To address the limitations of VIS as discussed above, PDS is proposed which directly learns the optimal policy by searching the policy space. Existing PDS algorithms can be classified into model-based and model-free depending on whether an environmental model (i.e., the state transition probabilities $\mathcal{P}$ and the reward function $\mathcal{R}$ in Subsection 2.2.6.1) is learned. In model-based PDS, since the optimal policy is directly derived from the learned environment model, an inaccurate model can easily lead to catastrophic failure [262]. Moreover, learning an environmental model can be impractical due to the environment uncertainties and high computational complexity.

On the other hand, model-free PDS optimizes the policy $\pi_{\boldsymbol{\theta}}$ directly using the samples obtained during the interaction with the environment without relying on an environmental model. In model-free PDS, a widely-used framework is Policy Gradient Search (PGS) which updates the policy parameters $\boldsymbol{\theta}$ to maximize the expected long-term cumulative rewards $J(\boldsymbol{\theta})$ defined in (2.1) following the direction of $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$.

According to the policy gradient theorem [260], policy gradient can be calculated as:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{s_t, a_t \sim \pi_{\boldsymbol{\theta}}}[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t|s_t) Q^{\pi}(s_t, a_t)] \tag{2.10}$$

Based on (2.10), different PGS algorithms are proposed which differ in the estimation of Q^{π}. One representative PGS algorithm is REINFORCE [285] in which Q^{π} is estimated using sampled rewards $R^t = \sum_{i=t}^{T} \gamma^{i-t} r_i$. Other algorithms learns an approximation of the true Q-function, value function, or advantage function. These approaches belong to actor-critic algorithms where the policy π is the actor. Depending on different algorithms, the critic can be the Q-function, value function, or advantage function, which is used for policy evaluation. In the following, we will introduce

two representative actor-critic algorithms Trust Region Policy Optimization (TRPO) [240] and Proximal Policy Optimisation (PPO) [242].

TRPO is designed to improve the policy by iteratively optimizing a performance lower bound. Specifically, the expected long-term cumulative rewards of a new policy $\tilde{\pi}$ can be calculated as the cumulated advantages (2.6) over an old policy π with respect to actions sampled from $\tilde{\pi}$ (i.e., $a_t \sim \tilde{\pi}(\cdot|s_t)$):

$$\begin{aligned} J(\tilde{\pi}) &= J(\pi) + \mathbb{E}_{s_t, a_t \sim \tilde{\pi}} \left\{ \sum_{t=0}^{\infty} \gamma^t A^{\pi}(s_t, a_t) \right\} \\ &= J(\pi) + \sum_{t=0}^{\infty} \sum_{s} P(s_t = s|\tilde{\pi}) \sum_{a} \tilde{\pi}(a|s) \gamma^t A^{\pi}(s, a) \\ &= J(\pi) + \sum_{s} \sum_{t=0}^{\infty} \gamma^t P(s_t = s|\tilde{\pi}) \sum_{a} \tilde{\pi}(a|s) A^{\pi}(s, a) \\ &= J(\pi) + \sum_{s} \rho_{\tilde{\pi}}(s) \sum_{a} \tilde{\pi}(a|s) A^{\pi}(s, a) \end{aligned} \tag{2.11}$$

where $\rho_{\tilde{\pi}}$ is the discounted frequencies of visiting any state s following policy $\tilde{\pi}$ [260]:

$$\rho_{\tilde{\pi}}(s) = P(s_0 = s) + \gamma P(s_1 = s) + \gamma^2 P(s_2 = s) + ...$$

(2.11) implies that the performance $J(\tilde{\pi})$ of the new policy $\tilde{\pi}$ is guaranteed to be improved as long as the policy update satisfies:

$$\sum_{a} \tilde{\pi}(a|s) A^{\pi}(s, a) \geq 0$$

i.e., a nonnegative expected advantage at every state s.

However, (2.11) is hard to directly optimized due to the complex dependency of $\rho_{\tilde{\pi}}$ on $\tilde{\pi}$. TRPO introduced an approximation to $J(\tilde{\pi})$ by replacing $\rho_{\tilde{\pi}}$ with ρ_{π} (i.e., ignoring the changes of state visitation density $P(s_t = s)$ introduced by policy changes):

$$\mathcal{L}(\tilde{\pi}) = J(\pi) + \sum_{s} \rho_{\pi}(s) \sum_{a} \tilde{\pi}(a|s) A^{\pi}(s, a)$$

Provided that $\tilde{\pi}$ and π are relatively close, the following performance bound can be guaranteed:

$$J(\tilde{\pi}) \geq \mathcal{L}(\tilde{\pi}) - \frac{4\epsilon\gamma}{1-\gamma^2}\alpha^2 \tag{2.12}$$

where $\epsilon \geq 0$ is a constant and

$$\alpha = \max_s D_{TV}(\pi(s,\cdot), \tilde{\pi}(s,\cdot)) = \max_s \frac{1}{2}\sum_a |\pi(s,a) - \tilde{\pi}(s,\cdot)|$$

D_{TV} is the total variation divergence.

(2.12) shows that we can improve the policy performance $J(\tilde{\pi})$ by maximizing its lower bound. Given that $D_{TV}^2 \leq D_{KL}(\pi, \tilde{\pi})$ where D_{KL} is the KL divergence [224], policy updates can be performed by solving:

$$\begin{aligned} \max_{\tilde{\theta}} \quad & \mathcal{L}(\tilde{\pi}) \\ \text{s.t.} \quad & \bar{D}_{KL}^{\pi}(\pi, \tilde{\pi}) \leq \delta \end{aligned} \tag{2.13}$$

where $\tilde{\theta}$ are the parameters of the new policy $\tilde{\pi}$ and $\bar{D}_{KL}^{\pi}$ is the average KL divergence over all states visited following policy π.

With the help of importance sampling, we can derive

$$\sum_a \tilde{\pi}(a|s_t)A^{\pi}(s_t,a) = \mathbb{E}_{a\sim\pi}\left\{\frac{\tilde{\pi}(a|s_t)}{\pi(a|s_t)}A^{\pi}(s_t,a)\right\}$$

for a single s_t.

Thus, (2.13) can be simplified as:

$$\begin{aligned} \max_{\tilde{\theta}} \quad & \mathbb{E}_{s,a\sim\pi}\left\{\frac{\tilde{\pi}(a|s)}{\pi(a|s)}A^{\pi}(s,a)\right\} \\ \text{s.t.} \quad & \mathbb{E}_{s\sim\pi}\{D_{KL}(\pi,\tilde{\pi})\} \leq \delta \end{aligned} \tag{2.14}$$

PPO was proposed to reduce the computation complexity of TRPO caused by the KL divergence constraint. Instead of using a KL constraint to limit a large policy update, PPO penalizes large changes to the policy changes $r_{\tilde{\theta}}(s,a) = \frac{\tilde{\pi}(a|s)}{\pi(a|s)}$ through a clipping function:

$$\max_{\tilde{\theta}} \mathbb{E}_{s,a\sim\pi}\left\{\min(r_{\tilde{\theta}}(s,a)A^{\pi}(s,a), g(\epsilon, A^{\pi}(s,a))\right\} \tag{2.15}$$

where

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A, & A \geq 0 \\ (1-\epsilon)A, & A < 0 \end{cases}$$

With (2.15), the change of the new policy $\tilde{\pi}$ from the old one π is clipped at $1+\epsilon$ with a positive advantage and $1-\epsilon$ with a negative advantage.

2.3 Related Work on Controller Architectures

To improve the control plane's ability in request handling, different controller architectures are proposed. Based on the physical deployment of the controller plane, existing architectures can be divided into physically centralized and physically distributed architectures, while the distributed architectures can be further classified into flat and hierarchical architectures based on their physical organization. Table 2.1 summarizes the existing controllers with their respective architectures and characteristics.

2.3.1 Physically Centralized Controller Architecture

Initially, SDN was designed with a physically centralized controller architecture where only one controller was deployed to manage all forwarding devices within an SDN network. Obviously, the centralized controller can be easily overwhelmed by massive requests as the network scales up [289] and fails to meet the performance requirements. For example, the throughput of the first OpenFlow controller NOX [107] is 30k requests per second which is far from sufficient for a data center network [35, 269].

To solve this problem while maintaining the simplicity of the single controller design, one way is to enhance the capacity of the single controller by exploiting optimization techniques such as buffering, pipelining and parallelism to increase the controller throughput. For example, Maestro [53] utilizes parallelism to effectively leverage the capability of multi-core systems, together with input and output batching to reduce the

Table 2.1: Comparison of existing SDN controller architectures in the literature.

Control plane design		SDN controller architectures	Control plane architecture	Languages	Application
Physically centralize		NOX [107]	Physically centralized	C++/Python	Data center, enterprise
		NOX-MT [269]		C++	Data center, enterprise
		Maestro [53]		Java	Data center, WAN
		Beacon [84]		Java	Enterprise
		Ryu [15]		Python	Data center, enterprise
		Floodlight [5]		Java	Data center, enterprise
		DIFANE [296]		Python	Enterprise
		LazyCtrl [305]		Java	Data center, cloud
		DevoFlow [75]		Depends on controller used	Data center
Physically Distributed	Hierarchical	Kandoo [112]	Logically centralized	C/C++/Python	Campus, cloud, data center
		Orion [96]		Java	WAN
		Espresso [293]		Python, C	WAN, cloud
		B4 [140]		Python, C	WAN, data center
	Flat	ONOS [45]		Java	Enterprise, WAN
		Onix [158]		Python, C	Cloud, data center, Enterprise, WAN
		HyperFlow [268]		C++	Cloud, data center
		ElastiCon [80]		Java	Data center, enterprise
		DISCO [223]	Logically distributed	Java	Data center, enterprise, WAN

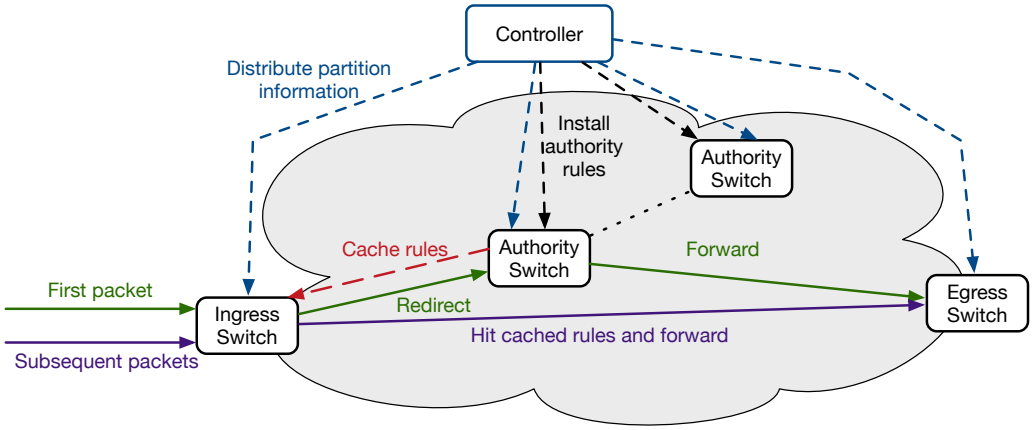


Figure 2.7: DIFANE, adapted from Figure 1 in [296]. (Dashed lines are control messages. Straight lines are data traffic.)

operation overhead [53]. Similar techniques are also adopted in multi-threaded controllers such as NOX-MT [269], Beacon [84], Ryu [15] and Floodlight [5].

On the other hand, other centralized controllers such as DIFANE [296], DevoFlow [75], and LazyCtrl [305] reduce the controller workload by devolving partial workloads of request processing to switches. As shown in Figure 2.7, DIFANE [296] generates and proactively distributes the flow rules across a subset of switches called authority switches. Upon receiving packets that do not match the rules, the ingress switch redirects the packets to the authority switches instead of the controller for further inquiry. Based on the received packets, the authority switch forwards the matching rules to the egress switch to direct the packet handling. Therefore, the traffic is kept among the switches in the data plane (e.g., TCAM and DRAM) instead of directing to the controller, providing lower delay and better scalability. However, these architectures require switch modifications and potentially break the general principles of SDNs [112].

Though the centralized controller provides the simplicity in the network design, it may not be feasible in a large scale network for several

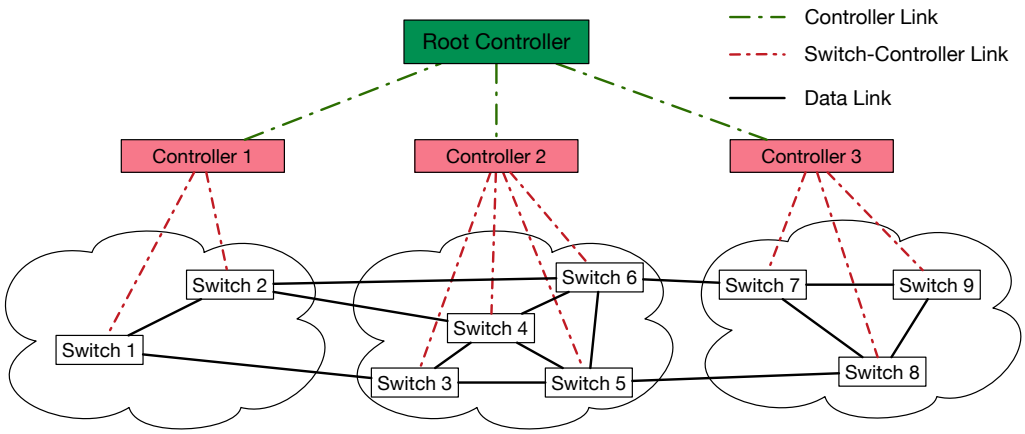


Figure 2.8: A hierarchical distributed controller architecture.

reasons. First, the workload of the controller grows with the number of switches. Second, if the network covers a large area, some switches may encounter long network latencies wherever the controller is placed. Third, since the system performance is bounded by the controller capacity, flow response time may grow significantly as demand increases with the size of the network. Finally, the single controller architecture is prone to a single point of failure.

2.3.2 Physically Distributed Controller Architecture

To mitigate the issues in physically centralized controller architectures (e.g., poor scalability and single point of failure), distributed controller architectures (e.g., Onix [158], ONOS [45], and DISCO [223]) have been introduced. In distributed controller architectures, multiple controllers are deployed in the control plane to collaboratively manage the network. Based on the control plane structure, distributed controller architectures can be further divided into hierarchical and flat architectures.

2.3.2.1 Hierarchical Controller Architecture

As shown in Figure 2.8, hierarchical architectures vertically partition the control plane into different levels. Controllers at the low level maintain the local view of their individually managed sub-networks and handle frequently generated local requests. On the other hand, the centralized controller at the top level abstracts each low-level controller and its managed sub-network as a logical node. Based on a global topology of these logical nodes [128], the centralized controller is capable of processing requests that required global information (e.g., requests triggered by inter-sub-network packets).

Inspired by DevoFlow and DIFANE, Kandoo [112] improves the control plane scalability by reducing its workload from frequent events without introducing switch modification or sacrificing the visibility of the control plane. Specifically, Kandoo distinguished local control applications[7] from non-local ones[8]. The workload from local events is offloaded from the logically centralized controller to a group of inter-dependent local controllers running local applications close to switches. Instead of managing switches, the logically centralized controller with a global network view controls all local controllers and handle non-local events.

B4, a software-defined Wide Area Network (WAN) connecting Google's data centers across the planet, is another example of hierarchical controller architectures. B4 operates at two levels. The lower/site level consists of multiple data-center sites/server clusters and each data-center site is managed by an SDN site controller running local control applications. All site controllers are managed by the top/global level which is equipped with an SDN Gateway and a central Traffic Engineering (TE) server. In particular, the SDN Gateway aggregates network information from the sites and provides an abstract network topology to the TE server where each site is represented as one node. Based on the global network

[7] Applications that process events locally using only the states from one switch.
[8] Applications that require access to the global network view.

information, the TE server optimizes the bandwidth allocation among competing applications across different sites by generating high-level TE policies that instruct site controllers on traffic forwarding.

Although hierarchical controller architectures alleviate the scalability issue, they suffer from the visibility loss of the local networks. This is mainly due to the abstracted hierarchical network view maintained by the centralized controller, which leads to path stretch problems[9] [73, 96, 148].

2.3.2.2 Flat Controller Architecture

As shown in Figure 2.9, flat architectures horizontally partition the network into different subsets and each subset of the network is managed by one controller. In terms of controllers' network view, flat architectures can be further classified into logically centralized and logically distributed architectures. Controllers in the centralized architectures maintain a global network view while the controllers in the latter one only maintain the local information of their managed sub-network while other sub-networks are abstracted as a logical node. In this thesis, we mainly focus on the centralized ones. Regardless of logically centralized or distributed architectures, inter-controller communication is required for state information (e.g., reachability information) exchanges [148].

Onix [158] and ONOS [45] are typical examples of logically centralized controller architectures. To provide a logically centralized control plane, Onix maintains a Network Information Base (NIB) which stores all network states (e.g., network topology, switch states) and is used by network applications. To enhance its scalability, Onix introduces a NIB distribution framework through partitioning, aggregation, consistency, and durability strategies to tackle the memory requirement and complexity of the NIB. First, it allows control applications to partition the NIB so that each controller can keep only a part of the NIB. Second, Onix allows a sub-network

[9] A stretch from node u to node v is defined as the ratio between the length of the actual path the traffic takes and the shortest length of the path from u to v.

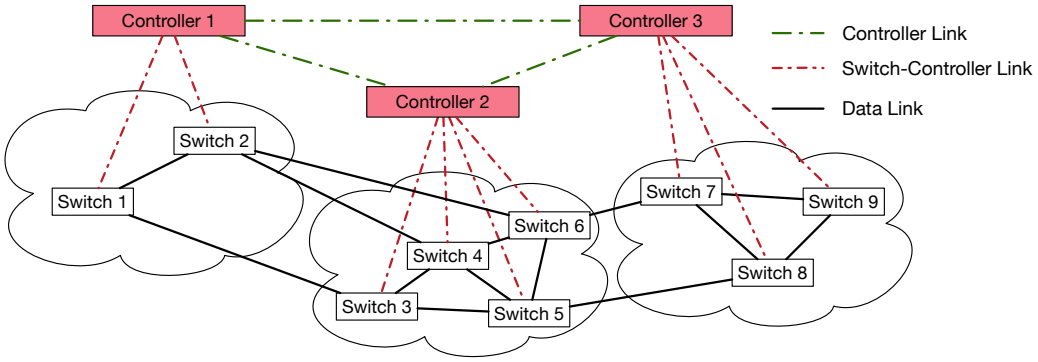


Figure 2.9: A flat distributed controller architecture.

managed by one controller to be aggregated as a single node in other controllers' NIBs. Third, since different applications may have different consistency requirements, Onix provides two data stores for durability and consistency.

Similarly, ONOS also maintains a database managing the global network view and shard network states across controllers. However, the remote database can potentially limit ONOS's scalability because the control applications need to frequently visit the remote database. To reduce the number of database operations, each ONOS controller caches the topology information in memory and updates it using a notification-based replication scheme. Apart from that, an inter-controller communication system is built to directly update controllers with any network state changes to avoid periodical database polling.

With a global network view, flat controller architectures can avoid the path stretch problem in hierarchical controller architectures. However, they may face the super-linear computation complexity growth[10] in large networks [96, 164].

Apart from that, though both hierarchical and flat architectures alleviate the scalability and reliability of the control plane, they are prone to

[10]For example, when the number of nodes increases by X in the network, the computation complexity of the control plane increases by X^2.

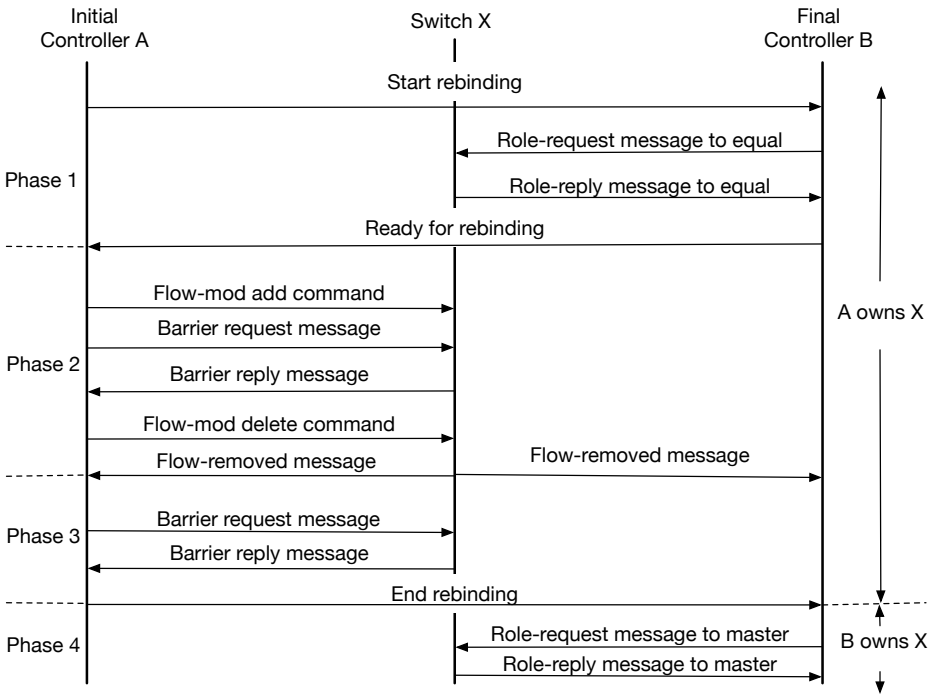


Figure 2.10: The four phase switch-migration protocol, adopted from Figure 2 in [80].

the load-imbalance issue stemming from the *Static Binding* between controllers and switches. As shown in Figures 2.8 and 2.9, each switch is pre-configured to statically bound to one controller and only sends its requests to that controller. For example, two switches (Switch 1 and Switch 2) are bound to Controller 1 and four switches are bound to Controller 2. Although this static-binding switch-controller association is simple, it results in unbalanced load distribution among controllers. For example, Controller 2 with 4 switches is likely to be overloaded because it is bound to more switches. Controller 1 is also possible to be overloaded if large amount of network traffic is generated in the sub-network due to popular public events.

To alleviate the load imbalance issue caused by static binding, distributed controller architectures [56, 65, 80, 280] featuring switch-

controller *Dynamic Binding* have been proposed where switches are no longer statically bound to a fixed controller. Instead, the switch-controller association is dynamically configured to enable controller load balance. In particular, these architectures periodically monitor the workload of each controller. When the load imbalance occurs, a switch will be unbound from the overloaded controller and rebound to another controller [80]. To guarantee the correctness of request processing and avoid packet loss during the switch-controller rebinding process, a four phase switch-migration protocol is used [80] as shown in Figure 2.10.

Though the load imbalance issue caused by static binding can be alleviated by dynamically rebinding switches, these dynamic-binding controllers still suffer from a few major limitations. Specifically, multiple messages need to be exchanged during the migration, which adds a considerable amount of communication complexity to the controller system. Moreover, the communication complexity results in a time-consuming migration process (ranging from millisecond level to second level depending on the original controller's workload [252] while the request response time is at the millisecond level). Thus, the migration may prolong network latency. More importantly, although the switch could be migrated, the requests from one switch can still be processed by only one controller, which still limits the scalability of the control plane. For example, the workload of the new controller will increase dramatically if the migrated switch is accumulating a large number of pending requests, increasing its risk of being overloaded. In other words, the controller resources are still not utilized effectively.

2.4 Related Work on Controller Placement

As we discussed in Section 1.1, RM in SDN must address the CPP. In general, existing studies on the CPP can be classified into two categories: the Uncapacitated Controller Placement Problem (UCPP) and the Capac-

itated Controller Placement Problem (CCPP), depending on whether the controller workload distribution (i.e., the CSP) is considered.

2.4.1 Uncapacitated Controller Placement Problem

The UCPP tackles the CPP without considering the CSP. For example, when the CPP was first proposed in [118], the distance between controllers and switches was adopted as the performance metric and several well-known network topologies were evaluated through simulations to find the optimal placement. Specifically, the CPP was considered within WANs and was formulated as a Mixed Integer Linear Program (MILP) that focuses on minimizing the propagation latency. To ensure the optimality of the solution, brute force was used in [118] to solve the CPP.

Similarly, [143] was proposed to design a scalable distributed SDN control plane from the perspective of the CPP. They claimed that a scalable distributed control plane should (1) restrict propagation delay between controllers and switches to avoid long controller response time and (2) restrict communication delay among controllers to guarantee timely network consistency update. Thus, they formulated the CPP by minimizing the number of controllers required to cover the whole network and selecting the locations of controllers that satisfy the propagation delay requirements. In terms of controller workload distribution, they simply assumed that controllers were not a bottleneck. More works on the UCPP can be found in Table 2.2.

Note that even with appropriate controller placement, we may still run into the risk of poor network performance if the requests cannot be properly distributed among controllers. Specifically, without solving the CSP properly, controllers can easily experience high workload. In such a situation, the controller processing time can dominate the total request response time [269, 291]. As reported in [74], the response time of a controller can increase significantly to 15 times of its normal value under light

Table 2.2: Comparison of the UCPP.

Reference	Objectives			Unknown number of controllers	Method	Application Scenario	
	Propagation latency	Reliability and resilience	Cost			Data center	WAN
Heller et al. [118]	✓				K-center	✓	
He et al. [116]	✓				Mixed integer programming	✓	
Lange et al. [165]	✓				Pareto simulated annealing		✓
Sahoo et al. [235]	✓			✓	Particle swarm optimisation and Firefly algorithm		✓
Zhang et al. [303]	✓				Integer linear programming and evolutionary algorithm	✓	
Ishigaki et al. [139]	✓				Greedy algorithm	✓	
Zhao et al. [304]	✓			✓	Exemplar-based clustering method		✓
Killi et al. [153]	✓				Inter linear programming		✓
Rath et al. [227]	✓		✓Operational expenses	✓	Non-zero-sum based game theory	✓	
Vizarreta et al. [277]	✓	✓			Mixed integer programming	✓	
Jimenez et al. [143]	✓	✓		✓	K-critical	✓	
Zhong et al. [306]	✓	✓		✓	Min-cover		✓
Liu et al. [186]		✓			Simulated annealing		✓
Liu et al. [182]		✓			K-means clustering and greedy algorithm		✓
Moazenni et al. [203]		✓		✓	Bully algorithm		✓
Hu et al. [133]		✓			Greedy algorithm	✓	
Guo et al. [109]		✓			Greedy algorithm	✓	✓

workload.

2.4.2 Capacitated Controller Placement Problem

In contrast to the UCPP, the CCPP explicitly recognized the importance of managing the workload distribution among controllers when dealing with the CPP. Therefore, studies shown in Tables 2.3 to 2.5 have been proposed to jointly determine the controller placement and the controller-switch assignment. Depending on how the CSP is handled, the CCPP can be further classified into three subcategories: (1) treating controller capacity as a constraint, (2) balancing controllers' workload, and (3) modeling the processing latency of a controller. These three subcategories have been summarized respectively in the three tables: Table 2.3, Table 2.4, and Table 2.5.

2.4.2.1 Controller Capacity as a Constraint

Studies in Table 2.3 explicitly treat the controller capacity as a placement constraint to prevent requests from any switches to overwhelm their associated controllers.

For example, the CCPP was first introduced in [291] by extending the original the CPP in [118]. This paper aimed to minimize the propagation latency subject to the constraint that the workload of all controllers should not exceed their respective capacities.

Similarly, a management framework for dynamically deploying multiple controllers within a WAN was proposed in [38] where both the number and locations of controllers were adjusted with changing network conditions. In this paper, the authors aimed to minimize the amount of communication overhead within the control plane in terms of controller state synchronization and switch statistics collection. Although flow setup time was also minimized in their objective, the flow setup time was purely measured as the propagation latency between switches and controllers while completely neglecting the controller processing time.

Table 2.3: Comparison of the CCPP with controller capacity as a constraint.

Reference	Objectives / constraints			Unknown number of controllers	Method	Application Scenario	
	Propagation latency	Reliability and resilience	Cost			Data center	WAN
Liao et al. [179]	✓			✓	Density-based cluster		✓
Yao et al. [291]	✓				Capacitated K-center		
Bari et al. [38]	✓			✓	Greedy knapsack and simulated annealing		✓
Khorramizadeh et al. [151]	✓		✓Operational expenses	✓	Greedy algorithm		✓
Sallahi et al. [237]	✓		✓Operational expenses	✓	Inter linear programming	✓	
Sallahi et al. [238]	✓		✓Operational expenses	✓	Linear programming	✓	✓
Mostafa et al. [151]	✓		✓Operational expenses	✓	Heuristic algorithm		✓
Hu et al. [131]	✓		✓Energy		Genetic Algorithm	✓	
Femandez et al. [92]			✓Energy		Greedy algorithm		✓
Killi et al. [152]	✓	✓			Simulated annealing		✓
Tanha et al. [263]	✓	✓		✓	Clique-based approach		✓
Survivor [207]			✓	✓	Integer Linear Programming	✓	
Li et al. [173]			✓	✓	Greedy algorithm		✓
Ros et al. [228]			✓	✓	Heuristic algorithm		✓

Clearly, the CSP is only handled in a simple manner using a capacity constraint in studies summarized in Table 2.3. With the capacity constraint, overloading controllers may not happen. However, some controllers can still experience high workload and the response time of a highly loaded controller can be very sensitive to workload changes. In this case, even though a relatively small amount of burst requests can seriously worsen the average response time [74].

2.4.2.2 Balancing Controller Workload

Follow-up studies as shown in Table 2.4 argued that the workload among controllers should be evenly balanced. For example, LiDy+ [272] calculated the number of controllers based on the network traffic and then assigned an equal amount of workload to each controller. [180] explicitly minimized the load imbalance among controllers while [162] ensured that the maximal workload difference among controllers should not exceed a predefined threshold to guarantee controller load balance. [123] proposed a Pareto-based Optimal COntroller-placement (POCO) framework to provide the selection flexibility to network operators depending on their specific requirements (e.g., minimize network latency or balance controller workload).

Compared to taking the controller capacity as a constraint, balancing the controller workload offers a better placement solution since it can avoid the response time growth resulting from controller load imbalance. However, these methods still face certain limitations. For example, balancing the controller workload can achieve good performance when all controllers are evenly distributed in the network with identical capacity. In a network where controllers with different capacities are located unevenly, dispatching more requests to nearby low-capacity controllers without overloading them obviously presents a better option. Apart from that, some studies [123, 180] oversimplified the measurement of controller workload by assuming each switch contributed the same amount of work-

Table 2.4: Comparison of the CCPP with controller load balancing.

Reference	Objectives / constraints				Unknown number of controllers	Method	Application Scenario	
	Propagation latency	Reliability and resilience	Cost	Controller load balancing			Data center	WAN
Liao et al. [180]	✓			✓		Genetic Algorithm with Particle swarm optimisation	✓	✓
Jalili et al. [141]	✓			✓		Multi-objective Genetic Algorithm		✓
Hu et al. [50]	✓			✓		Multi-objective Genetic Algorithm		✓
Killi et al. [154]	✓			✓	✓	K-means with Cooperative game		✓
Ksentini et al. [162]	✓			✓		Bargaining game	✓	✓
Ahmadi et al. [26]	✓			✓		Multi-start hybrid non-dominated sorting genetic algorithm		✓
GreCo [231]	✓		✓Energy	✓		Heuristic algorithm	✓	
Zhang et al. [298]	✓	✓		✓		Adaptive bacterial foraging optimisation algorithm		✓
POCO [123]	✓	✓		✓		Exhaustive search		✓
LiDy+ [272]	✓			✓	✓	Network partition		✓

Table 2.5: Comparison of the CCPP with controller processing latency.

Reference	Objectives / constraints			Unknown number of controllers	Method	Application Scenario	
	Latency		Reliability and resilience			Data center	WAN
	Propagation latency	Controller processing latency					
Cheng et al. [66]	✓	✓		✓	Greedy algorithm, primal-dual-based algorithm, network-partition-based algorithm		✓
Zeng et al. [297]	✓	✓		✓	Heuristic algorithm		✓
Wang et al. [281]	✓	✓		✓	Randomised fixed horizon control algorithm and modified stable matching with coalition game theory	✓	
Wang et al. [279]	✓	✓		✓	Clustering-Based Network partition		✓

load, which clearly does not hold in real networks.

2.4.2.3 Modeling the Controller Processing Latency

In order to accurately quantify the impact of the CSP on the CPP, studies have been proposed to explicitly measure the performance of the CSP while considering controller processing latency, as shown in Table 2.5.

For example, [281] considered the CPP as a dynamic controller provisioning and assignment problem. Specifically, the authors decomposed the dynamic problem into a series of one-time-slot switch-controller assignment problem. Within each time slot, the switch-controller associations are adjusted to accommodate traffic variance with the goal of minimizing the request response time and system costs in terms of state synchronization overhead and switch reassignment cost. The impact of the CSP was captured by the average request response time. In particular, given a CS solution, each controller was modeled as an independent queue and the response time was calculated using queuing theory.

Apart from that, a Multi-controller Selection and Placement Algorithm (MSPA) [279] was proposed to address the CSP together with the CPP. First of all, to reduce the maximum propagation latency between controllers and switches, MSPA partitioned the full network into a given number of sub-networks. Secondly, MSPA measured the impact of the CSP on the CPP as the queuing latency in controllers using an M/M/c queuing model.

However, in their work [279], the queuing model assumed that all requests within a sub-network must go through a centralized scheduler before reaching any controllers. Apparently, the scheduler can be easily overwhelmed by the enormous traffic and is prone to a single point of failure. In addition, the location of the scheduler must be carefully selected, which presents another challenge for the CPP. Furthermore, only one single queue is maintained by the scheduler and the scheduler makes dispatching decisions solely based on the controller workload without con-

sidering propagation latency, potentially increasing the response time if the request is forwarded to a remote controller. Experimental studies of MSPA will be conducted in Subsection 4.5.6.

In addition, although the impact of the CSP was explicitly quantified in these studies, they were mostly designed for binding-based controller architectures. In other words, each switch is connected to only one controller and all of its requests are sent to that controller. This limitation restricts the granularity of the CSP and may cause poor network performance (e.g., increased response time) since incoming requests from switches can be affected during the switch-controller reassignment process [255, 294].

2.4.3 Placement Algorithm

Apart from the problem formulation, different algorithms were proposed to find the optimal CP.

Exact methods are always a popular choice since they can guarantee optimality. In particular, when the CPP was first introduced, Heller et al. [118] adopted brute force to enumerate all possible CP solutions. Similarly, POCO [123] also conducted an exhaustive search to evaluate the entire solution space to find the optimal CP solution. Although these studies demonstrated that finding an optimal solution is computationally feasible in certain networks, they also pointed out that the optimality guarantee comes at the cost of high computational cost (e.g., weeks of CPU time) [118]. Moreover, current exact CP methods can only be applied to small-scale networks. With an increasing number of controllers, the computation cost grows exponentially, rendering the exact methods inapplicable.

Another category of widely-used approaches is based on game theory. For example, [162] formulated the CPP as an optimization problem with two contradictory objectives. One objective was to minimize the propagation latency between switches and controllers by deploying more con-

trollers to the network. To limit the number of controllers, an additional objective was adopted to minimize the communication overhead among controllers measured by the sum of their propagation latencies. Then a bargaining game was used to find the CP solution to balance the trade-off between two objectives. Similarly, [281] used game theory to calculate the switch-controller assignment to minimize the average response time in data centers. In [227], a non-zero-sum game-based distributed approach was presented to optimally deploy controllers. Despite the promising results, game-theory-based approaches have certain limitations. For example, game theory based approaches aimed to find the Nash equilibrium which is not necessarily the optimal solution [159, 249]. A well-known example is that the socially optimal solution in the prisoner's dilemma is cooperation which is not a Nash equilibrium.

As an alternative, heuristic methods have been widely used to address the CPP [66, 297] to balance the trade-off between computation time and solution quality, ranging from simple greedy strategies [92, 139, 151] to meta-heuristics (e.g., Simulated Annealing and EC) [141, 152, 165, 180, 235]. For example, simulated annealing was used in [152] to minimize the controller failure impact when deciding the CP solution. In [165], Pareto simulated annealing was adopted to solve the multi-objective CPP. Similarly, to search for the Pareto optimal CP placements, multi-objective GA has been adopted in both [141] and [180]. The promising results inspired us to tackle the CPP through an EC method. However, as a population-based approach, EC requires a large number of performance evaluations of randomly generated CP candidates until a satisfying solution is obtained, which can be computationally expensive. Especially in a large network, the corresponding search space becomes too large for an EC method to handle effectively.

To reduce the complexity of the search space in the CPP, network clustering has also been widely used [66, 279]. However, existing research treated each clustered sub-network as being completely independent. In

other words, workload sharing across different sub-networks is forbidden. Thus, when the traffic within a sub-network substantially increases and cannot be handled by existing controllers in the sub-network, the control plane will be significantly slowed down. This might even lead to a breakdown of the whole network.

2.5 Related Work on Controller Scheduling

Another important aspect of RM in SDN is how to schedule the use of controller resources. In other words, we need to consider how to distribute the requests from switches among all controllers to optimize network performance. Generally, the CSP is formulated on a discrete-time model where the time is split into multiple time slots [281]. CS is performed at the beginning of each time slot. Depending on how many controllers can a switch send its requests to, existing approaches can be widely divided into two categories: single-mapping-based CS and multiple-mapping-based CS [129, 254, 255].

2.5.1 Single-mapping-based Controller Scheduling

We refer the first category as single-mapping-based CS since each switch is assigned/mapped to only one controller which is in charge of processing all the requests generated from that switch. In this category, different approaches were proposed, e.g., approximation algorithms [65, 66, 100, 294], heuristics [38, 66, 74, 80, 278, 307, 308], and game theory [61, 64, 280].

- *Exact Methods*: Mohanasundaram et al. [206] formulated the switch-controller mapping problem as an MDP. At each time step, the state is represented as the current switch-controller mapping and the action is defined by all per-unit prices proposed by each controller. The reward of a controller is calculated as the product of the number of processed requests and its proposed per-unit price. In line with

these definitions, controllers compete with each other to serve more switches within their processing capacity so as to reduce the number of switch remapping/migrations and maximize their long-term reward. The optimal policy was found using the dynamic programming approach. Other exact methods such as brute-force search can be found in [258].

Although exact methods guarantee the optimality of the solution, their optimality guarantee comes at the price of high computational costs. Note that CS decisions need to be made periodically at each time slot to accommodate the traffic variation. In a network with high traffic fluctuation, the length of a time slot can be short. Therefore, computational expensive exact methods may not be applicable.

- *Approximation Algorithms*: Gao et al. [100] formulated the CSP as an integer programming problem with the goal of balancing the workload among controllers. The problem was transferred into linear programming using relaxation and solved by an approximation algorithm called deterministic rounding. Related approximation approaches on the CSP can be found in [65, 66, 294].

- *Heuristics*: As pointed out by existing studies [56, 278, 280, 281], the switch-controller assignment is NP-hard. It is computationally expensive to find the optimal solution in a large network. Thus, heuristic methods have been widely used in existing studies [38, 56, 66, 74, 80, 278, 307, 308].

  In [178], whenever the load difference between the heaviest-load controller and the lightest-load controller is greater than a predefined threshold, a switch will be migrated from the heaviest-load controller to the lightest-load one. Note that the selection of a suitable threshold that triggers the migration plays an important role in network performance. A small threshold may trigger frequent migrations which introduce huge communication overhead due to the

migration message exchanges and disruption to ongoing traffic. On the other hand, a large threshold leads to constant load imbalance because some controllers are undergoing high workload while others are still relatively idle [74]. However, how to select the threshold is not mentioned in their paper.

Note that request response time is the most intuitive and critical factor in measuring the network performance. Cui et al. [74] suggested that the changes in response time can be used as the threshold to trigger switch migration. Based on the proposed threshold, a greedy heuristic approach was proposed to first select the controller with the largest response time and the most heavily loaded switch. To reduce the controller's response time, the selected switch is then migrated to the controller under the lightest workload.

In [38], the CSP was also formulated as an integer programming problem to minimize the communication cost as well as switch reassignment cost to avoid frequent switch reassignments. To address this problem, two heuristic approaches were proposed including greedy knapsack and simulated annealing.

Similarly, note that migrating switches among controllers can improve the control plane resource utilization but also introduce additional migration costs[11]. A greedy approach was proposed in [278] to balance the trade-off between load balancing variation and migration costs. Similar ideas can also be found in EASM [130].

Motivated by the observation that the switch migration should be performed at the granularity of switch cluster instead of a single switch, BalCon [56] and BalConPlus [290] used a heuristic approach to cluster and migrate heavily connected switches based on their

[11]The migration costs are measured by the number of messages exchanging during the migration and the potential longer propagation latency between the switch and the new controller.

communication patterns.

Compared to exact methods, these heuristic approaches can solve the CSP within a reasonable time. However, they cannot guarantee the solution quality.

- *Game Theory*: Wang et al. [280] considered the CSP as dynamically deciding the switch-controller assignment to accommodate the traffic changes and minimize the request response time. The CSP was formulated as a stable matching problem with transfers. In particular, the CSP was first transformed into a classical college admissions problem and solved using matching theory to guarantee the worst-case performance. Then the obtained switch-controller assignment was fed into the coalitional game where switches were transferred between coalitions (i.e., controllers) to achieve a Nash stable solution.

  On the other hand, Chen et al. [61] proposed a zero-sum game-based approach to dynamically migrate switches from heavily loaded controllers to balance the control plane resources. Specifically, a game was initialized with a randomly selected switch (i.e., a commodity) managed by an overloaded controller. All neighboring available controllers were invited as game players for competition from which a new destination controller was selected with maximal utility changes. However, since every time only one switch is migrated which may not be able to alleviate the overloaded issue in the controller, multiple games need to be executed, reducing the load-balancing efficiency.

  In [64], the CSP was formulated as a resource utilization maximization problem with constraints on multiple resources (e.g., CPU, bandwidth, and memory), which was solved using non-cooperative game theory.

  By modeling the CSP as a game and solving it using game the-

ory, these approaches can provide solutions with guaranteed performance. However, selecting a suitable game model and formulating the problem using the model requires a high level of domain expertise. Also, as we mentioned in Subsection 2.4.3, these approaches aimed to find the Nash equilibrium which is not necessarily the optimal solution [159, 249].

Note that all algorithms in the single-mapping-based category are based on one-to-one switch-controller mapping, which inevitably inherits the limitations of binding-based controller architectures. First, CS is achieved by migrating the workload generated by one switch from one controller to another. Thus, CS can only be performed at a coarse switch level, restricting the opportunity of properly distributing workload across all controllers and reducing resource utilization. Second, identifying the suitable switch to be migrated is challenging. For example, if a switch with low workload is selected, additional switch migrations may need to be done to alleviate the controller overloading issue, resulting in low balancing efficiency. On the other hand, if a switch with high workload is selected, it may potentially overload the new controller, causing load oscillation among controllers. Third, single-mapping-based CS can easily cause frequent switch reassignment with burst traffic. For example, a traffic burst can overload a local controller and trigger switch migration from the local controller to its neighboring controllers. However, when the burst traffic passes, the migrated switches may be assigned back to the local controller to minimize the load imbalance. The repetitive migration inevitably introduces high migration overhead.

2.5.2 Multiple-mapping-based Controller Scheduling

In multiple-mapping based CS, the requests from one switch are no longer restricted to be handled by only one controller. Instead, they can be distributed and processed among multiple controllers.

To improve the efficiency of CS, Flow Stealer [252] was proposed to combine a lightweight flow-stealing approach with switch migration. Particularly, idle controllers share workloads with overloaded controllers temporarily by partially stealing requests from them. Switch migration will only be executed in the case when long-term traffic changes happen (e.g., a controller is undergoing a long period of overloading). With the help of flow stealing, Flow Stealer can significantly reduce the migration overhead in traffic-fluctuation or burst-traffic situations.

Both BalanceFlow [132] and COLBAS [245] are typical multiple-mapping based CS approaches which divided the workload among controllers in a centralized manner. In particular, a super controller was selected which periodically collected the network traffic information from all controllers (i.e., the number of requests generated by the switches managed by the controller) in the network. Based on the global traffic information, the super controller ran a greedy heuristic algorithm to partition the workload among all controllers. The partition decision was enforced by a switch extension which transformed the traffic redirection decision as flow rules installed in the switches. Thus, packets arriving at the switch which match these rules will be directly redirected to corresponding (different) controllers, reducing the workload of the original controller without introducing any additional propagation latency.

Considering that both BalanceFlow and COLBAS completely relied on the super controller to perform CS which can be easily overloaded, HybridFlow [292] and the controller load-balancing scheme in [257] were proposed with the aim to reduce the super controller's workload by introducing local CS. Particularly, they all considered a controller plane consisted of a super controller and multiple controller clusters. Different from BalanceFlow and COLBAS, they performed CS at two different levels. Whenever a controller was overloaded, it would first seek help from controllers within the cluster. To simplify the cluster level CS without the involvement of the super controller, a cluster vector was introduced in [257]

which was maintained by each controller. The vector contained the list of controllers within the same cluster, which can be used to make CS decisions directly by the overloaded controller. When the workload exceeded the processing capacity of the overall cluster, the super controller was involved to perform CS at a global level, i.e., distributing workload among controller clusters.

Sridharan et al. [255] formulated the CS as an optimization problem to minimize the total response time under the network resilience constraint[12] with the help of queuing theory. Numerical analysis was conducted to compare the multiple-mapping based CS with the single-mapping based CS. The results demonstrated that the multiple-mapping-based CS outperformed the single-mapping based CS in terms of request response time and resilience in the event of controller failure. It also showed that the multiple-mapping-based CS was more robust against dynamic traffic fluctuations regarding the probability of controller overloading.

Although a fine-grained multiple-mapping-based CS was proposed in [255] where CS was performed at the individual flow level, this paper only provided numerical analysis without developing algorithms to effectively find the CS solution. To address this issue, the same authors later proposed a Multi-Controller Traffic Engineering (MCTE) scheme [254] which developed an improved round-robin based heuristic algorithm to determine the switch-controller mapping as well as the request distribution among switches and controllers. The complexity of their proposed algorithm is $\mathcal{O}(M^2N)$ where M and N are the numbers of switches and controllers respectively.

Despite the benefits offered by the multiple-mapping-based CS, existing works have certain limitations. For example, to support the load redistribution from one switch to multiple controllers, an additional switch

[12]The network resilience constraint restricts the maximum fraction of requests sent from a switch to one controller, thus guaranteeing that at least a certain minimum fraction of traffic is not disrupted in the event of a controller failure.

extension was required in BalanceFlow. Apart from that, the request distribution in both BalanceFlow and COLBAS was performed at a coarse level based on source-destination switch pairs of the flows [255]. The introduction of a centralized super controller still limits the scalability of the control plane [255, 292]. Especially in BalanceFlow and COLBAS, the super controller is responsible for making all CS decisions, which can be easily overloaded. Although both HybridFlow and Sufiev et al.'s work [257] reduced the super controller's workload by introducing intra-controller-cluster CS, the CS decisions were made based on local information which may not be optimal [257]. Moreover, the cluster-level CS also introduces a considerable amount of communication overhead due to the frequent controller workload information exchanges within each cluster.

Compared to single-mapping-based CS, multiple-mapping-based CS enables CS to be performed at a fined-grained level (e.g., in a per-request manner [254, 255]). This flexibility comes at a cost of increasing the CSP complexity from $\mathcal{O}(N^M)$ to $\mathcal{O}(N^R)$ where N is the number of switches, M is the number of controllers, and R is the number of requests with $R \gg M$. Although a heuristic approach was proposed in [254] with complexity $\mathcal{O}(M^2N)$, it cannot guarantee the optimality of the obtained solution. Moreover, their algorithm was derived to minimize the network response time modeled with queuing theory while the effectiveness of the queuing model relies on several assumptions (e.g., the requests arriving at the switches follow a Poisson distribution, the system is stable and the model only reflects the long-term average performance in a stationary system).

2.5.3 Reinforcement-learning-based Controller Scheduling

Recently, machine learning has been used to solve RM problems in computer networks. For example, Barbancho et al. [36] proposed Sensor In-

telligence Routing (SIR), a routing algorithm based on Self-Organizing Maps (SOM) for wireless sensor networks with Quality of Service (QoS) guarantee. In particular, the QoS of a network link was represented as a weight which was calculated by SOM given the link latency, error rate, duty cycle, and throughput. Based on the calculated weights of different links, the path with the shortest weighted distance was calculated using Dijkstra's algorithm and was selected as the routing path from the source node to the base station. Similarly, Mao et al. [192] tackled the routing problems in dynamic traffic environments by constructing routing tables using a supervised deep learning approach. Liu et al. [184] exploited a deep belief net to evaluate the network link criticalness under different network traffic and removed the links that were not likely to be scheduled without affecting the optimality of the solutions. With the help of link removal, the search space of the optimization problem can be reduced, which can effectively reduce the computation cost. Other applications of machine learning algorithms can also be found in [51, 54, 183].

Among all machine learning algorithms, DRL has received an increasing amount of interest since RM can be naturally mapped to a sequential decision making process. In this thesis, we also consider DRL to be a powerful paradigm for solving the CSP for several reasons. First, an explicit model of the underlying complex environment is not required. DRL can automatically learn the optimal solution while interacting with the unknown dynamic environment through a trial-and-error process. Second, DRL can improve the current policy mainly based on experiences/data obtained from an old policy through a technique known as experience replay [201]. In comparison, the samples from previous generations cannot be reused in the next generation in EC methods where candidate solutions need to be extensively reevaluated in each generation in either simulated or real-world environments. Thus, compared to EC methods, the sample cost of training any new policies in DRL can be greatly reduced. Third, performing CS under a specific network setting can be

naturally formulated as an MDP (see Subsection 2.2.6.1 for detailed MDP discussion). Due to these reasons, DRL has been successfully utilized to design scheduling policies in many related resource management problems [67, 193, 194, 210, 265].

For example, Liu et al. [185] proposed a hierarchical framework comprising global and local tiers to respectively address the resource allocation and power management problem in cloud computing systems. In particular, the global tier was in charge of Virtual Machine (VM) allocation among a cluster of servers using a centralized agent trained by DQN. Whenever a job (VM) arrives, the agent observes the environment state (i.e., servers' states and the new job description) and selects a server from the cluster for VM allocation. Thus, the size of the action space equals the total number of servers. On the other hand, the local tier used a power manager trained by Q-learning to adaptively turn on or off the servers to reduce power consumption while maintaining performance degradation to an acceptable level based on the workload predicted by a long short-term memory network.

Li et al. [174] applied DRL to address the network slicing[13] problem in 5G. Specifically, given a fixed number of existing slices with the shared aggregated bandwidth and the demands of each slice, the agent trained by DQN dynamically adjusts the bandwidth sharing so as to maximize the resource utilization while maintaining high user experience satisfaction. Similarly, Hua et al. [135] proposed a generative adversarial network-powered deep distributional Q-network to allocate network resources for diversified services in 5G networks.

Moreover, Tesauro et al. [265] proposed an RL-based approach to automatically allocate the server resources in data centers. DeepRM [193] was proposed to address the multi-resource cluster scheduling problem

[13]Network slicing is a concept proposed to slice the physical and computational resources of the network into different network slice instances and each instance is configured to meet the specific service requirements of the slice tenant.

using a DNN policy trained by TRPO [240] to optimize various objectives, e.g., average job completion time and resource utilization. Similarly, Decima combined DRL and graph neural networks to learn workload-specific scheduling policies for data processing clusters [194]. Chinchali et al. [67] leveraged the delay-tolerant feature of IoT traffic and developed an RL-based scheduler to handle traffic variation so that the network utilization can be constantly optimized.

Different from heuristics, DRL fully automates the policy design process and noticeably improves the adaptability and performance of designed policies [194]. However, existing DRL-based approaches for request dispatching policy design suffer from several limitations:

1. *Impractical problem formulation*: It is typical to design a policy by a single learning agent supported by global network information (i.e., a fully observable environment). For example, in [193], a centrally trained agent must learn to dispatch jobs among a large cluster of computers. Such a central approach is prone to scalability issues [300]. Particularly, the use of a single agent inevitably introduces extra communication delay. Meanwhile, obtaining timely global information over the entire SDN network can cause substantial communication overhead [287]. Even though these issues can be alleviated by employing multiple co-learning agents, the single-agent DRL algorithm they use cannot cope with inter-agent interference and localized network information, resulting in poor and unpredictable network performance.

2. *Non-adaptive and inefficient policy design*: Many existing approaches directly train a DNN as their policy with a fixed number of output nodes [240, 242]. Each output node corresponds to a separate SDN controller. Unfortunately, the trained policies may fail to function well whenever the number of controllers is changed in order to meet the varying traffic demand in the network.

3. *Ineffective policy training*: Existing single-agent DRL algorithms cannot allow multiple learning agents to train an *adaptive policy* reliably and

effectively. In particular, new techniques must be developed to compute the policy gradient with respect to an adaptive policy that supports a dynamically changing number of controllers. Moreover, inter-agent interaction must be carefully controlled and inter-agent collaboration must be explicitly encouraged through effective information sharing at the policy training (or design) stage.

2.6 Summary

This chapter first introduced the basic concepts of SDN and optimization techniques. Guided by the research goal of tackling the RM problem in SDN, related work on SDN controller architectures, controller placement methods, and controller scheduling algorithms have also been reviewed in this chapter.

The limitations of the existing work highlighted in this chapter form the background and motivate this research. Enhancing the scalability of the SDN control plane is challenging which requires effective management of the controller resources. The main goal of this research is to solve the SDN RM problem from the architectural and algorithm design perspectives. The literature showed that various distributed controller architectures have been developed to tackle the RM problem. Moreover, many algorithms have been proposed to tackle the CPP and the CSP. However, there are still a number of open questions that need to be further investigated.

In particular, the limitations of existing work can be summarized below.

- *Distributed SDN Controller Architecture*: Distributed controller architectures have been widely used to enhance the scalability of the SDN control plane. However, existing controller architectures feature a binding-based switch-controller association which restricts the re-

quests from a switch to be only processed by one controller. This association results in unbalanced load distribution among controllers since each switch comes with different workload and the workload can be time-variant. Realizing the limitations stem from the switch-controller binding, research must be conducted to propose a new distributed controller architecture.

- *Controller Placement*: A majority of exiting studies on the CPP aimed to improve network performance without explicitly quantifying the impact of the CSP. Particularly, the CSP is usually handled in a simple manner, e.g., treating the controller capacity as a placement constraint to prevent controllers from being overwhelmed by requests from associated controllers or assuming all switches have the same workload. This simplification may not always apply to real network situations. Although a few studies have explicitly quantified the impact of the CSP using queuing theory, they were designed for the binding-based controller architectures, which restricted the granularity of the CSP. Due to the large search space in the CPP, network clustering has been widely used to divide the network into independent clusters. However, when a cluster encountered traffic bursts that cannot be handled by existing controllers, the control plane will be significantly slowed down. Therefore, further research must be performed to tackle both the CPP and the CSP simultaneously in a coherent framework and to effectively handle the traffic bursts.

- *Controller Scheduling*: Effectively scheduling the given controller resources by dispatching requests from switches to controllers is a challenging task. This task will be even more difficult without the switch-controller binding constraint. Specifically, existing algorithms were mostly designed for the binding-based association. Thus, the CS decision is made for each switch. In comparison, when CS is performed at a per-request level, each request demands

a separate CS decision. Note that requests obviously outnumber the switches, leading to a significant increase in the complexity of the CSP. Although DRL has demonstrated its potential in tackling challenging scheduling tasks, existing DRL applications were mainly designed for centralized scheduling. Request dispatching decisions were made using timely global information that is not always available. Apart from that, current fixed DNN policy designs fail to function well whenever the number of controllers is changed in order to meet the changing network traffic demand. Moreover, existing policy training algorithms were designed for training policies directly represented as a DNN that targets networks with a fixed number of controllers. They did not support adaptive policy training since they cannot calculate the gradient of an adaptive policy. Therefore, research must be performed to design an adaptive policy and develop new training algorithms.

Based on the limitations of existing work discussed in this chapter, the following chapters will address the limitations by developing a new architecture and new algorithms. Chapter 3 will develop a new distributed controller architecture featuring a bindingless switch-controller association. Chapter 4 will formulate a new problem which captures the strong inter-dependencies between the CPP and the CSP. In line with the new formulation, a new algorithm will be proposed. Chapter 5 will propose a DRL-based approach to automatically learn an effective and generally applicable policy that dispatches requests from switches to appropriate controllers.

Chapter 3

BLAC: A Bindingless Architecture for Distributed SDN Controllers

3.1 Introduction

Distributed controller architectures (e.g., ONOS [45] and Onix [158]) have been proposed for Software-Defined Networking (SDN) to ensure scalability and reliability. As we discussed in Subsection 2.3.2, one major drawback of the existing distributed architectures is the uneven load distribution among controllers stemming from the *Static Binding* between controllers and switches [80]. In particular, it restricts the requests from a switch to be only processed by one controller [14]. Since each switch comes with different workload and the workload can be time-variant, the static-binding association renders the bound controller susceptible to either being overloaded or underloaded.

To address this issue, several existing studies [80, 280] introduce *Dynamic Binding* by adopting a switch migration mechanism [80] that rebinds switches from overloaded controllers to underutilized controllers. However, as we mentioned in Subsection 2.3.2, the migration process adds a considerable amount of complexity to the system and incurs significant network latency due to the use of a four phrase switch migration proto-

col [80]. Furthermore, the performance of switch migration is limited as it works at the switch level. In view of the limitations stemming from switch-controller binding, designing a new architecture is necessary.

However, designing a new architecture is challenging since it has to satisfy several design principles as we mentioned in Subsection 1.2.1. In particular, the new architecture should guarantee its scalability and compatibility without introducing any potential bottlenecks of network performance or extensions to the switches. In addition, the new architecture should provide flexible and transparent CP and CS without the interruption of switch-controller re-binding. Moreover, it should be extensible with easy support for new functionalities.

3.1.1 Chapter Goals

In fulfillment of Objective 1 stated in Section 1.3, a new Bindingless Architecture for Distributed SDN Controllers (BLAC) is developed in this chapter. In view of the limitations caused by the switch-controller binding, BLAC features bindingless association between switches and controllers where a switch does not have to be bound to a controller. In other words, the requests from a switch can be processed by any controllers, providing more flexibility and finer granularity in distributing workload among controllers (i.e., CS). Moreover, since the switches are no longer bound to any particular controllers, the bindingless association minimizes the network interruption caused by the change of CP. Precisely, this chapter aims to address the following objectives:

- Design a new architecture that achieves bindingless association between switches and controllers;
- Implement a prototype of the new architecture based on real-world controllers;
- Compare the performance of BLAC with both static and dynamic

binding-based distributed controller architectures (e.g., ONOS [45] and ElastiCon [80]) in terms of response time and throughput.

3.1.2 Chapter Organization

The rest of this chapter is organized as follows. Section 3.2 presents the system architecture by demonstrating the architecture components and highlighting the architecture features. Section 3.3 describes several supported scheduling algorithms in BLAC. Implementation details of BLAC are discussed in Section 3.4 and the evaluation results are reported in Section 3.5. Section 3.6 concludes this chapter.

3.2 System Architecture

This section demonstrates the key components of BLAC and highlights its unique features. At the end, we discuss how BLAC satisfies the design principles we presented in Subsection 1.2.1.

3.2.1 Architecture Components

In view of the limitations of switch-controller binding, BLAC features a novel bindingless switch-controller association in SDN controller. This is achieved by introducing a new scheduling layer which is logically located in between the data plane (i.e., switches) and the control plane (i.e., controllers). Within the scheduling layer, multiple scheduling instances called schedulers can be flexibly deployed[14]. Specifically, each scheduler consists of three components including the Switch Adapter, the Controller Adapter, and the Scheduling Module as shown in Figure 3.1. The details of each component are provided as follows.

[14]During the implementation, the schedulers can be placed in the data plane or the control plane

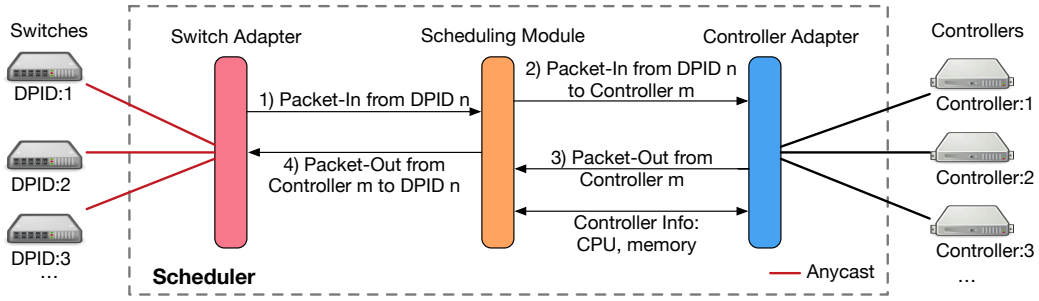


Figure 3.1: The design of a scheduler in BLAC.

- *Switch Adapter:* The switch adapter serves as an interface that connects the data plane (switches) and the scheduling layer. Through the switch adapter, the scheduling layer can receive requests from the switches and/or send responses back to the switches. Specifically, the adapter listens on a pre-configured anycast IP address for requests coming from neighboring connected switches. Detailed discussion on the use of anycast will be provided in Subsection 3.2.2.1.

- *Controller Adapter:* Similarly, the controller adapter manages the interactions between the scheduling layer and the control plane (controllers), such as connection establishment and packet transmission. Apart from that, controller-related statistics (e.g., number of responses received from each controller) is also collected via the controller adapter, which can be used to address the CSP.

- *Scheduling Module:* The scheduling module plays a critical role in adjusting controller workload distribution. In particular, whenever a new request is received from a switch, the new request will be routed to a controller chosen according to the deployed request dispatching policy. The output of a policy can be the probabilities of dispatching any new requests to available controllers. Alternatively, it can also be a selected controller for the new request. Note that a policy can be a manually designed scheduling heuristic (e.g., round-robin) or in

a form of a neural network trained by machine learning algorithms (e.g., reinforcement learning). Depending on the network setting and performance requirement, different policies can be designed and applied in the scheduling module. For example, in a network where all controllers have identical capacities and are located close to switches, round-robin can be a good option due to its simplicity. On the other hand, a neural network based policy may be more preferable in a complicated network environment (e.g., uneven propagation latency and hierarchical controller settings).

3.2.2 Architecture Features

With the new scheduling layer, BLAC enables *bindingless switch-controller association*. BLAC also takes advantage of *anycast* technology to minimize the switch-scheduler propagation latency as well as simplifying switch configuration without introducing any hardware or special software modification. Moreover, BLAC also enables directly use of existing consistency maintenance mechanisms supported by many distributed controller systems [45, 158] to maintain a *consistent and centralized network view*.

3.2.2.1 Anycast Switch-Scheduler Association

In our architecture, a unique anycast controller IP address which is listened to by schedulers is assigned to all switches. The use of anycast has two essential advantages:

- *Easy Configuration*: The use of anycast technique simplifies and unifies BLAC deployment by assigning a unique anycast controller IP address to all switches.

- *Low Propagation Latency*: With the help of anycast technique, switches could automatically connect to a scheduler that has the least distance to them. Thereby, the propagation latency could be minimized.

3.2.2.2 Bindingless Switch-Controller Association

In BLAC, each switch connects to the scheduling layer instead of directly connecting to a controller. With the help of the scheduling layer, requests from the switches can be transparently and flexibly distributed among controllers, providing request-level CS granularity as we discussed in Subsection 1.2.2.2. The logical flow of request dispatching, as shown in Figure 3.1, can be described as follows.

1) The switch forwards its requests to the closest scheduler using anycast.
2) Inside the scheduler, the switch adapter forwards the new requests to the scheduling module.
3) With the help of the policy, the scheduling module generates a request dispatching decision instructing the controller adapter to forward the new request to a specific controller.
4) The controller sends a response back to the scheduler after it finishes processing the request.
5) The controller adapter intercepts the response from the controller and feeds it into the scheduling module.
6) The scheduling module then identifies the corresponding switch that receives the response and sends it back through the switch adapter.

3.2.2.3 Preserving Consistency among Controllers

To guarantee the logically centralized control in distributed controller architectures, all controllers must maintain a consistent global network view. To achieve this goal, sharing and distributing the network state information among all controllers is inevitable.

BLAC performs request dispatching without violating the information consistency among controllers. BLAC utilizes the network topology consistency maintenance mechanism widely and efficiently supported by the

distributed controller systems [45, 158]. Taking ONOS as an example, each controller caches a consistent global network view in memory which is maintained by a notification-based replication scheme [45]. This mechanism allows us to dispatch requests to different controllers which have a consistent network view and obtain the correct results.

3.2.2.4 Addressing Architectural Challenges

In this subsection, we will discuss how BLAC satisfies the design principles we presented in Subsection 1.2.1.

- *Scalability*: The new scheduling layer introduced by BLAC is unlikely to become a performance bottleneck due to the following reasons:
  - The schedulers are lightweight. As mentioned in Section 3.2, the schedulers are solely responsible for dispatching requests to controllers according to pre-calculated probabilities, which incurs negligible computation overhead (simply toss a coin for every request to be dispatched).
  - The schedulers barely incur communication cost since they make request dispatching decisions completely independently and in a fully decentralized manner.
  - Schedulers can be easily created and flexibly deployed in the scheduling layer to avoid the scheduling layer being overwhelmed. Since the schedulers are neither computationally expensive nor communication intensive, they can be easily implemented as light-weight Virtual Network Function (VNF)[15] components. When the number of requests grows suddenly and substantially within the network, more schedulers can be easily deployed.

[15]Virtual network function (VNF) refers to network functions (e.g., firewall and load-balancer) that can be deployed on commercial off-the-shelf hardware (e.g., virtual machines) instead of proprietary, dedicated hardware [176]

- *Transparency*: With the newly-introduced scheduling layer, BLAC achieves bindingless switch-controller association. A switch does not have to be bound with a controller, that is, the requests from the switch can be processed by different controllers, enabling transparent and request-level request dispatching. In terms of CP, connections between the scheduling layer and any newly deployed controllers need to be established. However, any ongoing traffic between the data plane and the control plane will not experience network interruption because when new controllers are being connected, the scheduling layer can continue to send upcoming requests to whatever available controllers.

- *Compatibility*: From an architectural perspective, the scheduling layer is transparent between control and data planes. Thus, the scheduling layer is compatible with existing SDN design which can be justified from two aspects:

  - From controllers' point of view, the scheduling layer is part of the data plane since the scheduling layer is responsible for dispatching requests among controllers. Existing distributed controller architectures (e.g., ONOS [45] and Onix [158]) can be easily integrated with BLAC without introducing any major changes. Implementation details on integrating BLAC with existing distributed controller architectures can be found in Subsection 3.4.2.
  - Similarly, switches barely notice the existence of the scheduling layer since no hardware changes or software extensions are needed for existing SDN switches. In particular, all switches connect to the scheduling layer using an assigned anycast IP address as we discussed in Subsection 3.2.2.1.

- *Extensibility*: The introduced scheduling layer provides the opportunities of integrating new functionalities. For example, flexible re-

quest dispatching is enabled by employing different policies in the scheduling layer, such as random heuristic [110, 146] and weighted round robin [88] which will be discussed in Section 3.3. Moreover, since every request travels through the scheduling layer to the control plane, it allows additional flow rules to be cached in the scheduling plane so that the number of requests sending to the controllers can be reduced.

- *Separation of Concerns*: Guided by this principle, each scheduler consists of three components (i.e., the Switch Adapter, the Scheduling Module, and the Controller Adapter) as we mentioned in Subsection 3.2.1. Each component communicates with each other using well-defined interfaces while detailed implementation is encapsulated inside. For instance, with the help of the switch adapter, the scheduling module only concerns about how to dispatch requests regardless of what SDN southbound interface (e.g., OpenFlow [14] or OVSDB [11]) is used by the switches.

3.3 Request Dispatching Policies

The main objective of the scheduling module is to dispatch requests to suitable controllers to minimize the request response time and maximize the throughput. In this chapter, we mainly focus on examining the effectiveness of BLAC in CS. For simplicity, the policies we investigated in this chapter are mostly heuristics, e.g., random scheduling. Even with simple heuristics (e.g., random scheduling), BLAC can still significantly outperform the binding-based architectures in terms of response time and throughput as demonstrated in Subsection 3.5.2. We also simplify the network setting by considering a small-scale network, e.g., local data centers hosted by universities or private enterprises on the premises of the organizations to serve local users. Designing and evaluating policies in large-

scale networks (e.g., WAN) with heterogeneous controllers will be further investigated in Section 4.3 and chapter 5.

To start with, we consider a network composed of a cluster of SDN controllers and schedulers. Each scheduler assigns a received request to a controller. Each controller maintains a request queue and executes requests in a FIFO manner. We here assume that all controllers have similar capacities and each request requires similar controller workload, which are fairly common in a data center [37].

According to existing studies [80], CPU is typically the throughput bottleneck of a controller and the CPU load is roughly proportional to the request arrival rate at the controller. Thus, we define the load on a controller as its CPU utilization. Since the network we considered is deployed locally, the propagation latency can be safely ignored (below 1ms) [108]. Based on the above assumptions, to optimize the network performance in terms of response time and throughput, requests should be dispatched in a way that all controllers' workload is balanced.

3.3.1 Omniscient Scheduling

The omniscient scheduler uses a greedy policy based on real-time global information about the workload of each controller. Specifically, whenever a request arrives, the scheduler forwards it to the least loaded controller, which clearly is a suitable decision to achieve controller load balancing.

However, obtaining real-time global network information requires each controller to frequently update its workload to all schedulers, which inevitably introduce communication overhead. Particularly, in a network with M schedulers and N controllers, the total number of messages to be sent from the control plane to all schedulers per update is M^N, which can cause substantial communication overhead especially with a high update frequency or in a network with a considerable number of controllers and schedulers. Thus, this method may not be practical.

Moreover, it is unnecessary to achieve the best possible balance which may not always be essential for the purpose of improving system performance. For example, 20% and 25% CPU utilization results in almost the same response time [80].

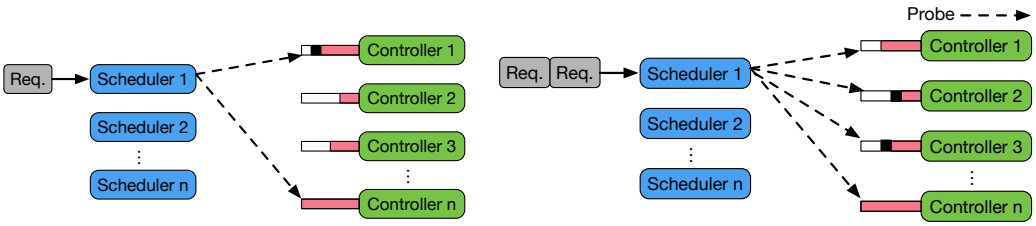
Therefore, we argue that a light-weight policy that does not incur heavy communication overhead is more suitable.

3.3.2 Random and Round-robin Scheduling

BLAC supports two widely-used simple-yet-effective approaches for distributing workload including RANDom (RAND) scheduling and Round-Robin (RR) scheduling. RAND simply distributes a request to a controller chosen independently and uniformly at random. On the other hand, RR passes a received request to a specific controller by circling through all available controllers.

Since both RAND and RR require minimum controller status information, they can easily scale to a large-scale network with many distributed controllers and schedulers. Due to their simplicity, they are widely used in real-world networks [146, 177]. Note that the effectiveness of both RAND and RR relies on the assumption that all controllers have identical processing capacities, propagation latencies, and initial workload, which is fairly common in data centers. However, in the presence of exceptions or if the assumption is not satisfied, their performance would degrade. For instance, even though a controller is heavily loaded, it still receives the same number of requests as an underloaded controllers, which can easily overload the controller. Therefore, it is natural to raise a question: is it possible to solely utilize limited controller information (e.g., CPU utilization) to improve the scheduling performance?

In this chapter, we will study the usefulness of two different approaches: one is the Improved RANDom (IRAND) scheduling, which relies on instantaneous information obtained from a small number of con-



(a) Per request probing selects Controller 1

(b) Batch assignment selects Controller 2 and Controller 3

Figure 3.2: Improved randomized scheduling. In comparison to per request probing, batch assignment is more likely to find the least loaded controllers for the requests.

trollers; the other is the Weighted Round-Robin (WRR) scheduling which periodically updates controller information to reduce the communication overhead.

3.3.3 Improved Random Scheduling

BLAC adopts an improved version of RAND taking inspiration from the power of two choices technique [200] as shown in Figure 3.2(a). Intuitively, the approach probes a small number of randomly selected controllers for their workload and assigns a request to the least loaded one. The scheduler repeats this process for each request received. With the small amount of communication overhead between the schedulers and controllers, the approach can improve the response time significantly compared with RAND [200]. In our implementation, we probe 2 controllers for each request as further explained in Section 3.5.

One problem with this approach is that the performance gradually deteriorates as the overall workload increases, since it is difficult for schedulers to find less loaded controllers on which to place requests. To address this problem, we further improve the approach by allowing batch

assignment of consecutive requests. Without batch assignment, one pair of probes may have easily sampled two heavily loaded controllers, resulting in no benefits to load balance. Batch assignment aggregates load information from the probes sent for consecutive requests and assigns them to the less loaded of all the controllers probed. Specifically, upon receiving d requests, the schedulers send probes to $2 \times d$ controllers. Clearly, there exists a trade-off between the load balance performance and the response time; with a large d value, it is more likely to assign the request to a less loaded controller while sacrificing the response time as more requests are buffered. In our implementation, we let $d = 2$ based on our experiment in Section 3.5, which yields desirable compromise between performance and communication cost.

3.3.4 Weighted Round-robin Scheduling

Instead of sending probes for each received request, BLAC also allows controllers to periodically push their workload information to the schedulers. Based on the information, a set of scheduling algorithms such as greedy scheduling and Weighted Round-Robin (WRR) scheduling can be adopted. Similar to omniscient scheduling, greedy scheduling simply dispatches requests to the least loaded controllers. However, since the workload information from controllers are updated periodically, greedy scheduling can easily overload the least loaded controller with a long updating period since all requests will be sent to it until the next workload update.

In comparison, WRR assigns requests to controllers in a cycling order, which is similar to RR. But instead of distributing requests evenly, the number of requests sent to a controller is proportional to the reciprocal of its workload. Thus, a heavily loaded controller receives less requests compared to an underloaded controller, which ensures load balance in the control plane.

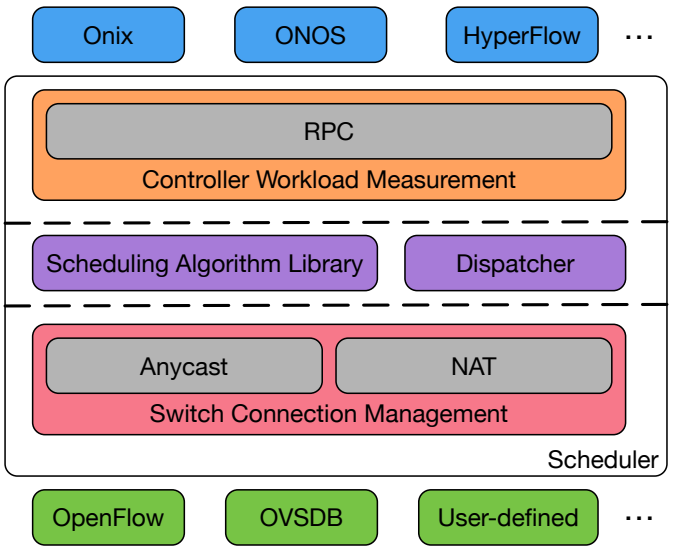


Figure 3.3: System component of BLAC.

3.4 Implementation

In this section, we provide the detailed implementation of BLAC. As shown in Figure 3.3, BLAC can support a range of protocols such as OpenFlow [14] and OVSDB [11]. Since currently OpenFlow is the most widely accepted and deployed SDN protocol [161], our discussion will mainly focus on OpenFlow.

Based on the OpenFlow protocol, we discuss the details on how to manage the switch connections using anycast and NAT. After establishing the switch connection, BLAC dispatches requests to controllers as determined by selected algorithms from the scheduling algorithm library. Finally, we present how to measure controller workload using REST APIs and RPC.

3.4.1 Anycast

As we mentioned in Subsection 3.2.2.1, to simplify switch configuration and reduce propagation latency, switches connect to the scheduling layer

using an anycast IP address. Specifically, anycast [220] is a network addressing and routing method in which packets from a single sender are routed to the topologically nearest node in a group of potential receivers. The deployment of anycast can be divided into two steps. First, multiple schedulers listen on the same IP address. Second, modify the routing tables and direct packets to the topologically nearest instance.

Although the idea of using anycast is simple, the practical implementation entails a technical hitch. Anycast was originally designed for connectionless protocols (such as UDP), rather than connection-oriented protocols such as TCP used by OpenFlow. To ensure OpenFlow to work with anycast, one has to assume a deterministic destination for all requests originated from a switch. The assumption holds true when the network topology remains unchanged, which is the general case [163, 217]. However, there are still cases where the scheduler receiving the requests may change from time to time due to topology changes, silently breaking any ongoing conversations [220]. Therefore, anycast is not completely compatible with existing OpenFlow networks.

To enable anycast for OpenFlow, a solution that features two-round connection establishment has been implemented in our prototype.

First-Round: A switch connects to a scheduler using an anycast address such that it will be directed to the nearest scheduler instance. After the connection is established, the scheduler alters the controller IP setting on the switch to the unicast address of the scheduler. To achieve this, we could leverage switch configuration protocols, such as the OpenFlow Management and Configuration Protocol (OFConfig) [13] and the Open vSwitch Database Management Protocol (OVSDB) [11], which are widely supported by the switches [12].

Second-Round: Upon receiving the unicast address of the scheduler instance, the switch establishes another connection with the instance using its unicast address.

As an alternative solution, BLAC also supports the Network Address

Translation (NAT) technique which preserves the advantages of anycast. Specifically, we still configure all the switches with the same virtual controller IP address. The virtual IP address is then mapped to a real scheduler IP address at runtime based on the NAT table, which is configured according to the scheduler workload and the distance between schedulers switches. And packets could be transparently directed to the appropriate scheduler. As the NAT technique inherently supports the TCP connections, the NAT-based method could avoid the reconnection problem of anycast while preserving its convenience.

3.4.2 Integrating BLAC with ONOS

BLAC is designed to provide universal support for distributed controller systems. In this section, we implement our architecture based on ONOS. However, it should be noted that minor modifications might be needed during the implementation due to the mechanisms for maintaining network consistency varied from different controller systems.

In ONOS, multiple controllers cooperate with each other and behave as one logical entity. In other words, each controller has a synchronized global view of the entire network which is maintained by a notification-based replication scheme as we discussed in Subsection 3.2.2.3. Therefore, regardless whichever controller a request is sent to, the response is expected to be the same. This characteristic greatly facilitates the implementation of BLAC as it can easily dispatch requests to different controllers according to the policy.

However, during the implementation of BLAC, we notice that dispatching requests to different controllers experienced failures on ONOS. We investigate the cause and find that ONOS assumes that only the master controller could process the Link Layer Discovery Protocol (LLDP) packets received from the switch, for the purpose of discovering and updating the underlying network topology. Since our policy indiscriminately dis-

patches the LLDP packets to any controller in the cluster, ONOS fails to build up the network topology.

To tackle this problem, two solutions are proposed. The first one requires the schedulers to detect the LLDP packets and dispatch them to a switch's master. As a result, the schedulers have to maintain the master controller record for every switch. The information can be obtained by inquiring the switches about a controller's role with OpenFlow *Role-Request* messages. Specifically, the scheduler can send a *Role-Request* message on behalf of a controller to a switch. According to OpenFlow [14], a *Role-Reply* message with the controller's current role will be replied from the switch. By examining the *Role-Reply* message, the scheduler could correctly associate a switch with its master controller. Another alternative approach works directly on ONOS controller system. We patch the ONOS system such that each controller, upon receiving an LLDP packet belonging to others, the controller forwards the packet internally to the master controller.

3.4.3 Querying Controller Information

In BLAC, schedulers could gather the workload information from controllers in either an active or a passive mode. In an active mode, the scheduler issues a Remote Procedure Call (RPC)[16] to pull the current workload from the controllers. Comparatively, in a passive mode, the workload information is periodically pushed to schedulers by controllers.

The active mode can be easily implemented for some controller systems that provide monitoring interfaces. For instance, ONOS implements a set of REST APIs that enable the schedulers to retrieve the workload information via HTTP requests. For the controller systems that do not pro-

[16]RPC is a protocol that one program invokes another program in a different address space. In particular, an RPC is initiated by a sender or client, which sends a request message to a remote server. After processing the request, the server returns the response to the client.

vide such interfaces, we can implement and deploy our own load measurement module on each controller, which provides the similar REST APIs for the schedulers.

In the passive mode, the load measurement module periodically delivers the workload information via UDP packets. This is mainly because duplicated data need to be transmitted when multiple schedulers are deployed. Using a unicast TCP connection is not an optimal option since it could result in the waste of network bandwidth and limited system scalability. Instead, we take advantage of the multicast capability of UDP, which allows a stream of data to be sent to multiple destinations with a single transmission operation to reduce network traffic.

3.5 Evaluation

Generally, it is preferable to compare BLAC with dynamic binding based controller architectures, e.g., ElastiCon [80]. However, their implementations are not publicly available. Apart from that, the performance of dynamic binding based architectures rely on switch migrations which is sensitive to a balancing threshold [74]. Generally, to trigger a switch migration process, a balancing threshold is required to identify highly loaded controllers or imbalanced workload distribution in the control plane. However, selecting a suitable threshold can be challenging. For example, if the threshold is too small, it may lead to frequent migration. On the other hand, if the margin is too big, the highly loaded controller can easily get overloaded before the migration takes place. Due to their sensitiveness to parameters, it would be difficult to replicate the best results of these architectures.

Nevertheless, we can still conceptually compare BLAC with ElastiCon. Specifically, ElastiCon can be compared with WRR, as both of them sample controller workload at the cost of extra communication overhead. In fact, ElastiCon incurs a lot more overhead and adds a large amount of complex-

ity. In particular, the switch migration mechanism [80] used in ElastiCon is a complex and time-consuming procedure (above 25 ms [80]), requiring 4 phases and multiple message changes as we discussed in Subsection 2.3.2. To make the switch migration decision, ElastiCon also needs load information that each switch imposes on a controller, which easily multiplies the amount of communication overhead.

In this section, we present the detailed performance evaluation on BLAC. We first illustrate the experimental testbed which is used to measure the response time and throughput of BLAC. Then we compare and analyze the results between BLAC and the static binding controller system supported by ONOS to demonstrate the effectiveness of our architecture.

3.5.1 Experiment Setup

The experimental testbed is built on top of Mininet [167], which is a network emulator that creates an SDN network by emulating links, hosts and switches. It has been widely used for examining the functionalities of newly designed SDN components or systems.

However, Mininet is generally not considered for performance evaluation mainly due to the emulation overhead from packet exchanges and context switch. Specifically, to emulate data plane traffic, actual packets are exchanged across a collection of software switches (e.g. Open vSwitch [222]), which introduces a great amount of traffic in the emulator. Apart from that, Mininet runs the data plane in kernel space while the control plane is executed as a user space process. Whenever a request is sent from switches to controllers, kernel to user space context switch is triggered. Moreover, Mininet is designed to run on a single Linux kernel where the CPU, memory, and bandwidth resources are shared among all the virtual hosts and switches [166]. With limited resources, these overheads significantly limits the maximum flow arrival rate that Mininet can emulate in the data plane. This in turn slows down the number of re-

quests that can be sent to the control plane. Thus, it is necessary to enhance Mininet for performance evaluation.

Inspired by [80], three approaches have been adopted to reduce the emulation overhead. First of all, we modify Open vSwitches to directly inject requests to controllers without actually transmitting packets through the data plane so as to reduce the data plane traffic. This is mainly because our evaluation focuses on controller workload. Emulating data plane traffic with high overhead is unnecessary. Secondly, we disable the transmission of the *Flow-Mod* messages from controllers to switches so as to reduce the overhead of modifying switches' flow tables. It can be simply achieved by dropping the messages in the schedulers. Thirdly, to isolate the switch-controller traffic from the data plane traffic, the controllers and Open vSwitches are running on separate machines. After applying these modifications, the performance evaluation running on top of Mininet becomes reasonable reflection of the real network [80].

All of our experiments are conducted on a testbed running on 5 physical machines with quad-core 3.4 GHz Intel Core i7 3770 processors and 16 GB DDR3. The machines are installed with the latest 1.6 stable version of ONOS. To show the feasibility and efficiency of BLAC, we test a number of applications such as ACL, reactive routing, and reactive forwarding [10]. All of them exhibit similar trend in the results. It is clear that the evaluation results are dependent on the selected application and the specification of the hardware used in our testbed. Nevertheless, since we are interested in the performance gain instead of the absolute values, we only present the results with reactive forwarding for demonstration purposes.

We also measured and compared the system performance with different numbers of schedulers. The results show that there is no significant performance difference regardless of the numbers of schedulers used in BLAC. This finding confirms our belief that schedulers are not the bottlenecks for scalability and network performance.

In the experiments, two performance metrics including *Response Time*

and *Throughput* are evaluated. We measure the response time of a request as the timespan from when a *Packet-In* request is sent by the switch to when the corresponding *Packet-Out* response is received by the switch. It reflects how fast a controller can handle one request. Besides, we define the throughput of a system as the maximum number of packets the system can process within one second. In other words, it measures the controller system's ability to handle a large amount of traffic.

3.5.2 Experimental Results

Response time: We generate requests at different arrival rates and measure the response time with and without the support of BLAC respectively. Specifically, the response time is recorded in a network with seven ONOS controllers at different request arrival rates. When BLAC is used, we test different scheduling algorithms including RANDom (RAND) scheduling, Round-Robin (RR) scheduling, Improved RANDom (IRAND) scheduling, and Weighted Round-Robin (WRR) scheduling. We also measure the response time without BLAC for comparison purpose. In this case, ONOS's default role selection mechanism will decide each controller's role (e.g., master or slave) for switches. Once the master controller is selected, the connection/binding between the switch and the master controller remain fixed.

The results are illustrated in Figure 3.4. They show that without the support of BLAC, the response time for ONOS remains almost constant (around 1 ms) when the arrival rate is less than 25000 pkt/s. It is mainly attributed to the low utilization rate of all controllers. After that, the response time shoots up rapidly in accordance with increasing incoming traffic. In the case of RAND and RR, their performance is indistinguishable. In particular, their response time remains below 5 ms as long as the arrival rate is below 39000 pkt/s. The response time becomes substantially large when the arrival rate is higher than 43000 pkt/s. In comparison, the

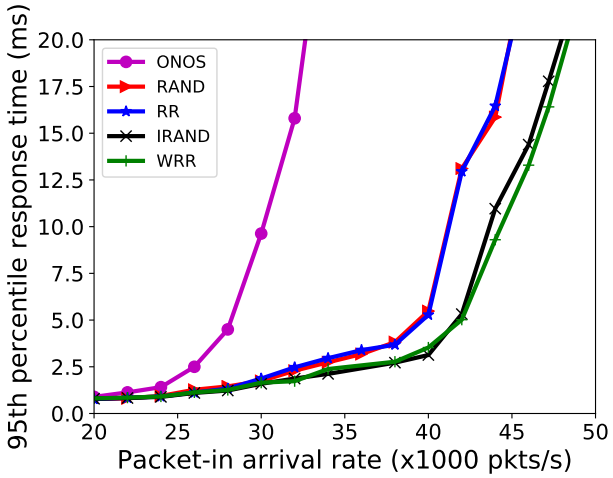


Figure 3.4: System response time comparison. In this figure, ONOS represents ONOS controllers without the support of BLAC. RAND, RR, IRAND, and WRR are results of BLAC equipped with different scheduling algorithms.

response time of IRAND and WRR could still remain around 5 ms even the arrival rate is greater than 40000 pkt/s.

To provide a simple intuition, we could roughly regard each controller as an M/M/1 queue. So long as the service rate of the queue is greater than the arrival rate, the service time (i.e., the response time) could remain at a relatively low value. Once the arrival rate significantly outweighs the service rate, the length of the queue starts growing rapidly, resulting in the increase of the response time. That is the reason why a sharp increase of response time can be spotted in Figure 3.4 when the arrival rate exceeds a certain value no matter which architecture or algorithm we use.

Meanwhile, the response time varies with different architectures and scheduling algorithms due to the way requests are distributed to controllers. In the ONOS case, the inherent nature of static connections could easily lead to uneven workload distribution among the controllers. Thus, some controllers are overloaded while the others remain underutilized,

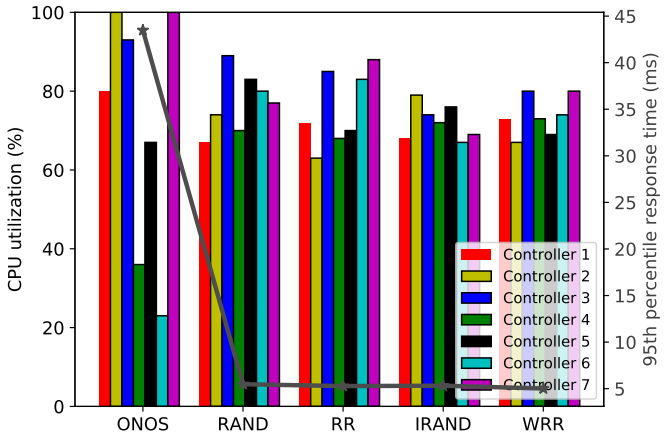


Figure 3.5: Controller workload comparison.

which leads to longer response time, even though the workload for the whole system is relatively low. In comparison, BLAC features binding-less switch-controller association and requests can be evenly distributed among controllers with the help of the introduced scheduling layer. Thus, the workload imbalance issue among controllers can be effectively alleviated, which correspondingly lowers the response time of the whole system.

To verify this, we compare the controller CPU utilization and response time of different approaches at a request arrival rate of 40000 pkt/s. Figure 3.5 shows the system performance benefits from balancing the controller workload. Specifically, load imbalance is severe when ONOS is used alone with fixed switch-controller binding; three controllers are overloaded (i.e., over 90% CPU utilization) while two controllers remains underutilized (i.e., less than 40% CPU utilization), rendering the response time to be over 20 milliseconds. Compared with ONOS, the response time of BLAC is relatively low (approximately 5 ms) for different scheduling algorithms.

In addition, we also notice in Figure 3.4 that IRNAD outperforms RAND. The same trend is also observed between WRR and RR. This is

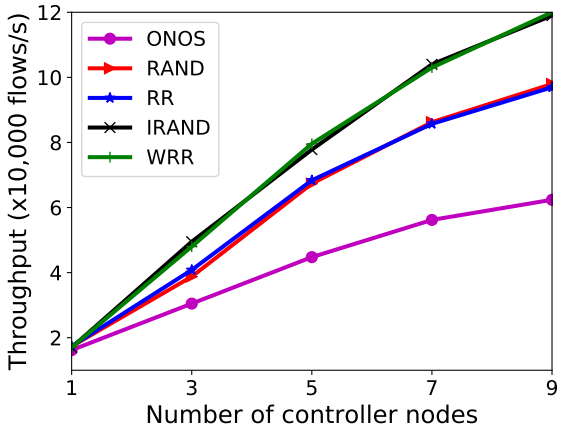


Figure 3.6: System throughput comparison.

mainly because both IRAND and WRR make packet-forwarding decisions considering the workload of the controllers. Another interesting phenomenon is that WRR outperforms IRAND when the *Packet-In* arrival rate is greater than 42000 pkt/s. This observation may be due to the probing overhead which negatively impacts the system performance. Specifically, the number of probes sent to controllers is proportional to the number of packets received from the switches. Therefore, when more packets are received by the scheduler, more probes are sent to the controllers, which introduces communication overhead and may degrade the system performance.

Throughput: We measure the number of requests that ONOS can process without using BLAC within one second and plot it in Figure 3.6. It should be noted that during the experiment, we deployed an odd number of controllers in the cluster due to the cluster management mechanism of ONOS[17].

In general, the throughput of the whole system increases linearly along

[17]ONOS uses the Atomix framework [94] to manage its cluster. Because Atomix is based on the RAFT consensus algorithm, it requires the cluster to be composed by an odd number of controllers [1].

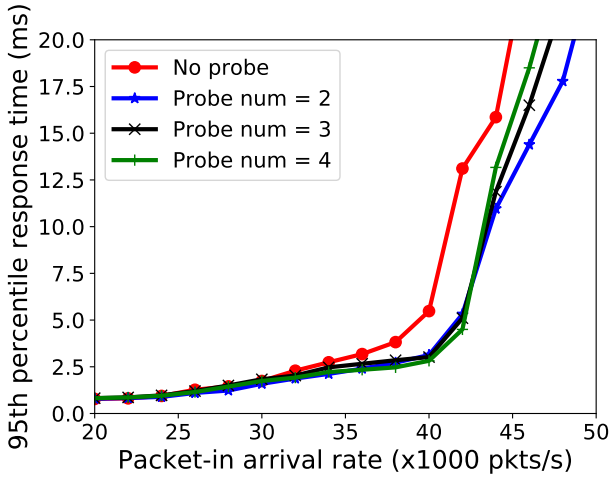


Figure 3.7: System response time with different probe numbers.

with the controller number, as evidenced in Figure 3.6. With the help of BLAC, the whole system achieves a much higher throughput compared with ONOS. Additionally, among different scheduling algorithms, two trends can be observed from Figure 3.6. First of all, as expected, the throughput of RAND and RR has a similar trend, so do IRAND and WRR. Secondly, in terms of the throughput, both IRAND and WRR outperform RAND and RR.

Probe number: Normally, when the request demand is low, changing the number of probes for each request seldom affects the system performance since the network resources are mostly underutilized. However, as the request arrival rate increases, the choice of the probe number starts to show its impact on the response time. To demonstrate the impact of the probe number on network performance, we investigate how the response time changes with the arrival rate when different probe numbers are used. The result is shown in Figure 3.7. We can see that using probes could significantly reduce the response time compared with dispatching requests randomly (i.e., RAND/no probe); Specifically, using probes reduces response time by more than 50% compared with random dispatch-

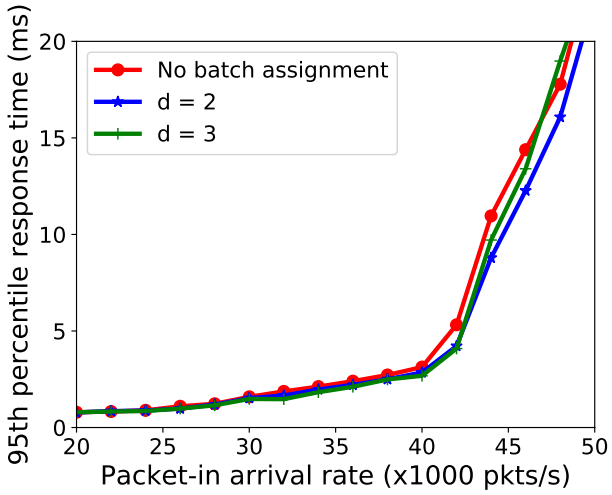


Figure 3.8: System response time with different request-buffering numbers.

ing when the *Packet-In* arrival rate is 42000 pkt/s. Figure 3.7 also demonstrates a sweet spot in the probe number; the system performance barely benefits from more than 2 probes at low request demand. Moreover, excessive probes could negatively impact the system performance with high request demand (e.g., when the *Packet-In* arrival rate is greater than 45000 pkt/s) due to communication overhead introduced by the probing messages. Therefore, we use 2 probes for IRAND throughout our experiments.

Request-buffering number: Figure 3.8 demonstrates that IRAND can further reduce the response time by utilizing batch assignment. The red line represents the power of two choices technique, which dispatches requests without buffering. In the seven-controller network, algorithms with batch assignment outperform the power of two choices technique since the schedulers are more likely to find less loaded controllers by aggregating load information from the probes sent for consecutive requests. However, the response time seldom reduces while buffering over two requests. Furthermore, the system performance even deteriorates when more requests are buffered with high request demand. This can be explained by the fact

that when the number of requests received by the schedulers increases, it is less likely to sample a lightly loaded controller. Meanwhile the response time will be sacrificed when more requests are buffered. In the meanwhile, high request demand is likely to result in traffic congestion, further leading to longer buffering time. Thus, we adopt the request-buffering number $d = 2$ in our implementation throughout our experiments.

3.6 Chapter Summary

The overall goal of this chapter is to tackle the RM problem from the architectural perspective. This goal was fulfilled by developing a novel BindingLess Architecture for distributed Controllers called BLAC which features bindingless switch-controller association.

In particular, BLAC introduces a scheduling layer located between the switches and controllers where multiple lightweight schedulers can be flexibly deployed. The scheduler schedules each individual request to different controllers according to the chosen scheduling algorithm, performing CS in a transparent manner. Without the switch-controller binding constraint, controllers can be flexibly added or removed from the control plane, minimizing the interruption of ongoing traffic. Moreover, BLAC is highly compatible with existing SDN controllers and can be deployed in the network by simply assigning a unique anycast controller IP address to all switches. A prototype of BLAC has been built and experiments have been performed on the prototype. The results showed that BLAC could significantly reduce the response time and improve the system throughput compared with the static binding controller system. With the support of BLAC, the CPP is addressed through the algorithm design perspective in the next chapter.

Chapter 4

A Scalable SDN Control Plane: High Utilization Comes with Low Latency

4.1 Introduction

With the wide adoption of distributed controller architectures, the Controller Placement Problem (CPP) becomes a critical research issue. Although over-provisioning controllers for an expected peak traffic load can be considered as a possible strategy to solve the 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[18]. As defined by *Heller et al.* [118], 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 [118, 277, 279] and improving resource utilization [271]. For instance, a suitable number of controllers must be placed in proximity to demanding switches in ac-

[18] According to network traffic measurements [44], the peak-to-median ratio of request arrival rate can be up to 2 orders of magnitude.

cordance with the dynamic fluctuations of traffic workload. This is critical to the overall network reliability and performance, especially for *wide-area networks* [165, 272, 279].

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 thesis, 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 the 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 [74] and our simulation study in Subsection 4.5.1. Such a controller will become sensitive to its workload changes. A slight increase in request arrival rate can easily overload the controller, resulting in high variation in response time.

Motivated by the above understanding, *Yao et al.* [291] 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 [271, 272, 279]. More details are provided in Section 2.4.

Although considerable efforts have been made to address the CPP, the importance of the CSP has always been underestimated and sometimes ignored [118, 215]. Only a few recent works have explicitly quantified the impact of the CSP on the performance of the control plane [279]. Moreover, most of the existing approaches for the CPP are designed for binding-based controller architectures (e.g., ONOS [45] and Onix [158]) that require every switch to always contact its bound controller [138], restricting the opportunity to properly distribute workload across all controllers. Although it can be alleviated by the switch migration mechanism [80] to reassociate 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 enable fine-grained workload distribution, we investigate the CPP based on BLAC which we introduced in Chapter 3. BLAC introduces bindingless switch-controller association so that requests from one switch can be flexibly processed by different controllers. Key reasons for choosing BLAC will be elaborated in Subsection 4.2.1.

4.1.1 Chapter Goals

In fulfillment of Objective 2 stated in Section 1.3, a new Controller Placement and Scheduling Problem (CPSP) is defined and solved in this chapter, emphasizing on the necessity and importance of tackling both the CPP and the CSP simultaneously in a coherent framework. Precisely, this chapter aims to address the following objectives:

- Design a model to precisely measure the influence of the CSP on network performance in a realistic manner;
- Propose a new problem formulation to jointly tackle both the CPP and the CSP within a coherent framework, aiming to optimize network performance;
- Develop a CS algorithm to achieve a good balance between scheduling performance and problem scalability;
- Develop a CP algorithm to effectively solve the CPP;
- Improve the CP algorithm so that it can scale to large networks;
- Develop a mechanism to enhance the robustness of the CP algorithm in terms of handling unexpected traffic surges;
- Compare the performance of our proposed CS algorithm with existing popular approaches, e.g., weighted round-robin [88, 101].

- Compare the performance of our proposed CP algorithms with existing popular approaches (e.g., k-center [118]) and the state-of-the-art approaches (e.g., MSPA [279]).

4.1.2 Chapter Organization

The rest of this chapter is organized as follows. We formulate the CPSP in Section 4.2. Since the CPSP is comprised of the CSP and the CPP, we first study several solutions for the CSP in Section 4.3. Given the CSP solution, several effective approaches are developed in Section 4.4 for addressing the CPP. Section 4.5 demonstrates the simulation results. Section 4.6 concludes this chapter.

4.2 Problem Formulation

In this section, we will present a model of the network environment and formulate the CPSP. The key notations have been summarized in Table 4.1 for the ease of reference.

4.2.1 Using BLAC for the Controller Placement Problem

In this research, we evaluate the CPP using BLAC for two main reasons:

First, BLAC enables dynamic and transparent controller placement without incurring time-consuming switch migration. Other distributed controller architectures such as ONOS [45] and ElastiCon [80] support either static or dynamic switch-controller binding. Therefore, when the controllers are relocated, switches need to migrate from previous controllers to new ones. This process can be time-consuming and even introduce network disruption. On the contrary, BLAC introduces bindingless switch-controller association. In BLAC, switches are connected directly to the scheduling plane. Thus, controllers can be flexibly and transparently relocated without interrupting the switch connections.

Table 4.1: Mathematical notations for the CPSP

Notation	Definition
V	A finite set of network nodes
λ_v	Request generated rate from node v
α_v	Controller processing capacity of node v
β	Decay factor of a controller node
x_v	Whether a controller is allocated at node v
x	A binary vector representing a CPP solution
$D(v, v')$	One-way propagation latency between v and v'
$P_{v,v'}$	Probability of node v sending requests to v'
M	Total number of network nodes
N	Total number of selected controllers ($N = \|x\|_1$)
γ	Synchronization factor
ω	Clustering factor
t_{th}	Response time threshold
ϵ	the sum of transmission latency, switch processing latency, and scheduler processing latency

Second, BLAC enables fine-grained controller load control. Due to the switch-controller binding constraint, the majority of distributed controller architectures can only manipulate the controller workload at switch level. In other words, requests from one switch can only be processed by one controller. However, BLAC can dispatch requests from one switch to different controllers, achieving flow-level load control. Thus, the architecture improves the flexibility of distributing the workload among controllers.

Apart from providing flexible and explicit control over the controller workload distribution, the complete separation between the control plane and the data plane also enables us to formulate the CPP without the switch-controller binding constraint. With the help of BLAC, the CPP is effectively formulated in Subsection 4.2.3.

4.2.2 Network Environment

Note that in a small-scale network (e.g., a local data center hosted by universities on the premises of the organizations to serve local users), the propagation latency can be safely ignored (below 1ms) [108]. Therefore, the network response time mainly depends on the processing time of the controller. Thus, optimizing the CP barely affects the network performance. In comparison, for a large-scale network which may span large geographic areas, the propagation latency becomes significant. According to existing studies, the propagation latency can vary from 10 ms to 200 ms [68, 115, 172]. Obviously, inappropriate placing controllers in remote locations will inevitably increase the network response time. Thus, as we mentioned in Section 4.1, CP is critical to the overall network performance, especially for wide-area networks.

Based on the above understanding, we considered a large enterprise global communication backbone network $G = \langle V, E, D \rangle$ 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[19].

With the help of BLAC, controllers in the control plane can be flexibly deployed at any node in the network. 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 [270, 301]. We hence use α_v to measure the maximum number of requests processable by the controller located at node v within

[19]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.

one second. A full solution to the CPP can subsequently be represented as a binary vector x where 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 selected nodes where controllers are deployed can be easily identified and denoted as $\widetilde{V}$.

Unlike [118, 165, 291], we assume that each switch can flexibly dispatch requests to any deployed controllers with the help of a locally installed scheduler as shown in Figure 4.1. This assumption can be easily fulfilled by our proposed controller architecture A BindingLess Architecture for Distributed SDN Controllers (BLAC). For instance, a scheduler can be installed on each switch as a light-weight Network Function Virtualization (NFV) component [204] to dispatch requests as shown in Figure 4.1. 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 Internet Service Provider (ISP) the flexibility about where the schedulers can/should be placed based on their understanding of the network dynamics.

In BLAC, since a switch in theory can ask multiple distributed controllers to handle its requests, maintaining consistency across all controllers is critical for controllers to correctly handle the requests. In general, there are two types of consistency mechanisms: strong consistency and eventual consistency. Strong consistency guarantees that each controller always has the up-to-date network view. On the other hand, eventual consistency provides a weak form of consistency. Data modifications on one controller will be eventually propagated to all controllers. Thus, a controller may read outdated network view at some point. It should be noted that the degree of staleness actually depends on the location of the controller. In particular, a controller always has up-to-date informa-

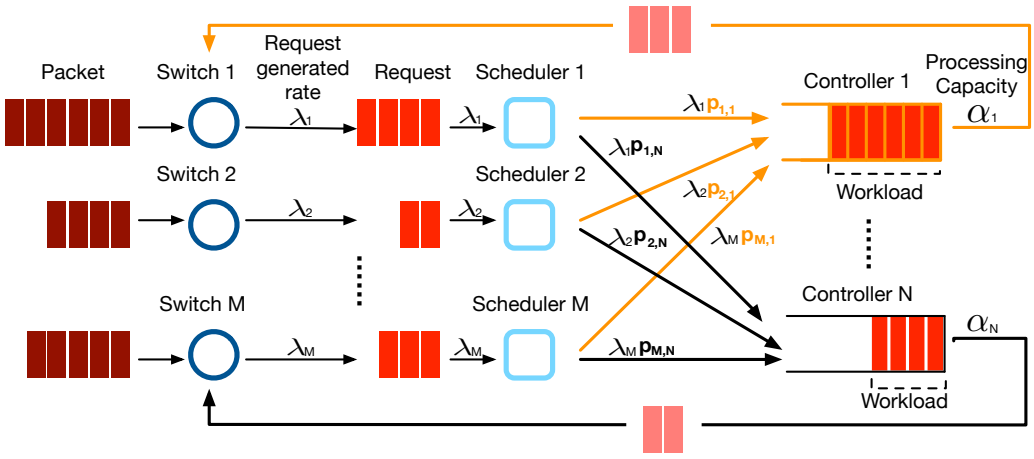


Figure 4.1: Request processing procedure.

tion about its nearby network, i.e., a network cluster (or a sub-network) where the controller belongs to. For the part of network that is far from the controller, more time is required for synchronization to take effect. But the topology information held by the controller about the remote network remains usable to a large extent. Due to the slow change of the network topology, we consider weak/eventual consistency is sufficient for our approach. In particular, since the network we considered is a large enterprise global communication backbone, frequent changes in network topology are unlikely to happen [163, 217]. Apart from that, since traffic generated by each node is aggregated from local networking devices, the traffic variation is supposed to be smooth [2]. Moreover, our algorithm design in Section 4.4 also encourages switches to send requests to nearby controllers for processing while carefully preventing any switch to frequently contact long-distant controllers. Thus, similar to many existing studies [170, 236], weak/eventual consistency is sufficient.

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 the CPP, we must find the corresponding solution to the CSP as well. Given the job of scheduling R requests within a specific time period, the complexity of the 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 Figure 4.1, 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:

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

4.2.3 Problem Formulation

As showed in Figure 4.1, we can model each controller as an independent M/M/1 queue [191, 280]. According to Little's Law, the long-term average number of customers in a stationary system can be calculated as the product of the customer 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 system processing rate and the customer arrival rate. Then the average customer waiting time can be calculated as the inverse of the difference between the system processing rate and the customer arrival rate [106]. Similarly, considering each request as a customer, the average processing latency of the controller at node v' can be calculated

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

To maintain a consistent network view, controllers need to synchronize network events with each others. Note that in an SDN network, 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 Subsection 4.2.2, frequent changes in network topology are unlikely to happen [163, 217]. 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 $\mathcal{O}(N^2)$. 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.

$$F_{syn} = \gamma N^2 \tag{4.1}$$

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:

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

Generally speaking, the response time consists of five components: transmission latency, propagation latency, switch processing latency, scheduler processing latency, and controller processing latency. In a high-bandwidth network (e.g., Sprint Network [19], a backbone network considered in this chapter), the transmission latency is trivial. Apart from that, with recent advancement of hardware technology, high-performance switches have become prevalent (e.g., NoviSwitch [8] and EZchip [4]), resulting in negligible switch processing latency [77]. Note that schedulers

are solely responsible for request dispatching according to the predetermined probability distribution P, incurring negligible computation overhead (simply toss a coin for every request to be dispatched). Thus, the scheduler processing latency can be safely ignored. Nonetheless, to make the model 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:

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

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

$$t_{avg} = \frac{\sum\limits_{v' \in \widetilde{V}} t_{v'} (\sum\limits_{v \in V} \lambda_v P_{v,v'})}{\sum\limits_{v \in V} \lambda_v} \tag{4.2}$$

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

$$u = \frac{\sum\limits_{v \in V} \lambda_v}{\sum\limits_{v' \in \widetilde{V}} \alpha_{v'}} \tag{4.3}$$

In consideration of realistic requirements from network operators [87, 272, 294, 302], we formulate our problem with the main goal of minimizing the network operation cost, which is realized through maximizing the control plane utilization. Accordingly, the 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 (4.2). This requirement gives rise to the constrained optimization problem below:

$$\max_{x,P} \quad u(x) \tag{4.4}$$

$$\text{s.t.} \quad t_{avg}(x,P) \leq t_{th} \tag{4.5}$$

$$\sum_{v \in V} \lambda_v P_{v,v'} \leq \beta(\alpha_{v'} - F_{syn}),\ \forall v' \in \widetilde{V} \tag{4.6}$$

$$\sum_{v' \in \widetilde{V}} P_{v,v'} = 1,\ \forall v \in V \tag{4.7}$$

$$0 \leq P_{v,v'} \leq 1,\ \forall v \in V, \forall v' \in \widetilde{V} \tag{4.8}$$

Since the improvement of control plane utilization should not come at the expense of network performance, (4.5) guarantees that average response time should remain below a certain 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 [196]. According to [20], Sprint's committed network round trip backbone delay is 55ms within North America while the delay increases up to 105 ms within Asia. (4.6) 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 [281]. Finally, both (4.7) and (4.8) 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 the CPSP (4.4)-(4.8) produces two sub-problems. The first one is to identify the number and locations of controllers (i.e., the CPP) while the second one requires us to determine the workload distribution among deployed controllers (i.e., the CSP). Accordingly, our solution

to the CPSP is made up of two inter-dependent parts: one for the CPP and the other one for the CSP. In consideration of the fact that the quality of a solution to the CPP depends on the corresponding solution to the CSP, we will first investigate different approaches for solving the CSP in Section 4.3. A study on various methods for solving the CPP will be carried out subsequently in Section 4.4.

4.3 Controller Scheduling Algorithms

In this section, we assume that a solution x to the CPP is given in advance. In other words, the CSP is solved based on all deployed controllers. Our task for the CSP is to effectively schedule the requests generated by all switches over the deployed controllers to reduce the response time so as to satisfy (4.5). Accordingly, the CSP can be formulated as:

$$\begin{aligned} \underset{P}{\text{solve}} \quad & t_{avg}(P) \leq t_{th} \\ \text{s.t.} \quad & (4.6), (4.7), (4.8) \end{aligned} \tag{4.9}$$

In order to achieve low average request response time, a switch will send a large portion of its requests to nearest controllers rather than remote controllers as verified by our simulation studies in Subsection 4.5.2. To solve (4.9), several approaches will be studied, ranging from simple heuristics to a newly proposed approach that periodically calculates the suitable distribution probability P over all controllers through a GD based technique. We will also analyze the pros and cons of adopting each approach.

4.3.1 Weighted Round-robin Scheduling

Speaking of heuristics, random and round-robin are two widely used scheduling methods [146, 226]. 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 [88, 101], $P_{v,v'}$ can be made proportional to the controller capacity, as shown below:

$$P_{v,v'} \propto (\alpha_{v'} - F_{syn}), \forall v \in V \tag{4.10}$$

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 Subsection 4.5.2.

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

$$P_{v,v'} \propto \frac{\alpha_{v'} - F_{syn}}{D(v, v')} \tag{4.11}$$

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 may follow different probabilities of dispatching its requests to the same controller.

4.3.2 Gradient-descent-based Scheduling

According to (4.9), t_{avg} can be reduced through adjusting P. Note that for any possible solution P, it must satisfy constraints (4.6)-(4.8). To handle the constraints, we adopt the most commonly used penalty function

Algorithm 1 Gradient-descent-based Algorithm for the CSP

1: Initialize the distribution matrix P
2: **while** $t_{avg}(P) \geq t_{th}$ **do**
3: Update P using (4.14)
4: **end while**
5: **return** P

method [69, 232] due to its simplicity, ease of implementation, and its general applicability to constrained problems with equality and inequality constraints [31]. Specifically, (4.9) is transformed into a constraint-free optimization problem by introducing the term in (4.12) to penalize any constraint violation [232].

$$\begin{aligned} F_{con}(P) = \mu_1 \sum_{v' \in \widetilde{V}} \min\Big(0, \beta(\alpha_{v'} - F_{syn}) - \sum_{v \in V} \lambda_v P_{v,v'}\Big) \\ + \mu_2 \sum_{v \in V} \Big|1 - \sum_{v' \in \widetilde{V}} P_{v,v'}\Big| + \mu_3 \sum_{v \in V} \sum_{v' \in \widetilde{V}} \min\Big(0, P_{v,v'}\Big) \end{aligned} \tag{4.12}$$

where μ_1, μ_2, and μ_3 are penalty parameters.

Consequently, (4.9) becomes:

$$\min_P F_{csp}(P) = \min_P \Big(t_{avg}(P) + F_{con}(P)\Big) \tag{4.13}$$

To solve this optimization problem, we develop a GD-based scheduling algorithm as described in Algorithm 1.

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. Simulation evaluation on the convergence of our GD-based approach is reported in Subsection 4.5.2.

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

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

$$P_{i+1} = P_i - \frac{\partial F_{csp}}{\partial P} \cdot l \tag{4.14}$$

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

The whole process repeats until the stopping criterion is reached (i.e., $t_{avg}(P) \leq t_{th}$ or the maximum number of iterations is reached). It is important to note that Algorithm 1 can be executed efficiently in practice with the help of advanced learning/optimization tools such as Theano [22] or TensorFlow [21]. Particularly, Theano is used in our implementation. During our simulation in Subsection 4.5.2, we noticed that our algorithm converges quickly within 10 iterations (i.e., around one second[20]). With the support of high-performance computers in industry, the convergence time can be further reduced. Moreover, one second of GD running time 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, any delay caused by rerunning GD 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 in (4.9) is optimized, it will guide request distribution for all switches.

4.4 Controller Placement Algorithms

Facilitated by a method for solving the CSP, we can now address the CPP for the purpose of improving controller utilization. As we explained previ-

[20]The algorithm is run on a physical machine with quad-core 3.4 GHz Intel Core i7 3770 processors and 16 GB DDR3

ously, it is unfeasible to find the optimal solution to the CPP in a large network. Existing approaches attempted to simplify the CPP [118, 134, 291] by assuming that either the number of required controllers is predetermined or all controllers have identical capacities so that the required controller number can be easily calculated. In this thesis, 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.

4.4.1 K-center

K-center is currently one of the most well-known CPP strategies [118] which aims to find the k locations of 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 product of β and the total capacity of selected controllers must be larger than the total request arrival rate. Therefore, the modified K-center can be formulated as follows:

$$\min_{x} \quad \max_{v \in V, x_{v'}=1} \{D(v, v')\} \tag{4.15}$$

$$\text{s.t.} \quad \sum_{v \in V} \beta(\alpha_v - F_{syn}) x_v \geq \sum_{v \in V} \lambda_v \tag{4.16}$$

$$\|x\|_1 = k \tag{4.17}$$

This modified K-center approach will be further compared with other methods in Subsections 4.5.3 and 4.5.4. Obviously K-center does not explicitly handle the CSP since it purely selects the closer controllers without considering the request assignment among them. Given two controllers with similar propagation latencies but significantly different capacities, K-center will choose the controller with lower propagation latency despite its low capacity, resulting in increased average network response time.

Furthermore, K-center overlooks the control plane utilization and may select more controllers, potentially incurring additional network operation costs/expenses as evidenced by our simulation results in Subsection 4.5.3.

4.4.2 Genetic Algorithm

Recently, evolutionary computation (EC) techniques have been widely exploited to effectively solve various NP-hard problems [30, 102, 120]. The promising results reported in the literature inspire us to tackle the CPP (4.4) through an EC method. Among all EC techniques, we prefer Genetic Algorithm (GA) for two main reasons. (1) The solution x to the 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 is particularly suitable for solving highly constrained problems [30, 102, 120, 197].

In this study, we follow the GA framework introduced in [256]. We first transform the CPP objective (4.4)-(4.8) into a constraint-free optimization problem:

$$\max_{x,P} F_{cpp}(x) = \max_{x,P} \Big(u(x) - \hat{F}_{con}(x, P) \Big) \tag{4.18}$$

where

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

In line with the CPP objective in (4.18), the fitness value of any solution x can be calculated as $F_{cpp}(x)$. Given a candidate solution x to the CPP, the control plane utilization $u(x)$ can be directly obtained via (4.3). Additionally, the solution P to (4.9) will be optimized through the GD-based approach developed in Subsection 4.3.2. Consequently, $F_{cpp}(x)$ can be easily calculated.

Although the combined use of GA and GD has been proposed to developed memetic algorithms in previous studies [30, 102, 120], our proposed

Algorithm 2 Genetic Algorithm with Gradient Descent

1: Initialize the population
2: Evaluate the population via (4.18) and Algorithm 1
3: **for** generation $= 1, 2, ...$ **do**
4: Generate individuals via Crossover and Mutation
5: Evaluate new individuals via (4.18) and Algorithm 1
6: Select individuals to form a new population
7: **end for**
8: **return** The best individual x and its corresponding probability P

algorithm is different. In particular, memetic algorithms combine evolutionary algorithm (e.g., GA) for new solution exploration and local search (e.g., greedy strategy) for existing solution improvement. In comparison, GD in our algorithm is purely for fitness evaluation instead of improving any existing solutions.

After evaluating the fitness of all solutions in the current population, a new population is created by performing genetic operators (e.g., crossover, mutation, and selection) on the current population. The whole process repeats over multiple generations until the maximum number of generations is reached, as summarized in Algorithm 2. In terms of scalability, the fitness evaluation of all candidate solutions in a GA population can be performed in parallel.

4.4.3 Clustering-based Genetic Algorithm

In order to effectively solve the 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. Moreover, with the exponential expansion of the search space, GA takes extremely long time to discover good solutions to the CPP [256].

To reduce the complexity of the search space in the CPP, we adopt

the famous divide-and-conquer strategy [71] and develop the Clustering-based Genetic Algorithm (CGA). 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 the 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, schedulers are unlikely to be overwhelmed by requests generated within their respective sub-networks.

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

$$\min_{G_1,...,G_k} \sum_{i=1}^{k} \sum_{v \in V_i} D(v, c_i) \tag{4.19}$$

$$\text{s.t.} \quad \bigcup_{i=1}^{k} V_i = V, \forall i \tag{4.20}$$

$$V_i \bigcap V_j = \phi, \forall i \neq j \tag{4.21}$$

where $c_i \in C$ is the center of sub-network G_i. (4.20) ensures that all nodes are divided into sub-networks. (4.21) guarantees that every node belongs to only one sub-network. After partitioning, the requests generated within a sub-network will be processed most of the time by controllers deployed in the same sub-network.

Although the network partitioning problem is essentially a clustering problem and K-means is undoubtedly the most widely used clustering algorithm, it may not be suitable for the network partitioning task since the algorithm is highly sensitive to the selection of initial center points [55]. Inspired by the previous research [279], the Clustering-Based Network Par-

Algorithm 3 Clustering-Based Network Partition Algorithm (CNPA)

1: Initialize the center set $C = \varnothing$
2: Find the initial center c_1 via (4.22) and add c_1 to C, i.e., $C = \{c_1\}$.
3: **while** $|C| \neq k$ **do**
4: Find a new center c_i via (4.23) and add c_i to C.
5: Assign each network node $v \in V$ to its closest center in C.
6: Update C via (4.22).
7: **end while**
8: **return** k sub-networks and their centers C.

tition Algorithm (CNPA) is adopted in this section to tackle the network partitioning problem. Compared to K-means, CNPA does not introduce the randomness in its algorithm since no randomly selected centers are required. As demonstrated in Algorithm 3, CNPA first initializes an empty center set denoted by C. 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 (4.22) as the initial center c which will be added into C:

$$c = \underset{v \in V_i}{\operatorname{argmin}} \sum_{v' \in V_i} D(v, v') \tag{4.22}$$

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.

$$c' = \underset{v \in V}{\operatorname{argmax}} \sum_{c \in C} D(v, c) \tag{4.23}$$

After determining the first two centers according to (4.22) and (4.23) 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 (4.22) and updated in C. After that, if the number of sub-networks is less

than k, a new center will be selected using (4.23). The whole process repeats until k sub-networks have been obtained.

In order to apply CNPA, we should first of all decide the desired value for k. Intuitively, increasing k will significantly reduce the search space for GA. However, more sub-networks demand for more controllers since at least one controller needs to be allocated within each sub-network. This will affect the overall control plane utilization and introduce high network operation costs/expenses especially when the request arrival rate is low.

To strike a balance between computation complexity and control plane utilization, a heuristic based on our preliminary studies will be utilized to decide the k value according to the current network condition. Specifically, given the request arrival rate, we roughly estimate the required number of controllers using the following formula:

$$N_c = \frac{\sum\limits_{v \in V} \lambda_v}{\frac{\sum\limits_{v' \in V} \alpha_{v'}}{M}}$$

Then k can be decided as follows:

$$k = \begin{cases} 2\,\omega, \text{ if } \frac{N_c}{M} \leq 0.5 \\ 4\,\omega, \text{ otherwise} \end{cases} \tag{4.24}$$

where ω is the clustering factor, a hyper-parameter that can be flexibly adjusted based on the network size.

The whole process of CGA is summarized in Algorithm 4. 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, 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

Algorithm 4 Clustering-based Genetic Algorithm (CGA)

1: Calculate the number of sub-networks k
2: Run CNPA (Algorithm 3) to obtain k sub-networks $G_1, ..., G_k$
3: **for** $i = 1, ..., k$ **do**
4: Run GA (Algorithm 2) within sub-network G_i to obtain the controller placement solution x_i and the probability P_i
5: **end for**
6: **return** $(x_1, P_1), ..., (x_k, P_k)$

the latest information about its sub-network which is sufficient to make routing decisions within its sub-network.

4.4.4 Clustering-based Genetic Algorithm with Cooperative Clusters

Although CGA can effectively reduce the search space of GA and prevent long-distance switch-controller communication, it also forbids workload sharing across different sub-networks. When the traffic within a sub-network suddenly and substantially increases, existing controllers in the sub-network cannot handle the extra workload. This may significantly slow down the control plane or even lead to breakdown of the whole network [286]. Hence, in the event of high workload in a sub-network, it is a common practice to seek help from controllers in neighboring sub-networks.

Specifically to cope with the traffic burst, we can adopt either a proactive or reactive approach. For the proactive approach, a fine-tuned traffic prediction model is required to accurately predict the future traffic trend based on historical data, posing significant difficulties of using this approach in practice. In particular, the successful use of a proactive approach depends on multiple factors, including the selection of a suitable predictor, the predictor training, and the use of the predictor in a separate

planning loop. Apart from that, the proactive approach requires us to develop complicated techniques for planing and uncertainty handling. Since the real world environment is complex, even the "best" prediction model cannot guarantee 100% accurate prediction. Thus, when unpredictable events happen, e.g., unexpected traffic surge, how to handle uncertainties to guarantee the network performance needs to be considered. Obviously, this is beyond the scope of the CPP in this thesis. Instead, a simple-yet-effective reactive offloading scheme is developed here.

CGA with Cooperative Clusters (CGA-CC), an extension of CGA, is proposed to improve the flexibility and robustness of CGA by facilitating workload sharing between adjacent sub-networks. For this purpose, we must decide which adjacent controllers should process these 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 section, we take advantage of the simple-yet-effective piggybacking mechanism. Inspired by [95], 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 described in Algorithm 1. To avoid potentially overloading any controllers, we only use the remaining capacities of these controllers as the input to the scheduling algorithm. The full process

Algorithm 5 Clustering-based Genetic Algorithm with Cooperative Clusters (CGA-CC)

1: Run CGA to obtain the controller sets $x_1, ..., x_k$ of k sub-networks $G_1, ..., G_k$
2: **for** $j = 1, ..., k$ **do**
3: **if** Traffic $\geq 90\% \times$ total capacity in G_j **then**
4: Identify neighboring candidate controllers using the piggybacking mechanism
5: **while** Traffic $\geq 90\% \times$ total capacity in G_j **do**
6: Find the closest controller within the remaining candidate controllers
7: Add the controller into the deployed controller set of G_j
8: **end while**
9: Run GD (Algorithm 1) to calculate P based on the deployed controller set of G_j
10: **end if**
11: **end for**

of CGA-CC is summarized in Algorithm 5.

Compared to a proactive approach, our approach does not rely on any prediction models and the overall workload-offloading process is simple yet effective. Apart from that, it can effectively avoid the network performance deterioration caused by controller overloading through a simple threshold mechanism. Specifically, with the threshold mechanism, we can react quickly by offloading traffic to neighboring networks before the controllers become overloaded. 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 network updates from the sub-network under

question more frequently than usual. The frequent network updates help the controller maintain the latest network view about its neighboring sub-network, guaranteeing the correctness of routing decisions made for packets from neighboring sub-networks. Due to this reason, CGA-CC eliminates the necessity for strong network-wide controller consistency and incurs no extra synchronization cost.

4.5 Evaluation

Given that no physical WAN (the central focus of this chapter) testbed is currently accessible to us, simulation is adopted as the main evaluation tool due to its flexibility and feasibility. From the flexibility point of view, simulation enables us to study the behavior of our algorithm under different network settings in terms of network sizes, controller capacities, and request demands. From the feasibility perspective, real-world evaluation requires not only substantial system implementations but also hardware devices which are usually not accessible to many researchers.

With the help of simulation, this section presents the performance evaluation of our proposed algorithms. In particular, we first introduce the simulation setting. The effectiveness of the GD-based scheduling approach is demonstrated in Subsection 4.5.2 under two network topologies that cover different geographical areas. After that, we compare the performance of GA, CGA and CGA-CC with the K-center approach (mentioned in Subsection 4.4.1) in Subsections 4.5.3 to 4.5.5. We also compare CGA-CC with MSPA (introduced and discussed in Subsection 2.4.2.3) in Subsection 4.5.6. The simulation setup[21] and our algorithm implementation[22] have been uploaded to Github.

Note that many K-center algorithms (e.g., the original K-center [118], the capacitated K-center [291], and the improved K-center [143]) have been

[21]https://github.com/VictoriaWong/sdn-simulator
[22]https://github.com/VictoriaWong/SDN-Controller-Placement

proposed previously. Our empirical studies show that the K-center algorithm described in Subsection 4.4.1 outperforms the original K-center, the capacitated K-center and the improved K-center. Therefore we will only report comparison results regarding our K-center algorithm, GA, CGA and CGA-CC.

4.5.1 Evaluation Setup

To provide the evaluation setup, we start with describing how the simulator simulates the network behaviors (e.g., requests generated from switches and processed by the controllers) and verifying the traffic generator with real-world traffic traces. After that, topology information used during our evaluation and detailed parameter settings are provided.

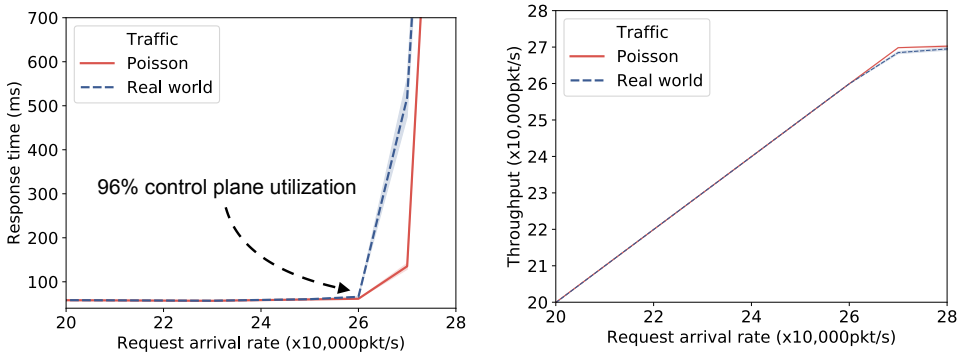
Simulator: The simulation starts with an idle network (all controllers are idle and no packets are transmitted in the network). Immediately after the simulation starts, the requests are generated randomly based on a predetermined fixed request arrival rate for each network node. The simulation runs for 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.



(a) Response time (b) Throughput

Figure 4.2: Performance comparison with real-world traffic and traffic generated by Poisson distribution.

(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[21].

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 4.2. 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 [44].

As demonstrated in Figure 4.2, the corresponding throughput achieved is indistinguishable regardless of whether real-world traffic or artificial traffic is used in the simulation. In addition, their response time is indistinguishable when the control plane utilization is below 96%. Beyond that, significant difference can be spotted. In particular, when the arrival rate reaches 270k 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 (4.6) presented in Subsection 4.2.3 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.

Topology: All simulations will be conducted using the topology information provided by Sprint [19] including the propagation latency, network links, and the numbers of switch nodes. The propagation latency between any two nodes is calculated by Dijkstra's algorithm [79]. 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 [118].

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 220k pkt/s to 620k pkt/s in the Asia Sprint Network, see Figure 4.5. The propagation latency between any two nodes is calculated by Dijkstra's algorithm [79]. A set of heterogeneous controllers with capacities ranging from 60k to 120k 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 [291], C2 is generally regarded as the most significant part of the control plane workload. Following our problem formulation in Section 4.2 , the impact of C2 is taken care of by the synchronization cost in (4.1).

Parameter setting: During our simulation, we tried different parameter

settings for both GA and GD. For example, a range of different population sizes for GA have been tested, ranging 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. 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 the CSP, GD is performed for 10 iterations for a good balance between the algorithm performance and efficiency as evidenced in Subsection 4.5.2. The decay factor β is set to 0.85 and the synchronization factor γ is set to 0.1 following existing studies [280, 281].

4.5.2 Effectiveness of the GD-based Scheduling Approach for the CSP

We first measure the convergent rate of our proposed GD-based approach. Then we compare the performance of the 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 60k, 90k, and 120k pkt/s respectively are deployed using K-center.

4.5.2.1 The convergence of GD

In order to measure the convergent performance, we run the GD-based scheduling algorithm for 20 iterations on a physical machine with quad-core 3.4 GHz Intel Core i7 3770 processors and 16 GB DDR3 using different network settings (e.g., network topologies, request arrival rates and so

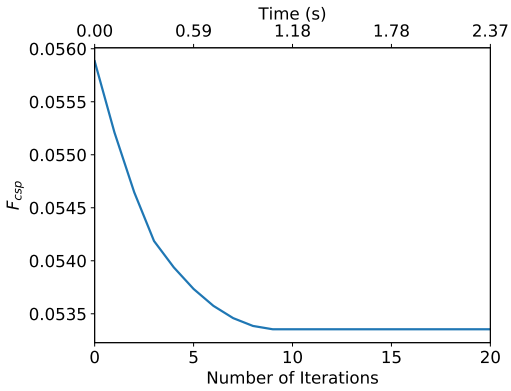


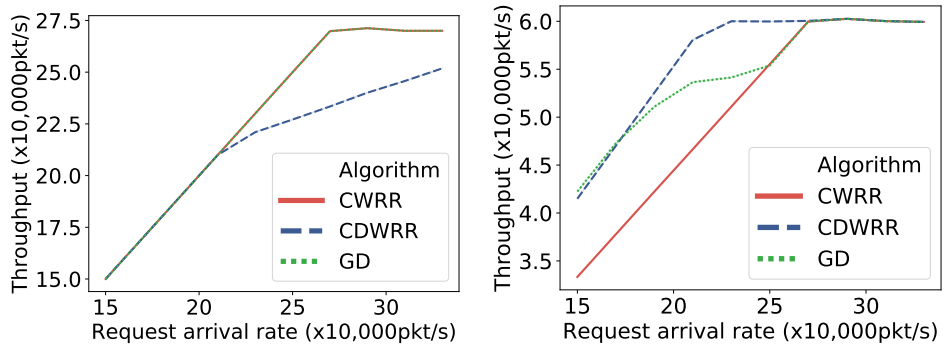
Figure 4.3: The convergence rate of the GD-based approach for controller scheduling problem.

on). Figure 4.3 depicts the change of F_{csp} in (4.13) across iterations. It was measured in the Asia Sprint Network topology with a request arrival rate at 110 k pkt/t. We can clearly observe from Figure 4.3 that our algorithm can quickly converge within 10 iterations of gradient-based improvement to P and spend around 1 second. Furthermore, with the support of high-performance computers, the convergence time can be further reduced.

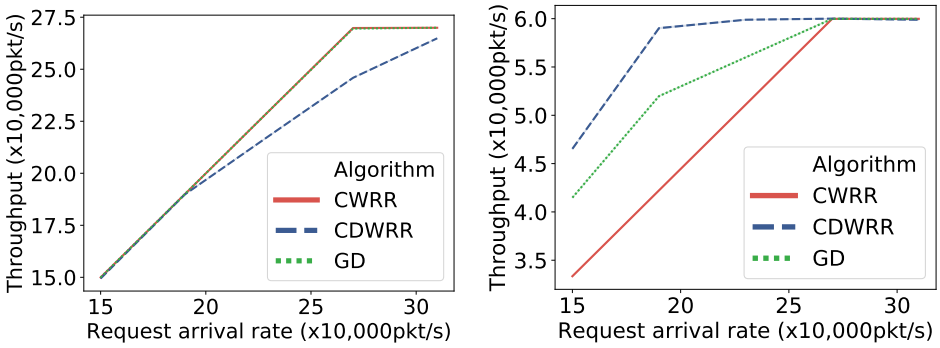
4.5.2.2 Overall Network Performance

From Figure 4.4 and Table 4.2, we can clearly notice that on all tested network topologies, GD achieved the lowest response time and highest throughput among all algorithms.

Specifically, Table 4.2 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 190k pkt/s. It is mainly attributed to the low utilization of all controllers. We also notice that at this stage, 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.



(a) Overall performance (Europe) (b) Individual performance (Europe)



(c) Overall performance (Asia) (d) Individual performance (Asia)

Figure 4.4: 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.

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

Table 4.2: Average response time (ms) in Europe and Asia Sprint Network.

Arrival rate (x10k pkt/s)		15	19	23	27
Figure 4.4(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
Figure 4.4(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
Figure 4.4(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
Figure 4.4(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

ity, which effectively prevents overloading any controller at an early stage. GD guarantees that all controllers are not overloaded by fulfilling the constraint in (4.6) when solving the optimization problem. Thus, the response time of both GD and CWRR remains low until the arrival rate increases to 270k pkt/s. On the other hand, when CDWRR is used, more requests will be sent to nearby controllers, quickly overloading them.

4.5.2.3 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 60k pkt/s. It can be seen from Figure 4.4(b) that at the arrival rate of 230k pkt/s, the throughput of CDWRR almost reaches 60k pkt/s, implying that the controller is fully-loaded. Simultaneously, a dramatical growth in response time of CDWRR can also be noticed from Table 4.2. Comparatively, only moderate increase in response time has been witnessed for both GD and CWRR in Table 4.2. Similar results have also been obtained in the Asia Sprint Network as depicted in Figure 4.4(c) and 4.4(d).

4.5.2.4 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 Figure 4.4(c)) is significantly larger than the gap in Europe (7 ms in Figure 4.4(a)) as we expected. Particularly, with the help of GD, the response time in Asia is successfully reduced from 85 ms to less than 60 ms (37.5% lower). In comparison, only 20% reduction in response time is achieved by GD in Europe. This is because the weights (4.10) 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 the 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.

4.5.3 Effectiveness of GA for the CPP

To demonstrate the effectiveness of GA[23], we conduct a set of simulations on Asia Sprint Network using 4 different controller settings as shown in Table 4.3. The controller capacities are set to either 60k or 90k pkt/s in each setting. To ease the discussion, we start with all controllers with the same capacities of 60k pkt/s as shown in setting 1 in Table 4.3. Then we gradually upgrade some controllers to larger capacities (90k 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 Figure 4.6. One controller within each region has the upgraded capacity of 90k pkt/s

[23]GA in the following discussion refers to Algorithm 2 introduced in Subsection 4.4.2 which combines the use of GA and GD without network partitioning

Table 4.3: Controller settings used in Asia Sprint Network.

Region	Number of Controllers								
	Total	Setting 1		Setting 2		Setting 3		Setting 4	
		90K	60K	90K	60K	90K	60K	90K	60K
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

Table 4.4: Control plane throughput (x10k pkt/s) with different controller settings and request arrival rates in Asia network.

Arrival rate (x10k pkt/s)		22	30	38	46	54	62
Setting 1	K-center	21.905	29.995	37.835	45.993	53.767	61.732
	GA	**21.906**	**29.997**	**37.837**	**45.994**	**53.768**	**61.733**
Setting 2	K-center	21.905	29.995	37.836	45.993	53.767	61.732
	GA	**21.906**	**29.996**	**37.837**	**45.994**	**53.768**	**61.733**
Setting 3	K-center	21.904	29.995	37.835	45.993	53.766	61.732
	GA	**21.906**	**29.996**	**37.837**	**45.994**	**53.768**	**61.733**
Setting 4	K-center	21.904	29.995	37.836	45.993	53.767	61.732
	GA	**21.906**	**29.996**	**37.837**	**45.994**	**53.768**	**61.734**

in setting 2. Sequentially, two controllers within each region are chosen and upgraded in setting 3. Finally all controllers are upgraded in setting 4.

4.5.3.1 Throughput Comparison

We measure and compare the throughput achieved by K-center and GA, as summarized in Table 4.4. The results show that there is no significant difference regardless of the controller settings and arrival rates, which is easily understandable for the following reasons. First, under all settings and request rates, the network is not fully loaded since 15% capacity is reserved at each controller according to (4.6). Second, GD-based scheduling approach ensures that no controller will be overloaded if total control

Table 4.5: 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**

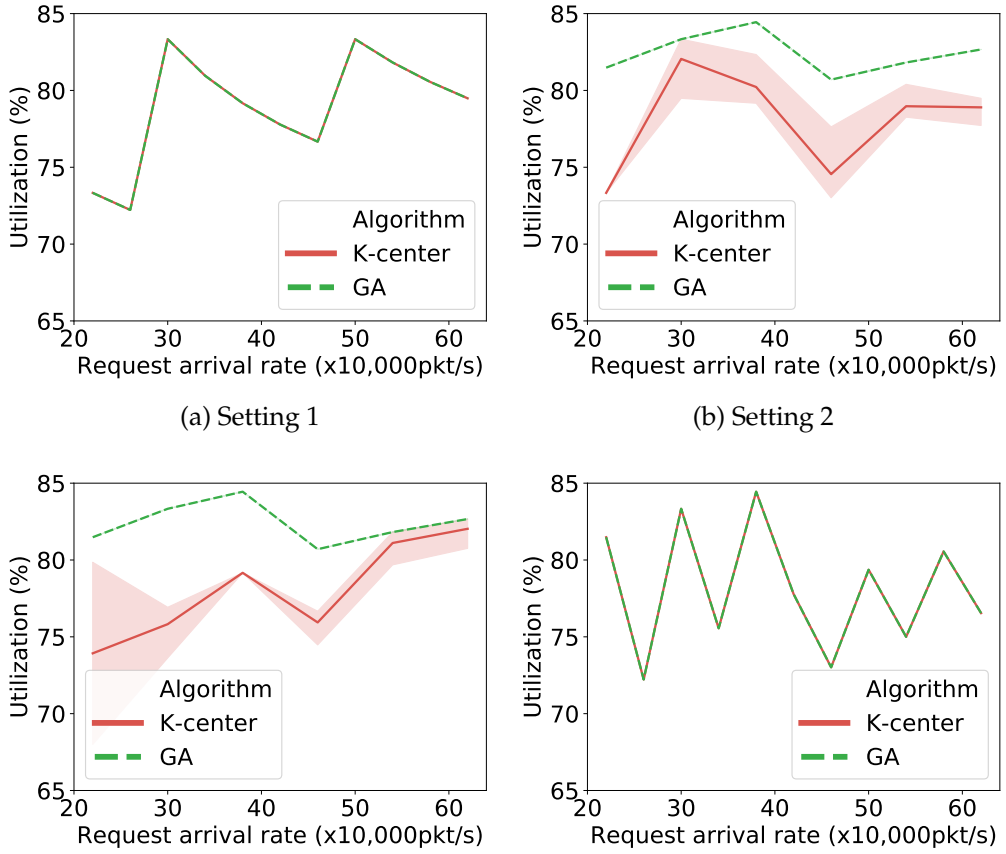
plane capacity is sufficient. For the remaining simulation studies, if the throughputs achieved by different placement algorithms are identical, the corresponding results will be omitted.

4.5.3.2 GA under 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 4.3). In Figure 4.5(a) and Figure 4.5(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 4.5. Detailed discussion can be found below.

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, both GA and K-center decided to deploy the same number of controllers, according to expectations. 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 420k pkt/s,



(a) Setting 1

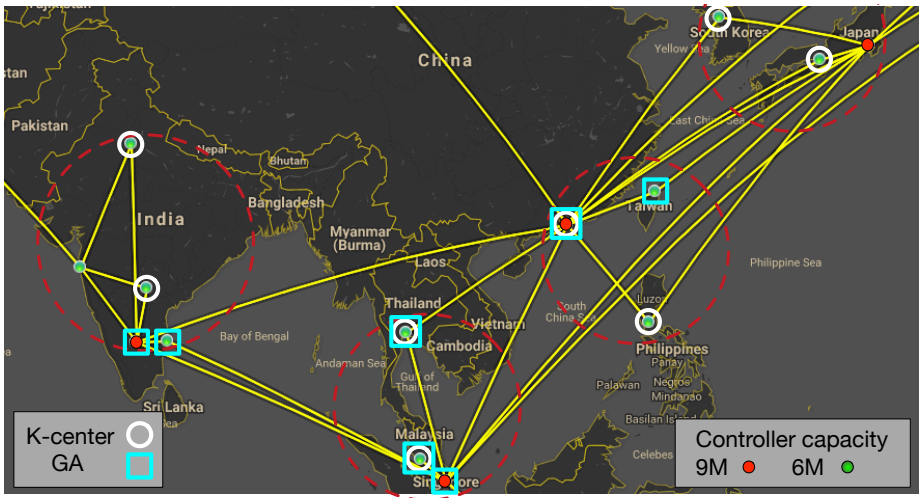
(b) Setting 2

(c) Setting 3

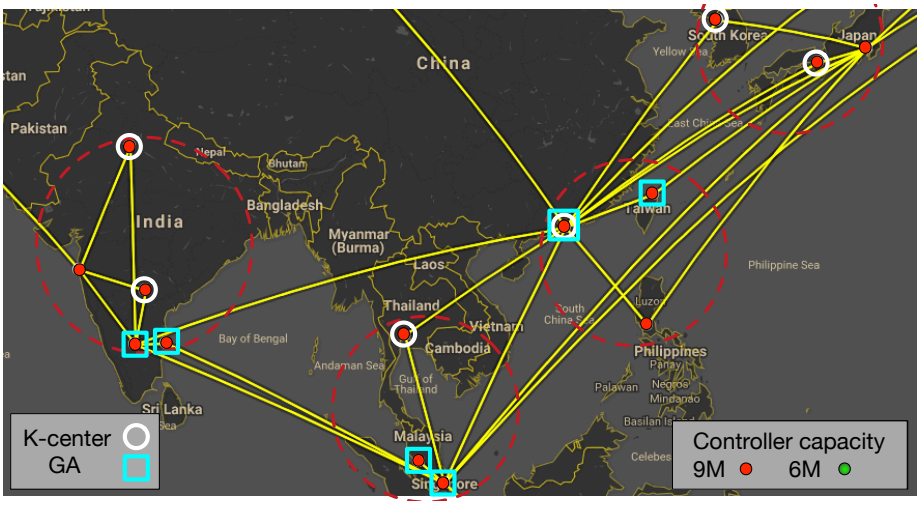
(d) Setting 4

Figure 4.5: 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 4.3 respectively.

which is shown in Figure 4.6(b). 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 bound-



(a) Setting 2



(b) Setting 4

Figure 4.6: Controller placement in Asia Sprint Network with different controller settings at the arrival rate of 420k pkt/s. (a) shows the controllers selected by K-center and GA in Table 4.3 setting 2 where controllers are with different capacities. Similarly, (b) shows the placement solution in Table 4.3 setting 4 where all controllers are with identical capacities.

ary 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%.

Another interesting phenomenon in Table 4.5 is that the average response time of GA rises slowly despite the large increase in arrival rate. This is because GA can adaptively allocate more controllers to cope with the increasing demand. On the other hand, when available locations for deploying new controllers become limited, the response time of GA will be slowly approaching the response time of K-center.

4.5.3.3 GA under Settings with Different Controllers

As shown in Figure 4.5(b) and Figure 4.5(c), when controllers have varied capacities (i.e., setting 2 and setting 3 in Table 4.3), GA can handle the CPP more effectively than K-center in terms of both utilization and response time as we expected.

We also visualize the placement results of both GA and K-center under setting 2 at the combined arrival rate of 420k pkt/s, as shown in Figure 4.6(a). Consistent with previous analysis, the gap between K-center and GA in response time comes from their choice of controller locations. In terms of utilization, GA strategically selects controllers with both high and low capacities so as to maximize the utilization. On the other hand, controllers with different capacities are oblivious to K-center. That is why the controllers chosen by K-center in Figure 4.6(a) are mostly with capacity 60k pkt/s. Such overlooking of controller capacity in K-center is also reflected by high fluctuation of utilization in Figure 4.5(b) and Figure 4.5(c).

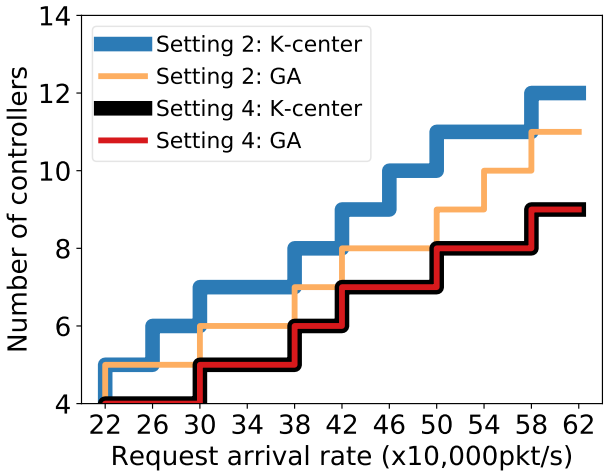


Figure 4.7: The changes of the number of selected controllers with different arrival rates using different controller settings in Asia Sprint Network.

Table 4.6: Controller settings used in Global Sprint Network.

Number of Controllers						
Total	Setting 1		Setting 2		Setting 3	
	90K	60K	90K	60K	90K	60K
82	0	82	18	64	42	40

4.5.3.4 Number of controllers

We also compare the numbers of controllers chosen by GA and K-center under different controller settings. As demonstrated in Figure 4.7, regardless of placement algorithms, more controllers are selected when the traffic increases. Furthermore, as for setting 2, K-center deployed more controllers than GA did, which results in the lower control plane utilization in Figure 4.5(b). For example, at the arrival rate of 500k pkt/s, the number of controllers chosen by GA is 8, which is 20% less compared to K-center. As for setting 4, we can notice that both K-center and GA select identical number of controllers, which further gives evidence of the same utilization in Figure 4.5(d).

Table 4.7: 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**

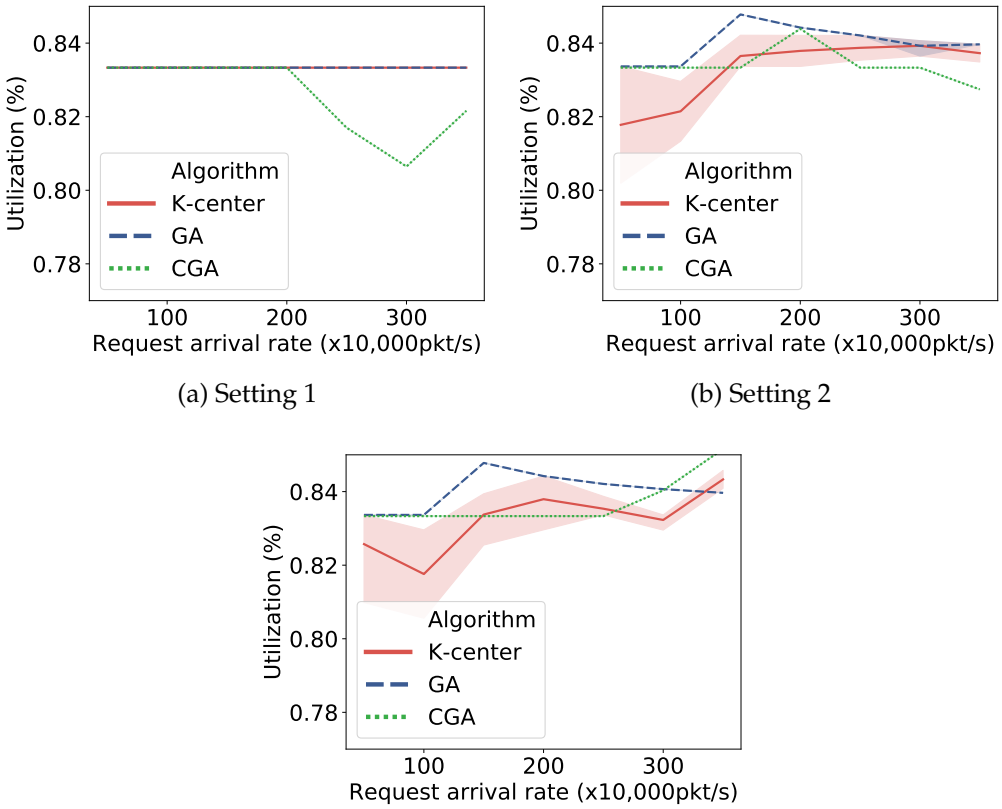
4.5.4 Effectiveness of CGA for the CPP

To further increase the difficulty of the CPP, a larger network (i.e., the global Sprint Network [19]) is adopted. Similar to Subsection 4.5.3, two optional controller capacities (60k and 90k pkt/s) are adopted during the simulation. According to our heuristic (4.24), the network will be divided into 2 sub-networks when the arrival rate remains below 2000k pkt/s. With further increase in arrival rate, the number of sub-networks will be doubled to 4.

Figure 4.8 and Table 4.7 show the results of all competing placement algorithms with three different controller settings summarized in Table 4.6. From Figure 4.8 and Table 4.7, we can clearly 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.

4.5.4.1 CGA under Settings with Identical Controllers

Similar to Figure 4.5(a) and Figure 4.5(d), we notice that in Figure 4.8(a), both K-center and GA achieved the same utilization since all controllers



(a) Setting 1

(b) Setting 2

(c) Setting 3

Figure 4.8: Performance comparison between K-center, GA, and Clustering-based GA for solving the CPP using different controller settings in Global Sprint Network. (a)-(c) are the network performance with controller setting 1 to 3 in Table 4.3 respectively.

have identical capacities in setting 1. 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 250k 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, we can clearly notice that CGA managed

to drastically reduce the response time by 50% (from 80 ms to 40 ms).

4.5.4.2 CGA under Settings with Different Controllers

When controllers have different capacities (i.e., setting 2 and 3 in Table 4.6), 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 Figure 4.8(b), which is almost negligible. Another interesting phenomenon is that in Figure 4.8(c), the utilization of CGA keeps increasing and even outperforms GA at the request arrival rate of $3,500$k pkt/s. Our results show that CGA enjoys higher chance of identifying better solutions in large search spaces.

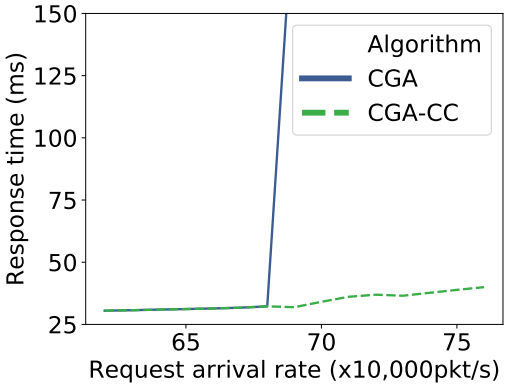
4.5.5 Effectiveness of CGA-CC for the CPP

In the simulations discussed previously, we focus mainly on CGA because no bursting requests occur in the simulated networks. In this situation, it is not necessary to share workload across different sub-networks. In this subsection, however, we want to evaluate when it is useful to support collaborative sub-networks through CGA-CC.

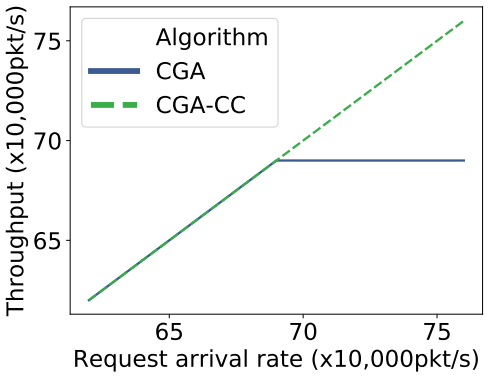
To demonstrate the effectiveness of CGA-CC on coping with unexpected traffic burst, we deploy a fixed collection of controllers in the Europe network with setting 3 in Table 4.6 and constantly increase the request arrival rate. When the arrival rate reaches the control plane capacity, we expect that some requests will be forwarded to controllers in neighboring sub-networks to avoid overloading the control plane.

As depicted in Figure 4.9(a), the response time of CGA stays below 32 ms and its throughput shows a steady growth before the arrival rate reaches 680k pkt/s. After that, a notable jump can be observed in CGA response time.

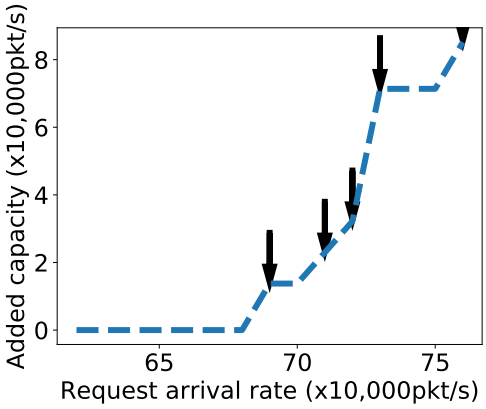
On the contrary, the response time of CGA-CC is consistent with CGA



(a) Response time



(b) Throughput



(c) Control plane capacity

Figure 4.9: Performance comparison with burst traffic. (a) and (b) are the response time and throughput using CGA and CGA-CC respectively. (c) shows when a new controller is "borrowed" from other sub-networks.

when the arrival rate is less than 680k pkt/s since the workload is still below the control plane capacity. After that, the gap in both response time (Figure 4.9(a)) and throughput (Figure 4.9(b)) 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 Figure 4.9(a). Correspondingly, we also demonstrate the changes of additional capacity of the sub-network control plane in Figure 4.9(c). We can see that CGA-CC can effectively borrow more controllers from other clusters as the arrival rate grows, preventing the control plane being overloaded.

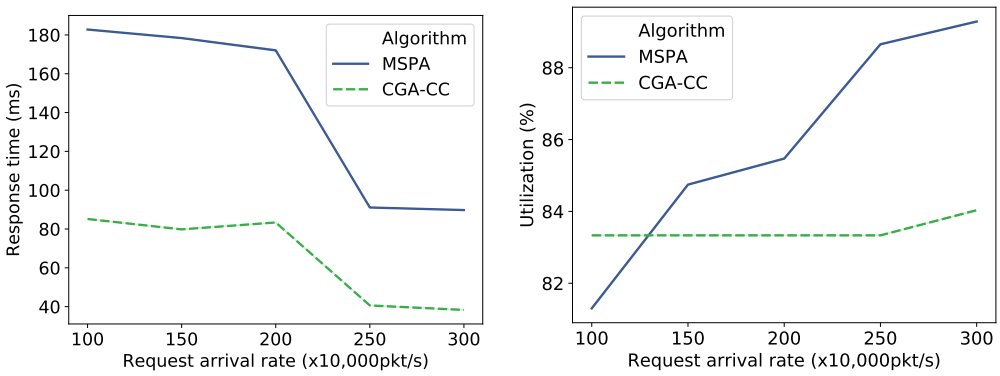
4.5.6 Comparison with MSPA

To further demonstrate the effectiveness of CGA-CC, we compare it with MSPA [279]. Similar to CGA-CC, MSPA first partitions the full network into a given number of sub-networks using the same partitioning algorithm CNPA (i.e., Algorithm 3) as we did. 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 Subsection 2.4.2.3. However, the authors in MSPA [279] 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 only the best performance achieved will be reported.

4.5.6.1 CGA-CC vs. MSPA without Burst Traffic

Figure 4.10 demonstrates the performance comparison between CGA-CC and MSPA. From Figure 4.10, we can clearly see that CGA-CC outperforms MSPA in terms of response time and controller utilization.



(a) Response time

(b) Utilization

Figure 4.10: Performance comparison without burst traffic. (a) shows the performance comparison without burst traffic using controller setting 3 in Table 4.6 with different request arrival rates. (b) shows the controller utilization among all selected controllers in Europe network and their normalized propagation latency with the scheduler in MSPA.

As shown in Figure 4.10(a), in comparison to MSPA, CGA-CC can significantly reduce the average request response time. For example, when the request arrival rate is 1000k pkt/s, the response time of MSPA is around 180 ms while CGA-CC is significantly smaller at 80 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 4.11 by showing the relationship between controller utilization and propagation latency between the scheduler and the controllers in MSPA. It can be clearly seen that controllers (e.g., C4) close to the scheduler re-

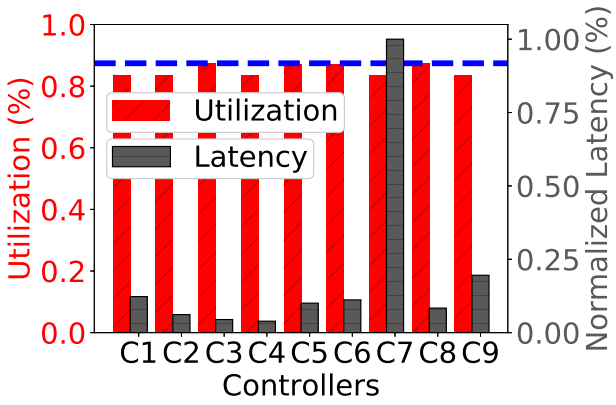


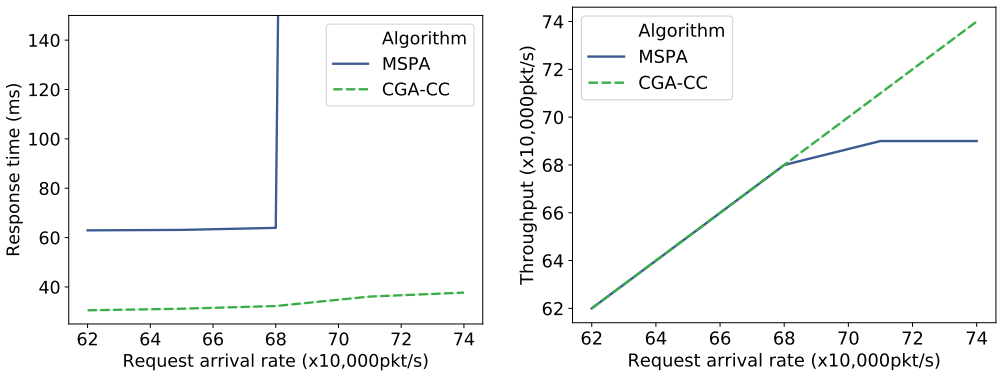
Figure 4.11: Controller utilization among all selected controllers in Europe network and their normalized propagation latency with the scheduler in MSPA.

ceive similar workload as the remote controller (e.g., C7), resulting in the higher response time compared with CGA-CC.

Although CGA-CC outperforms MSPA in terms of average response time, we can see from Figure 4.10(b) that MSPA achieves higher control plane utilization when the arrival rate is higher than 1500k 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 (4.6) which ensures that a proportion of capacity must be reserved for the control plane to withstand unexpected traffic bursts.

4.5.6.2 CGA-CC vs. MSPA with Burst Traffic

We also evaluate the performance of MSPA with burst traffic. Similar to Subsection 4.5.5, two sets of controllers selected by CGA-CC and MSPA in the Europe network are deployed. Since we aim to demonstrate the performance of CGA-CC and MSPA under burst traffic when there is no enough time to recalculate/deploy new controller placement, we fix the



(a) Response time

(b) Throughput

Figure 4.12: Performance comparison with burst traffic using the fixed controller placement from Europe network.

two sets of controllers and observe the network performance with increasing request arrival rate.

We can clearly notice that the performance of MSPA in Figure 4.12 follows a similar trend as CGA in Figure 4.9 which is easily understandable since both MSPA and CGA do not share workload across different sub-networks. In particular, the throughput of MSPA keeps growing steadily before the arrival rate reaches 680k pkt/s. After that, the throughput remains at 680k pkt/s regardless of any further increase in incoming traffic. In terms of the response time, we can see that MSPA manages to keep it at around 60 ms before the request arrival rate reaches 680k pkt/s. When increasing the arrival rate further, the response time soars up to above 90 ms since the control plane is overloaded.

In contrast, as we explained in Subsection 4.5.5, by sharing workload across sub-networks, CGA-CC can effectively cope with bursting traffic.

4.6 Chapter Summary

The overall goal of this chapter is to tackle the RM problem, particularly the CPP, from an algorithm design perspective. This goal has been successfully fulfilled by developing a Clustering-based Genetic Algorithm with Cooperative Clusters (CGA-CC) to solve both the CPP and the CSP simultaneously. In particular, the objectives identified in Subsection 4.1.1 have been addressed:

A new controller placement and scheduling problem (CPSP) has been introduced, which explicitly strengthens the importance of solving both the CPP and the CSP coherently within the same framework. The CPSP has been mathematically described as a constrained optimization problem with the goal of optimizing control plane utilization while simultaneously guaranteeing low network response time.

To address the CSP scalably and effectively, we have focused on optimizing the probabilities for request distribution over all controllers. A GD-based scheduling algorithm has been subsequently developed to balance the trade-off between scheduling performance and problem scalability.

In line with the solution of the CSP, CGA-CC has been proposed to address the CPP. In particular, to reduce the search space of GA, CGA-CC splits 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 has been developed to strategically offload indigestible requests to adjacent sub-networks.

Extensive simulations have been conducted based on real-world topologies and traffic. The results have showed that GD-based scheduling algorithm can effectively reduce the response time while maintaining high control plane throughput. On the other hand, CGA-CC has effectively improved the resource utilization of the control plane without sacrificing response time, in comparison to the widely-used K-center approach

and the state-of-the-art algorithm.

Note that Internet traffic is complex and can be affected by many factors (e.g., topology and applications), which makes the theoretical analysis very difficult. Thus, similar to existing studies, in this chapter, we modelled the requests following a Poisson distribution [281]. This is clearly a simplification. To understand the limitations of our Poisson model, we compared the network performance obtained by using the Poisson traffic and the real-world network traces. In general, their performance was similar. Significant difference can be spotted only when the controllers were experiencing high workload. In other words, when a controller was under high workload, the results generated using the Poisson model were more optimistic compared to real-world traffic. To address this issue, our problem formulation presented in CPP carefully preserves a small portion of capacity on each controller as determined by a decay factor. Thus, as long as the workload of a controller does not exceed a certain threshold, we can confirm that the deviation introduced by the Poisson traffic in our problem formulation can be safely ignored.

Note that CGA-CC has simultaneously addressed both the CPP and the CSP. However, there are certain limitations of our existing solution to the CSP. In particular, the queuing model (4.2.3) assumes that each controller queue is a stationary process and calculates the average response time for a given request arrival rate. Therefore, the performance of GD relies on the accuracy of the provided network information. Apart from that, whenever the request arrival rate changes, GD needs to rerun to solve (4.9) which can introduce additional computation overhead. Since the network we considered in this chapter is a large global communication backbone where frequent network changes are unlikely to happen, our GD-based scheduling is applicable. However, new CS solutions are needed which will be addressed in the next chapter.

Chapter 5

Deep Reinforcement Learning for Request Dispatching in SDN

5.1 Introduction

The architecture BLAC proposed in Chapter 3 enables switches dispatch requests to any controllers without the switch-controller binding constraint, effectively alleviating the workload imbalance issues. With the request dispatching flexibility provided by BLAC, designing a policy to properly dispatch requests originated from every switch to suitable controllers is of paramount importance to the overall functioning of multi-controller SDNs [136, 138]. Motivated by this understanding, we aim to address the Request Dispatching Policy Design (RDPD) problem in this chapter.

Particularly, the designed policy must satisfy three requirements:

(R_1) *Time efficiency*: Since request dispatching must be performed in real time with minimum delay, the designed policy needs to be sufficiently efficient in practice. Therefore, policies with long processing time or frequent execution (e.g., in a per-request manner) should be avoided.

(R_2) *Adaptiveness*: Note that the number of controllers in an SDN network can change in order to meet the varying traffic demand. Thus, the designed policy should perform consistently well over different numbers of controllers.

(R_3) *Performance effectiveness*: The designed policy should guide switches to properly dispatch requests to suitable controllers to minimize the average request response time.

To achieve R_3, existing studies [100, 254, 255, 280] constructed mathematical models to capture the correlation between the policy and the performance objective (e.g., average request response time). Although these model-driven methods can generally provide solutions with guaranteed performance, modeling the highly complicated network requires *substantial domain knowledge*. Moreover, in a highly complicated distributed computing environment (such as a distributed controller architecture), the response time can be caused by many factors that may not be fully captured using the proposed model.

Alternatively, the literature has considered either manually or automatically designing policies for resource allocation [138, 212, 219]. Specifically, two widely-used manually designed policies in operating systems and cloud computing are weighted round-robin and first-come-first-serve [239]. Obviously, they cannot achieve R_3 due to the lack of considering propagation latency. On the other hand, EC methods have been proposed to automatically design policies for standard job shop scheduling problems [212, 219]. However, EC methods have *high sampling costs* since data collected from previous generations cannot be reused in the next generation. Therefore, all candidate solutions in each generation need to be reevaluated in either simulated or real-world environments.

Recently, machine learning has been successfully applied to various RM problems [54, 184, 192, 193, 265]. Among all machine learning algorithms, we consider DRL to be a powerful paradigm for solving our RDPD

problem for several reasons. First, *no explicit mathematical model of the underlying complex environment is required*. DRL can automatically learn the optimal solution while interacting with the unknown dynamic environment through a trial-and-error process. Second, *DRL can improve current policies mainly based on experiences/data obtained from an old policy* through a technique known as experience replay [201]. Thus, in comparison to an EC approach, the sampling cost of training any new policies can be greatly reduced. Third, the scheduling problem under a specific network setting can be naturally formulated as an MDP (see Subsections 5.2.1 and 5.3.1 for a detailed discussion).

Despite the clear advantages offered by DRL, the direct application of existing value function indirect search techniques[24], e.g., DQN [202], may not be suitable for solving the RDPD problem. This is mainly because the policies learned by these techniques are only implicitly represented. Making a decision requires extensively enumerating the entire action space to find the action with maximum reward, which is time-consuming and violates R_1. Thus, policy direct search[25] which directly learns the optimal policy by searching the policy space is more appropriate. However, there are still several major issues must be addressed.

(1) *Non-adaptive and inefficient policy design*: Typically, many existing approaches [174, 175, 185] on RM were DQN-based policy direct search. These approaches were designed to handle problems with a discrete action space. In the SDN request dispatching context, an action can be defined as assigning a set of R requests to N available controllers. In this case, the size of the action space is R^N. With a large action space, the complexity of DNN architectures inevitably increases, resulting in longer computa-

[24]Value function indirect search learns the optimal value function which is used to extract the optimal policy by greedily selecting the action that maximizes the long-term rewards. More details about value function indirect search can be found in Subsection 2.2.6.3

[25]Policy direct search directly learns the optimal policy by searching the policy space. More details about policy direct search can be found in Subsection 2.2.6.3

tional time to make a request dispatching decision. To reduce the action space, an action can also be defined as assigning one request to a controller every time where the size of the action space is N. However, this action definition requires the policy to be processed repeatedly with respect to every new request, incurring non-negligible policy processing overhead. Therefore, both action definitions cannot satisfy R_1.

Moreover, existing DRL approaches directly represent their policy as a DNN with a fixed number of output nodes. The number of output nodes is the size of the action space. Such a representation apparently violates R_2 since the same policy may fail to function well whenever the number of controllers N is changed to meet the varying traffic demand.

(2) *Inapplicable adaptive policy training*: This thesis aims to design an adaptive policy that can support a dynamically changing number of controllers. However, the requirement to train an adaptive policy cannot be satisfied by directly following any existing DRL-based approaches [193, 265]. This is because the policy training requires the calculation of the policy gradient. In standard DRL, e.g., Trust Region Policy Optimization (TRPO) [240] and Proximal Policy Optimisation (PPO) [242], a policy is directly represented as a DNN targeting at a fixed number of output nodes where the policy gradient can be easily calculated. However, when a policy that can support a changing number of controllers, how to compute its gradient needs to be addressed.

(3) *Impractical problem formulation*: Designing a policy by a single learning agent requires the support of global network information (i.e., fully observable environment). Such a centralized approach is prone to scalability issues [300]. For example, the single agent can be overwhelmed by the enormous traffic and becomes the performance bottleneck. Meanwhile, obtaining timely global information over the entire SDN network can cause extra communication overhead [287].

As far as we know, none of the existing studies have considered and solved the above issues.

5.1.1 Chapter Goals

In fulfillment of Objective 3, new DRL-based methods are proposed with the aim to automatically learn effective, efficient and adaptive policies. In particular, we first propose a new DNN-based policy representation that can be applied to networks with a changing number of controllers. In line with the new policy representation, a new mathematical technique is developed to calculate its gradient. To demonstrate the effectiveness of the new policy design, the evaluation is conducted under the SA-DRL setting where the RDPD problem is formulated as an SA-MDP. In line with the new policy representation, an MA-DRL approach is proposed with a Multi-Agent (MA) training algorithm. This chapter aims to address the following objectives:

- Formulate the RDPD problem as an MDP in both SA and MA settings;
- Design an adaptive policy representation that can apply to a network with a changing number of controllers;
- Develop a new mathematical technique to calculate the policy gradient with respect to the new adaptive policy design;
- Design SA-DRL and MA-DRL algorithms that can effectively train an adaptive policy;
- Compare the performance of the new adaptive policy representation with the existing DRL policy design;
- Compare the performance of our policies designed by our SA-DRL and MA-DRL algorithms with man-made, model-based, and DRL-trained policies.

5.1.2 Chapter Organization

The rest of this chapter is organized as follows. Section 5.2 formulates the RDPD problem as an SA-MDP and presents the new policy design. With the new policy design, we consider a more general and practical scenario where multiple agents are involved in Section 5.3. In particular, the RDPD problem is reformulated as an MA-MDP and a new DRL algorithm called Multi-Agent Proximal Policy Optimisation (MA-PPO) is proposed. Extensive evaluations are presented in Subsections 5.2.5 and 5.3.3. Section 5.4 concludes this chapter.

5.2 An Adaptive Policy Design for Request Dispatching

This section mainly focuses on designing a policy that satisfies R_1, R_2, and R_3. To demonstrate the effectiveness of the new policy design, evaluating it under an SA-DRL framework is more preferable compared to MA-DRL. This is mainly because the performance of MA-DRL depends not only on the policy design but also other factors, such as inter-agent cooperation and non-stationary environment handling. In view of this, it is easier and more straightforward to verify the effectiveness of the new policy design with SA-DRL. In line with the proposed objectives of this chapter, this section first demonstrates the mapping from the RDPD problem to an MDP in Subsection 5.2.1. After that, the designs of an adaptive policy and a dispatching system are presented in Subsection 5.2.2 and Subsection 5.2.3 respectively. Along with the new dispatching system, a new training algorithm called Single-Agent Proximal Policy Optimisation (SA-PPO) is developed in Subsection 5.2.4 based on PPO [242] with a new gradient calculation technique in Subsection 5.2.4.1. The effectiveness of the new policy design is demonstrated in Subsection 5.2.5.

5.2.1 Modeling the RDPD Problem as an MDP

In this section, we consider a similar network environment introduced in Subsection 4.2.2 adopting the same notations. The difference is that since we consider the RDPD problem under the SA-DRL framework, a centralized scheduler/agent is deployed in the network that is responsible for dispatching requests generated by all switches. Specifically, in the data plane, once a request is generated at a switch, it will be immediately sent to the centralized agent. Upon receiving a request, the agent forwards it to a controller selected by our policy. After the controller finishes processing the request, the response originated from the controller is sent back to the corresponding switch through the agent.

In line with the MDP framework provided in Subsection 2.2.6.1, the RDPD problem can be naturally formulated as a *fully observable* SA-MDP with one agent that controls the request dispatching of M SDN switches. The SA-MDP can be described by a 4-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$. The overall network operating status, which is accessible to the centralized agent, is captured by a set of global states $\mathcal{S}$ including all current and historical network information, such as the request arrival rate $\boldsymbol{\lambda}$. The use of historical information can help the agent to better predict the future network status. More importantly, with the historical information, the global states $\mathcal{S}$ can better fulfill the Markov property (see Subsection 2.2.6.1).

At each time step t, the agent observes its current state $s_t \in \mathcal{S}$ and takes an action $a_t \in \mathcal{A}$ chosen from a policy $\pi_{\boldsymbol{\theta}}$. More details of $\pi_{\boldsymbol{\theta}}$ will be introduced in Subsection 5.2.2. The action $\boldsymbol{a}_t = \{a_t^{v'}\}_{v' \in V'}$ specifies the priority $a_t^{v'}$ of controller $C_{v'}$ during time t and $t+1$. The priorities $\{a_t^{v'}\}_{v' \in V'}$ are then used in the dispatching system to calculate the request dispatching probabilities $\{p_t^{v'}\}_{v' \in V}$. Each element $p_t^{v'}$ specifies the probability of dispatching new requests to controller $C_{v'}$. More details on how $\{a_t^{v'}\}_{v' \in V'}$ are utilized by the dispatching system are provided in Subsection 5.2.3. After performing $\boldsymbol{a}_t$, the network keeps operating until it enters the next state s_{t+1}.

Table 5.1: Reward v.s. Average response time

Request Set χ_t	Individual response time τ_x	Reward r_t (5.2)	Average response time
χ_t^1	0.1, 0.5, 0.5	14	0.36
χ_t^2	0.3, 0.3, 0.3	10	0.3

In order to train $\pi_{\boldsymbol{\theta}}$ towards minimizing the average request response time, we initially define the reward as

$$r_t = \sum_{x \in \chi_t} -\tau_x \tag{5.1}$$

where τ_x is the response time of a particular request. χ_t stands for the set of requests, for which the corresponding responses from controllers have been received by the respective switches in between time steps t and $t+1$. However, in our simulation studies, we notice that the reward cannot effectively lead to the improvement of network performance. This is because the reward r_t defined in (5.1) depends not only on response time τ_x but also on the number of requests $\|\chi_t\|$. Apparently reducing $\|\chi_t\|$ can actually increase r_t. Thus, the policy will learn to send requests to controllers with low capacities so as to reduce $\|\chi_t\|$.

During our preliminary work, we also experiment with the below reward definition:

$$r_t = \sum_{x \in \chi_t} \frac{1}{\tau_x} \tag{5.2}$$

where each request contributes $\frac{1}{\tau_x}$ to the total reward. However, we notice that a higher reward does not necessary lead to lower average response time and higher throughput due to the nonlinearity of the reward function. An example is provided in Table 5.1 where χ_t^1 achieves a higher reward and has a higher average response time compared to χ_t^2.

In consideration of the above reasons, the reward is redefined as:

$$r_t = \varsigma \|\chi_t\| - \sum_{x \in \chi_t} \tau_x \tag{5.3}$$

where ς is a weight factor that controls the importance of the throughput χ_t relative to the response time. In our simulation, ς is estimated as the average response time of CWRR in Subsection 4.3.1 which serves as a baseline for the policy. Guided by this reward, the policy is strongly motivated to receive more responses from controllers and to reduce the average response time. By maximizing the cumulative reward in (5.4), we can therefore fulfill our goal of reducing the request response time and improving the network performance.

$$J(\pi_{\boldsymbol{\theta}}) = \mathbb{E}_{s_t, \boldsymbol{a}_t \sim \pi_{\boldsymbol{\theta}}} \left\{ \sum_{t=0}^{T} \gamma^t r_t(s_t, \boldsymbol{a}_t) \right\} \tag{5.4}$$

where $\gamma \in [0, 1)$ is a discount factor. Evaluation with different γ values will be reported in Subsection 5.2.5.3.

5.2.2 DNN-based Adaptive Policy Design

By modeling the RDPD problem as an MDP, DRL algorithms can be utilized for policy design. However, as we mentioned in Section 5.1, existing policy representation fails to meet both R_1 and R_2. One possible strategy to solve R_2 is to train multiple policies while each policy targeting at a particular number of controllers. However, the cost of evaluation or training a policy in a production network can be high. This is mainly because production networks should always guarantee reasonable good performance while the policy can perform badly (e.g., overloading some controllers which leads to long response time) especially at the early training stage. Furthermore, policies for each particular number of controllers need to be individually evaluated or trained in advance before being deployed, which leads to high sampling costs. Thus, instead of training multiple policies, we should design and train an adaptive policy that can support different numbers of controllers.

Inspired by the successful use of dispatching rules[26] for supporting diverse job shop scheduling tasks [212, 219], we decide to employ a priority function f_{θ}^{p} to determine the priority for each controller to process every incoming packet. Consequently, the same f_{θ}^{p} can be used to prioritize an arbitrary number of controllers. In particular, we adopt Neural Network (NN) in this chapter to represent f_{θ}^{p} which can be justified as follows:

(1) Expressiveness: According to the universal approximation theorem, a depth-2 NN with suitable activation functions and a sufficient size[27] can approximate any continuous function [39, 76, 97, 126, 189]. However, existing studies [187, 189, 247] also pointed out that the expressiveness of such a depth-2 NN comes at the price of the network size. In particular, the size of such a neural network can be exponential in the input dimension, which means that the width of the NN can be large. To address this issue, DNN is adopted which was demonstrated that increasing the depth of the NN can exponentially reduce its width without sacrificing its expressiveness [70, 264].

(2) Trainability: An NN can be trained to improve its performance by iteratively updating/adjusting its weights using gradient descent optimization algorithms, e.g., Adam [157] and RMSprop [267].

(3) Efficiency: Note that with a properly designed policy, we expect to reduce the average request response time. To achieve this, short function execution time (i.e., R_1) is crucial to avoid potential network performance degradation. We consider that the NN can meet R_1 for the following reasons. First of all, small feed-forward NNs can be quickly processed by a commodity-level computer, supported by efficient and performance-optimized software such as TensorFlow [21]. Secondly, hardware technology advancement enables lightning

[26] A dispatching rule is used to prioritize all the jobs in the queue such that the job with the highest priority can be processed next.

[27] The size of an NN is the total number of neurons [187].

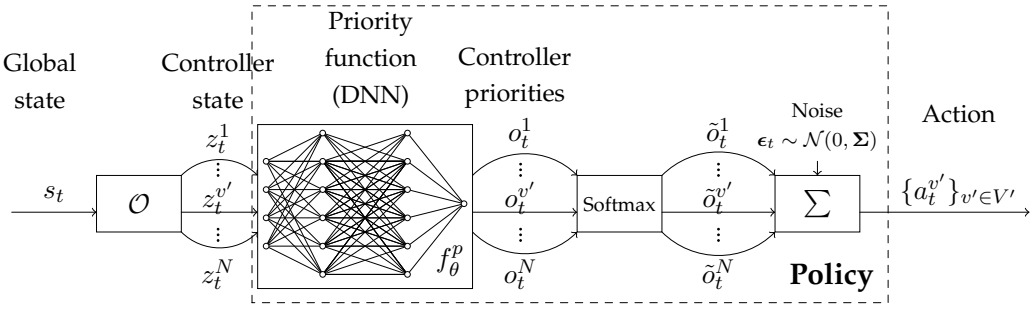


Figure 5.1: The DNN-based adaptive policy design.

fast processing of NNs in dedicated processing chips that have become widely available, even for mobile devices [6]. Thirdly, our policy is executed once in every given time interval. In comparison to the interval duration, the NN processing time is negligible (See Subsection 5.2.1 for the interval duration setting).

It is important to note that in job shop scheduling problems, the job with the highest priority will always be processed first according to the dispatching rule, which requires repeated use of dispatching rules with respect to each job. In comparison, the policy generates priorities for all controllers. The priorities are mapped into request dispatching probabilities to guide request dispatching in every given time interval, effectively avoiding the frequent policy calculation. Apart from that, the mapping from priorities to probabilities gives more controllers non-negligible opportunity of processing a request, thereby encouraging exploration during DRL in hope of achieving higher performance.

In line with this idea, we propose a new policy design, as shown in Figure 5.1. In particular, the policy $\pi_{\boldsymbol{\theta}}$ takes the state s_t as inputs and outputs an action $\boldsymbol{a}_t$. In our previous work [136], the action corresponds to the chosen controller for request processing. This design requires repeated processing of the policy with respect to every new request, preventing efficient use of the policy in large traffic-intensive networks. This issue is addressed by defining $\boldsymbol{a}_t = \{a_t^{v'}\}_{v' \in V'}$ as the controller priorities to guide

request dispatching, which are updated once in every given time interval as discussed earlier.

Priority mapping: As we discussed in Section 5.1, existing policies are generally represented as a DNN (e.g., policies trained using PPO [242]). These policies generate the request dispatching probabilities $\{p_t^{v'}\}_{v' \in V'}$ through one run of the DNN, which cannot adapt to a changing number of controllers. In our new policy design, the specific state information $z_t^{v'}$ with respect to each controller $C_{v'}$ is first extracted from s_t. Then $z_t^{v'}$ is fed one-by-one to the DNN in Figure 5.1 for all controllers. For each input state information $z_t^{v'}$, the DNN assigns a priority value $o_t^{v'}$ to $C_{v'}$. In particular, the DNN is the priority function $f_{\boldsymbol{\theta}}^p$ with trainable parameters $\boldsymbol{\theta}$, which is different from the policy $\pi_{\boldsymbol{\theta}}$ with additional components for *normalization* and *exploration*[28], as explained below.

Normalization and exploration: The softmax function is used to normalize all controllers' priorities $\{o_t^{v'}\}_{v' \in V'}$ into a probability distribution $\{\tilde{o}_t^{v'}\}_{v' \in V'}$, as indicated in Figure 5.1. Rather than using $\{\tilde{o}_t^{v'}\}_{v' \in V'}$ in a deterministic manner, the agent must continue to explore different request dispatching distributions and determine their impact on network performance during policy training. This is achieved by adding small Gaussian noises

$$\epsilon_t^{v'} \sim \mathcal{N}(0, \sigma^2) \qquad \forall v' \in V'.$$

to $\{\tilde{o}_t^{v'}\}_{v' \in V'}$, as shown below:

$$\boldsymbol{a}_t = \tilde{\boldsymbol{o}}_t + \boldsymbol{\epsilon}_t, \tag{5.5}$$

where $\boldsymbol{a}_t = \{a_t^{v'}\}_{v' \in V'}$, $\tilde{\boldsymbol{o}}_t = \{\tilde{o}_t^{v'}\}_{v' \in V'}$, and $\boldsymbol{\epsilon}_t = \{\epsilon_t^{v'}\}_{v' \in V'}$.

In association with the discussion above, the whole action generation

[28]The exploration component in a policy is only activated during policy training for stochastic exploration of different request dispatching distributions. While testing the trained policy on a SDN network, this component is deactivated.

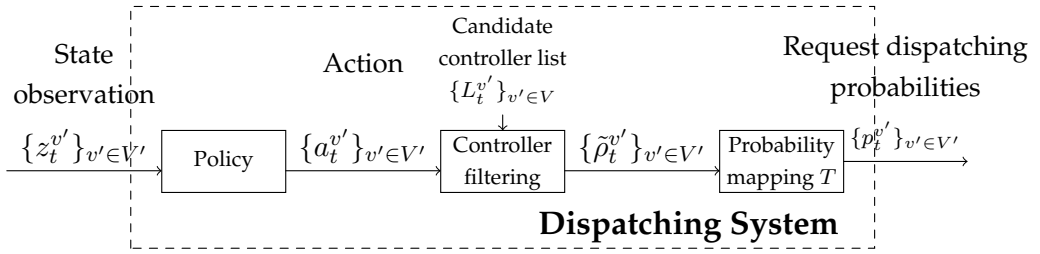


Figure 5.2: The design of the dispatching system.

process based on our new policy design can be formulated as:

$$\boldsymbol{a}_t = \begin{bmatrix} a_t^1 \\ \vdots \\ a_t^N \end{bmatrix} = \pi_{\boldsymbol{\theta}} \left(\begin{bmatrix} z_t^1 \\ \vdots \\ z_t^N \end{bmatrix} \right) = \begin{bmatrix} \text{Softmax}(f_{\boldsymbol{\theta}}^p(z_t^1)) + \epsilon_t^1 \\ \vdots \\ \text{Softmax}(f_{\boldsymbol{\theta}}^p(z_t^N)) + \epsilon_t^N \end{bmatrix} \tag{5.6}$$

Because of $\boldsymbol{\epsilon}_t$, (5.6) produces $\boldsymbol{a}_t = \{a_t^{v'}\}_{v' \in V'}$ as the continuous action output in a stochastic manner.

5.2.3 The Dispatching System Design

When performing request dispatching, the switch should avoid sending requests to unsuitable controllers, e.g., overloaded or remotely located controllers. Driven by this motivation, a *controller filtering mechanism* is designed and used before probability mapping. In particular, the agent keeps track of the operating status of all controllers[29] and maintains a candidate controller list $\boldsymbol{L}_t = \{L_t^{v'}\}_{v' \in V'}$. Preference is given to controllers with relatively small propagation latency from the agent as well as controllers under moderate or low workload[30]. Accordingly, up to χ controllers can

[29]Controller status update is realized through regular beacon messages send by every controller to the agent in the network. Due to the communication overhead, beacon messages are not communicated at high frequencies. Hence, the status information accessible to the agent can be slightly outdated. Despite this, we will show experimentally in Subsection 5.2.5.3 that the agent can achieve high network performance via DRL.

[30]The average queue length of a controller must fall below a predefined threshold for the controller to be considered for request processing.

be considered as *candidates* by the agent where χ can be flexibly set to control the level of exploration. For example, when $\chi = N$, all controllers are considered for request dispatching. Similarly, the agent can only explore at most χ controllers when $\chi < N$.

Armed with the adaptive policy and the controller filtering mechanism, a dispatching system is designed as shown in Figure 5.2. The dispatching system takes the state observations as inputs and outputs the request dispatching probabilities. In particular, given the state observations $\{z_t^{v'}\}_{v' \in V'}$, the policy generates the priorities $\{a_t^{v'}\}_{v' \in V'}$ as we discussed in Subsection 5.2.2. After that, overloaded or remotely located controllers are filtered by assigning 0 to their priorities with the help of $\{L_t^{v'}\}_{v' \in V'}$. The filtered priorities $\{\tilde{\rho}_t^{v'}\}_{v' \in V'}$ are then mapped to $\{p_t^{v'}\}_{v' \in V'}$ through function T. Mathematically, $\boldsymbol{L}_t$ is presented as a binary vector that covers all the N controllers, with the corresponding elements of $\boldsymbol{L}_t$ for candidate controllers taking the value 1.

5.2.4 SA-PPO for Policy Learning

In line with the new policy design, a training system is proposed to optimize the policy as shown in Figure 5.3. As we discussed in Section 5.1, existing DRL algorithms cannot train policies following our adaptive design proposed in Subsection 5.2.2. To train $\pi_{\boldsymbol{\theta}}$ effectively, Single-Agent Proximal Policy Optimisation (SA-PPO) is developed in this section. SA-PPO is an extension of the PPO algorithm [242] for DRL, armed with a newly developed mathematical technique (see Subsection 5.2.4.1) to compute the policy gradient.

Among existing DRL algorithms, we select PPO for policy training for several reasons.

(1) PPO can update the current policy using experiences/data sampled from an old policy, greatly improving sample efficiency.

(2) PPO employs only first-order optimization which is more computationally efficient compared to TRPO [240].

(3) In comparison to other actor-critic algorithms [117, 181] that train a Q-function with large function input including the action space, the policy training in PPO only relies on the V-function. Compared to the Q-function, the input dimensions in the V-function are substantially reduced, especially in an MA setting with a large MA joint action space. Therefore, learning the V-function is easier than the Q-function.

(4) PPO has been widely and successfully used in many problem domains. Studies [242] have shown that PPO can outperform many state-of-the-art algorithms such as TRPO [240] and A2C [201] on many difficult DRL problems. Therefore, we consider PPO to be a promising algorithm to tackle our RDPD problem.

While we only use PPO in this thesis, our research does not rule out the possibilities of using other DRL algorithms.

SA-PPO uses a DNN to approximate a parametric value function $\mathcal{V}_{\boldsymbol{\omega}}$ with the global state input $s_t \in \mathcal{S}$. Following PPO, $\mathcal{V}_{\boldsymbol{\omega}}$ will be learned in an on-policy fashion by maintaining a collection of network state-transition samples obtained from using the current policy $\pi_{\boldsymbol{\theta}}$. Each state-transition sample u is recorded in the following form:

$$u = \langle s_t, s_{t+1}, \boldsymbol{a}_t, r_t \rangle \tag{5.7}$$

Then several *mini-batches* of samples, i.e., $\mathcal{B}$, can be retrieved from the collection to repeatedly train $\mathcal{V}_{\boldsymbol{\omega}}$ to minimize the *Bellman loss* below:

$$\mathcal{H}(\mathcal{V}_{\boldsymbol{\omega}}) = \frac{1}{\|\mathcal{B}\|} \sum_{\mathcal{B}} \left(\mathcal{V}_{\boldsymbol{\omega}}(s_t) - r_t - \gamma \mathcal{V}_{\boldsymbol{\omega}}(s_{t+1})\right)^2 \tag{5.8}$$

Guided by the trained $\mathcal{V}_{\boldsymbol{\omega}}$, the *actor* in SA-PPO continues to use the sampled mini-batches to update $\pi_{\boldsymbol{\theta}}$ along the direction of estimated *policy gradient* in Subsection 5.2.4.1.

5.2.4.1 Policy Gradient Calculation in SA-PPO

Given the value function $\mathcal{V}_{\boldsymbol{\omega}}$, SA-PPO optimizes the policy $\pi_{\boldsymbol{\theta}}$ by maximizing the following clipping function:

$$\mathcal{L}(\pi_{\boldsymbol{\theta}}) = \mathbb{E}_{s_t,\boldsymbol{a}_t\sim\pi_{\boldsymbol{\theta}_{old}}}\left[\min\left(\frac{\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t|s_t)}{\pi_{\boldsymbol{\theta}_{old}}(\boldsymbol{a}_t|s_t)}A_t(s_t,\boldsymbol{a}_t), \text{clip}\big(\frac{\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t|s_t)}{\pi_{\boldsymbol{\theta}_{old}}(\boldsymbol{a}_t|s_t)}, 1-\varepsilon, 1+\varepsilon\big)A_t(s_t,\boldsymbol{a}_t)\right)\right]$$

where ε is a hyper-parameter that is set to 0.2 following PPO. $\boldsymbol{\theta}$ and $\boldsymbol{\theta}_{old}$ refer to the policy parameters after and before policy update in a Training Iteration (TI) respectively. $A_t(s_t,\boldsymbol{a}_t)$ is the advantage function obtained through $\mathcal{V}_{\boldsymbol{\omega}}$ by using the Generalized Advantage Estimation (GAE) technique developed in [241].

In particular, $\nabla_{\boldsymbol{\theta}}\mathcal{L}(\pi_{\boldsymbol{\theta}})$ can be estimated as shown below:

$$\nabla_{\boldsymbol{\theta}}\mathcal{L}(\pi_{\boldsymbol{\theta}}) \approx \frac{1}{\|\mathcal{B}\|}\sum_{\mathcal{B}}\frac{A_t(s_t,\boldsymbol{a}_t)}{\pi_{\boldsymbol{\theta}_{old}}(\boldsymbol{a}_t|s_t)}\nabla_{\boldsymbol{\theta}}\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t|s_t)$$

provided that $\frac{\pi_{\boldsymbol{\theta}}}{\pi_{\boldsymbol{\theta}_{old}}}$ falls in the range $(-\infty, 1+0.1)$ if $A_t(s_t,\boldsymbol{a}_t) > 0$ or $(1-0.1, +\infty)$ if $A_t(s_t,\boldsymbol{a}_t) < 0$, with respect to any $\boldsymbol{a}_t$ and s_t. Otherwise, the policy gradient of the corresponding $\boldsymbol{a}_t$ and s_t is 0.

According to (5.6),

$$a_t^{v'} - \text{Softmax}(f_{\boldsymbol{\theta}}^p(z_t^{v'})) = \epsilon_t^{v'} \sim \mathcal{N}\left(0, \sigma^2\right)$$

Therefore, each element $a_t^{v'}$ in $\boldsymbol{a}_t$ follows a Gaussian distribution:

$$a_t^{v'} \sim \mathcal{N}\left(\text{Softmax}(f_{\boldsymbol{\theta}}^p(z_t^{v'})), \sigma^2\right)$$

Note that the Gaussian noise $\epsilon_t^{v'}$ for each $a_t^{v'}$ is independently sampled. Therefore,

$$\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t|s_t) = \prod_{v'\in V'}\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{1}{2}(\frac{a_t^{v'}-\mu_t^{v'}}{\sigma})^2}$$

where $\mu_t^{v'} = \text{Softmax}(f_{\boldsymbol{\theta}}^p(z_t^{v'}))$ and $z_t^{v'} = \mathcal{O}(s_t, v')$ is the specific state information with respect to each controller $C_{v'}$ extracted from s_t.

For each sample $u_t = \langle s_t, s_{t+1}, \boldsymbol{a}_t, r_t \rangle \in \mathcal{B}$, $\nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t|s_t)$ can be calculated by using $\boldsymbol{a}_t$ and s_t recorded in sample u as shown below:

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t|s_t) &= \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t|s_t) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t|s_t) \\
&= \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t|s_t) \nabla_{\boldsymbol{\theta}} \log \Big(\prod_{v' \in V'} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{a_t^{v'} - \mu_t^{v'}}{\sigma})^2} \Big) \\
&= \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t|s_t) \nabla_{\boldsymbol{\theta}} \left(\sum_{v' \in V'} \Big(\log \big(\frac{1}{\sigma\sqrt{2\pi}} \big) + \log \big(e^{-\frac{1}{2}(\frac{a_t^{v'} - \mu_t^{v'}}{\sigma})^2} \big) \Big) \right) \\
&= \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t|s_t) \left(\sum_{v' \in V'} \nabla_{\boldsymbol{\theta}} \Big(-\frac{1}{2} (\frac{a_t^{v'} - \mu_t^{v'}}{\sigma})^2 \Big) \right) \\
&= -\frac{\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t|s_t)}{2\sigma^2} \Big(\sum_{v' \in V'} \nabla_{\boldsymbol{\theta}} (a_t^{v'} - \mu_t^{v'})^2 \Big) \\
&= \frac{\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t|s_t)}{\sigma^2} \Big(\sum_{v' \in V'} (a_t^{v'} - \mu_t^{v'}) \nabla_{\boldsymbol{\theta}} \mu_t^{v'} \Big)
\end{aligned} \tag{5.9}$$

Given

$$\mu_t^{v'} = \text{Softmax}(f_{\boldsymbol{\theta}}^p(z_t^{v'})) = \frac{e^{f_{\boldsymbol{\theta}}^p(z_t^{v'})}}{\sum_{i \in V'} e^{f_{\boldsymbol{\theta}}^p(z_t^i)}},$$

we have

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} \mu_t^{v'} &= \nabla_{\boldsymbol{\theta}} \text{Softmax}(f_{\boldsymbol{\theta}}^p(z_t^{v'})) \\
&= \frac{\big(e^{f_{\boldsymbol{\theta}}^p(z_t^{v'})} \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}^p(z_t^{v'}) \big) \big(\sum_{i \in V'} e^{f_{\boldsymbol{\theta}}^p(z_t^i)} \big) - e^{f_{\boldsymbol{\theta}}^p(z_t^{v'})} \big(\sum_{i \in V'} e^{f_{\boldsymbol{\theta}}^p(z_t^i)} \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}^p(z_t^i) \big)}{(\sum_{i \in V'} e^{f_{\boldsymbol{\theta}}^p(z_t^i)})^2} \\
&= \frac{e^{f_{\boldsymbol{\theta}}^p(z_t^{v'})}}{(\sum_{i \in V'} e^{f_{\boldsymbol{\theta}}^p(z_t^i)})^2} \Big(\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}^p(z_t^{v'}) \sum_{i \in V'} e^{f_{\boldsymbol{\theta}}^p(z_t^i)} - \sum_{i \in V'} e^{f_{\boldsymbol{\theta}}^p(z_t^i)} \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}^p(z_t^i) \Big) \\
&= \frac{e^{f_{\boldsymbol{\theta}}^p(z_t^{v'})}}{(\sum_{i \in V'} e^{f_{\boldsymbol{\theta}}^p(z_t^i)})^2} \sum_{i \in V'} e^{f_{\boldsymbol{\theta}}^p(z_t^i)} \Big(\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}^p(z_t^{v'}) - \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}^p(z_t^i) \Big)
\end{aligned} \tag{5.10}$$

where $\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}^p(z_t^{v'})$ is the gradient of the priority function (i.e., the DNN) in Figure 5.1.

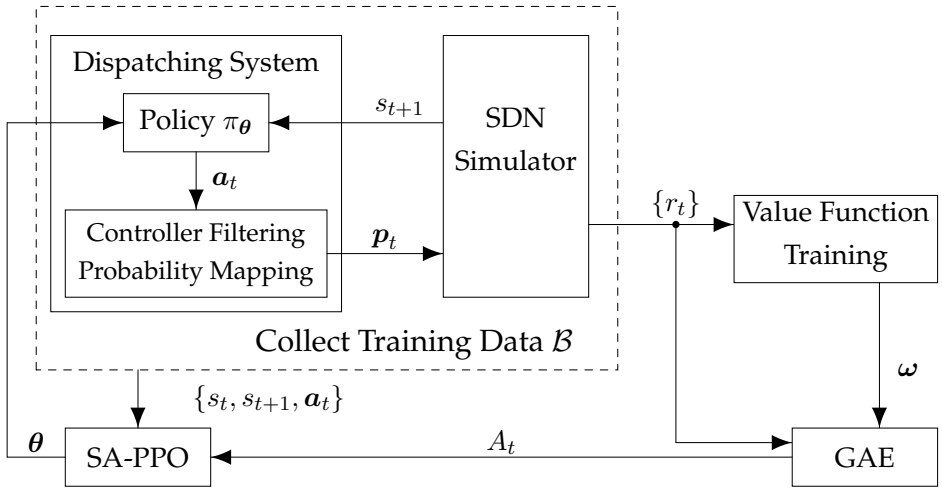


Figure 5.3: Training system design.

Summarizing the above discussions, with respect to a mini-batch $\mathcal{B}$, the policy gradient $\nabla_{\boldsymbol{\theta}}\mathcal{L}(\pi_{\boldsymbol{\theta}})$ is estimated by SA-PPO according to

$$\begin{aligned}\nabla_{\boldsymbol{\theta}}\mathcal{L}(\pi_{\boldsymbol{\theta}}) \approx & \frac{1}{\|\mathcal{B}\|}\sum_{\mathcal{B}}\frac{A_t(s_t,\boldsymbol{a}_t)}{\pi_{\boldsymbol{\theta}_{old}}(\boldsymbol{a}_t|s_t)}\cdot\frac{\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t|s_t)}{\sigma^2}\left(\sum_{v'\in V'}(a_t^{v'}-\mu_t^{v'})\cdot\right. \\ & \left.\frac{e^{f_{\boldsymbol{\theta}}^p(z_t^{v'})}}{(\sum_{i\in V'}e^{f_{\boldsymbol{\theta}}^p(z_t^i)})^2}\sum_{i\in V'}e^{f_{\boldsymbol{\theta}}^p(z_t^i)}\left(\nabla_{\boldsymbol{\theta}}f_{\boldsymbol{\theta}}^p(z_t^{v'})-\nabla_{\boldsymbol{\theta}}f_{\boldsymbol{\theta}}^p(z_t^i)\right)\right)\end{aligned} \tag{5.11}$$

According to PPO [242], the policy $\pi_{\boldsymbol{\theta}}$ can be improved by repeatedly updating the policy parameters $\boldsymbol{\theta}$ along the direction of $\nabla_{\boldsymbol{\theta}}\mathcal{L}(\pi_{\boldsymbol{\theta}})$. Note that this technique for calculating the policy gradient can be easily extended to the case with an arbitrary number of controllers. With the help of TensorFlow [21], the gradient calculation can also be fully automated in our training system, regardless of how many controllers are involved. The computational complexity is linear with respect to the number of controllers.

Algorithm 6 SA-PPO for Policy Training

Initialize value function $\mathcal{V}_{\boldsymbol{\omega}_{0,N_{epo}}}$ and policy $\pi_{\boldsymbol{\theta}_{0,N_{epo}}}$
for Each training iteration $ti = 1 : I_{\max}$ **do**
  Initialize replay buffer$= \varnothing$
  for Each episode $epi = 1 : N_{epi}$ **do**
    Network warm-up
    Observe initial state s_0
    for $t = 1, ..., t_{\max}$ **do**
      Select an action $\boldsymbol{a}_t = \pi_{\boldsymbol{\theta}_{ti-1,N_{epo}}}(s_t)$ as shown in Figure 5.1
      Execute action $\boldsymbol{a}_t$
      Receive reward r_t and observe new state s_{t+1}
      Store state transition $u = \langle s_t, s_{t+1}, \boldsymbol{a}_t, r_t \rangle$ in replay buffer
    end for
  end for
  Estimate advantages A_t^{π} using GAE
  for Each epoch $epo = 1, ..., N_{epo}$ **do**
    Sample random minibatch of transitions $\mathcal{B}$ from replay buffer
    Update the value function: $\boldsymbol{\omega}_{ti,epo+1} = \boldsymbol{\omega}_{ti,epo} + \alpha_{\boldsymbol{\omega}} \nabla_{\boldsymbol{\omega}} \mathcal{H}(\mathcal{V}_{\boldsymbol{\omega}})$ where $\mathcal{H}(\mathcal{V}_{\boldsymbol{\omega}})$ is defined in (5.8)
    Update the policy: $\boldsymbol{\theta}_{ti,epo+1} = \boldsymbol{\theta}_{ti,epo} + \alpha_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}} \mathcal{L}(\pi_{\boldsymbol{\theta}})$ where $\nabla_{\boldsymbol{\theta}} \mathcal{L}(\pi_{\boldsymbol{\theta}})$ is provided in (5.11)
  end for
end for

5.2.4.2 Training System Design

Along with the new gradient calculation technique, a training system is developed which simultaneously trains both the policy $\pi_{\boldsymbol{\theta}}$ and the value function $\mathcal{V}_{\boldsymbol{\omega}}$ as shown in Figure 5.3.

During the simulation, the network starts from an initial state where no packets have started to flow through the network. Then it enters a warm-up period when requests are dispatched using predefined policies (e.g.,

weighted round-robin). After the warm-up period, the DRL-designed policy takes over until the simulation time reaches a predefined value t_{max}. An episode is further defined as one network simulation which starts after the warm-up period until the end of the simulation. Simulation setting details are provided in Subsection 5.2.5.

As shown in Algorithm 6, the training is performed for $I_{\max}$ TIs. In each TI, the policy $\pi_{\boldsymbol{\theta}_{ti-1,N_{epo}}}$ that is updated from the last TI is used to guide request dispatching in a network for N_{epi} episodes (see line 8). For each episode, the network is simulated for $t_{\max}$ time steps. For each time step, the state-transition sample u defined in (5.7) is recorded in a memory replay buffer (see line 11). Given the replay buffer, SA-PPO is activated to train the policy $\pi_{\boldsymbol{\theta}}$ (see line 14-19). The updated policy is then applied in the next TI to generate state transition samples. This training process continues until the maximum number of TI $I_{\max}$ is reached.

5.2.5 Simulation

In this subsection, we first introduce the simulation setting which includes the algorithm implementation and the network simulation setting. After that, similar to [142], we investigate the influence of historical information and the discount factor γ on the performance of our policy respectively. To demonstrate the effectiveness of the new policy design (denoted as SA-PPO), we first present its training performance. After that, the new policy design is compared with the non-adaptive policy[31] (denoted as PPO). Specifically, NN configurations for SA-PPO and PPO are summarized in Table 5.2. For PPO, each output node corresponds to a separate SDN controller while SA-PPO has only one output node for a priority value. For the input layer, the state observation $z_t^{v'}$ with respect to Controller $C_{v'}$ is fed to the NN in SA-PPO (More details on selecting $z_t^{v'}$ will be discussed in Subsection 5.2.5.2). In comparison, observations from all controllers are

[31] A traditional policy design where the policy is directly represented as a DNN.

Table 5.2: NN architecture comparison between SA-PPO and PPO

NN layers	Input	Hidden	Output
SA-PPO	$z_t^{v'}$	Same	One priority $a_t^{v'}$
PPO	$[z_t^1, ..., z_t^N]$		N probabilities $[p_t^1, ..., p_t^N]$

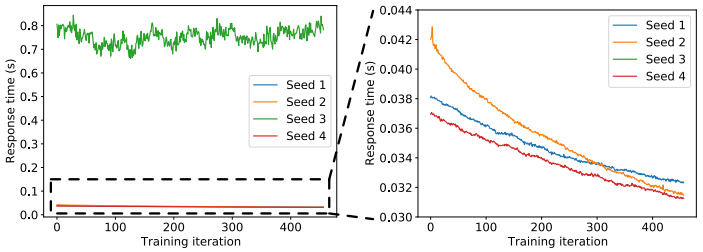
concatenated as inputs (i.e., $[z_t^1, ..., z_t^N]$) for the NN used in PPO.

5.2.5.1 Algorithm Implementation

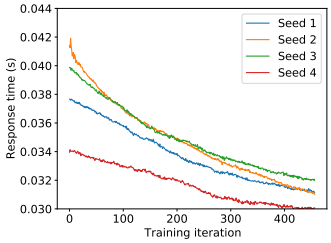
We implement SA-PPO based on the high-quality implementation of the training algorithm PPO [242] provided by OpenAI baselines[32]. In our simulation, we adopt the same NN architecture given in PPO for both the priority function $f_{\boldsymbol{\theta}}^p$ and value function $\mathcal{V}_{\boldsymbol{\omega}}$. Each NN is a fully connected multilayer feed forward neural network.

To investigate the impact of the NN architecture on the algorithm performance, NN settings with different numbers of hidden layers (ranging from 1 to 3) and nodes (32 and 64) are evaluated. As demonstrated in Figure 5.4(a) and 5.4(c), the behavior of an NN with one hidden layer changes with different random seeds, which implies that the single-hidden-layer NN is sensitive to the initial parameter settings. Apart from that, an NN with one hidden layer or a small number of hidden neurons (e.g., Figure 5.4(b)) may not be powerful/precise enough to capture the underlying relationship between the input and output [248]. On the other hand, NNs with too many nodes require more training data and may result in over-fitting. Figure 5.4(d) showed that an NN with two hidden layers of 64 units can achieve similar performance (in terms of response time and convergence speed) as an NN with a larger size (e.g., Figure 5.4(f)). Compared to Figure 5.4(e) where an NN with a similar size was evaluated, the NN used in Figure 5.4(d) can achieve lower response time after a same number of TIs with different seeds. Thus, to strike a good balance between

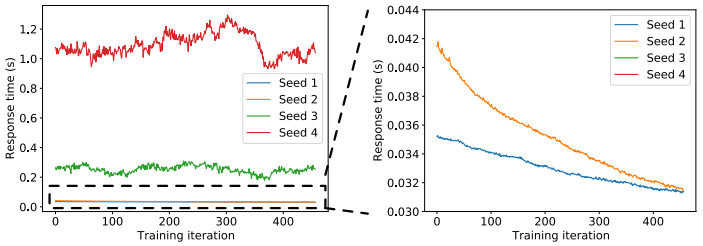
[32]https://github.com/openai/baselines



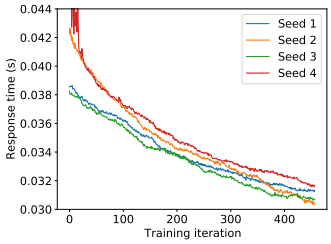
(a) 32 x 1 (left: overall figure, right: enlarged figure of the bottom area)



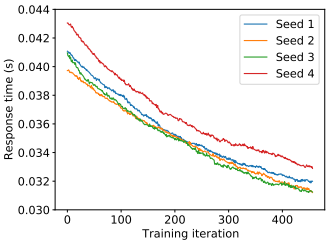
(b) 32 x 2



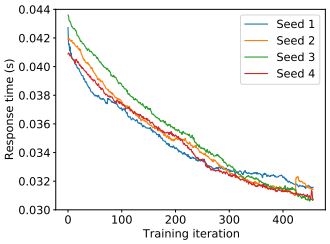
(c) 64 x 1 (left: overall figure, right: enlarged figure of the bottom area)



(d) 64 x 2



(e) 32 x 3



(f) 64 x 3

Figure 5.4: Performance comparison of different NN architectures. (a) - (f) show the average request response time changes as the training proceeds with respect to a specific NN architecture (number of nodes in each layer x number of layers).

the architecture complexity and the convergence speed, an NN with two hidden layers of 64 units is used in our simulation.

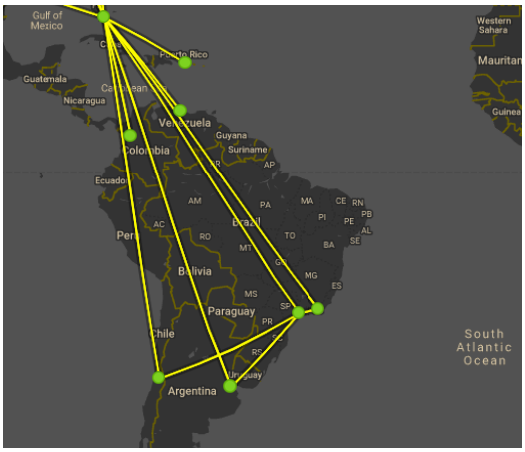
Meanwhile, our RDPD problem can be considered as a continuous control task that is similar to MuJoCo benchmarks in PPO [242]. Therefore, we closely follow its hyper-parameter settings. However, there are a few exceptions. Specifically, the Gaussian noises $\boldsymbol{\epsilon}_t$ in (5.5) have their standard deviations set to 0.01. During every algorithm run, the policy is trained for 900 TIs. Both $\boldsymbol{\theta}$ and $\boldsymbol{\omega}$ in Figure 5.3 are trained using data sampled from the current TI. The NN parameters $\boldsymbol{\theta}$ and $\boldsymbol{\omega}$ are updated using Adam optimizer with 3×10^{-4} learning rate, 40 minibatch size, and 8 epochs.

5.2.5.2 Network Simulation Setting

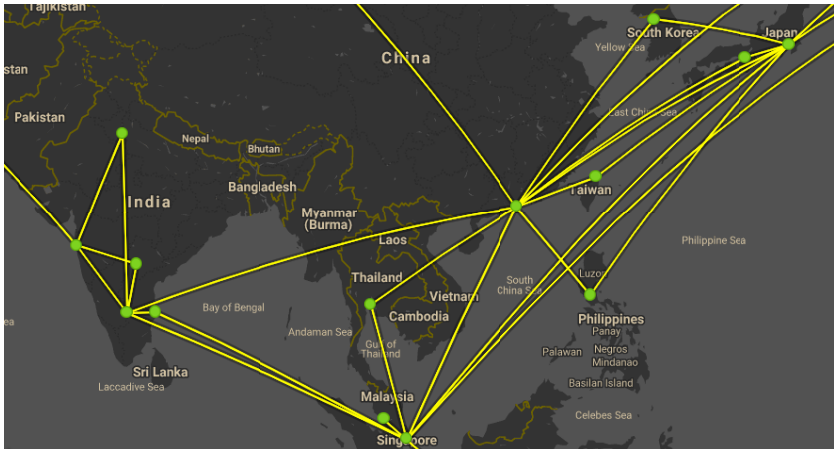
Simulations are conducted under the real network topologies provided by Sprint [19]: South America and Asia Sprint networks equipped with 8 and 14 switch centers respectively, as shown in Figure 5.5.

A set of heterogeneous controllers with capacities ranging from 6000 pkt/s to 9000 pkt/s have been deployed into the network with the help of our CPP algorithm in Chapter 4. Unless we explicitly specify, the numbers of controllers deployed in the South America and Asia networks are 3 and 4 respectively during the simulation. The location of the centralized request dispatching agent is selected so that the average propagation latency between the agent and all controllers is minimized.

Each episode is initialized with 0% utilization for all controllers and 0 packets in the network. Then the requests are generated following the Poisson distributions with a predefined arrival rate and sent to the scheduler/agent. Note that our MDP model in Subsection 5.2.1 does not assume that requests are generated following a specific distribution. The use of Poisson distribution in the simulation is to enable the comparison between GD-based scheduling algorithm proposed in Subsection 4.3.2 and MA-PPO as demonstrated in Subsection 5.3.3. Weighted round robin is used to make request dispatching decisions during the warm-up period.



(a) South America



(b) Asia

Figure 5.5: Network topologies used in simulation studies, obtained from Sprint [19].

The warm-up period lasts for 30 simulated seconds which is assumed to be sufficiently long for the network to enter and stay in a stationary condition.

Each simulation episode runs for 30 simulated minutes which is divided into a series of time steps. Every time step lasts for 30 consecutive simulated seconds. At the beginning of each time step, the agent executes

the policy to calculate the priority of dispatching any new requests to each controller in the network for the next time step, i.e., 30 simulated seconds.

To enable the agent to learn how to dispatch requests under different workloads, two episodes with two request arrival rates are simulated in each TI. In particular, for the low workload setting, the combined request arrival rate from all switches is set to be 50% of the total control plane capacity while the arrival rate under high workload is 80%.

For the policy to work properly, the centralized agent must provide its state observations $z_t^{v'}$ with respect to each controller $C_{v'}$ one-by-one to the priority function $f_{\boldsymbol{\theta}}^p$ in Figure 5.1. In consideration of the importance of controllers' capacities, their distance and current availability, as well as the communication demand experienced by the agent, the local observation $z_t^{v'}$ with respect to $C_{v'}$ consists of the following network statistics:

(1) Request arrival rate history from the whole data plane: $\sum_{v \in V} \lambda_v$;

(2) The processing capacity of $C_{v'}$: $\alpha_{v'}$;

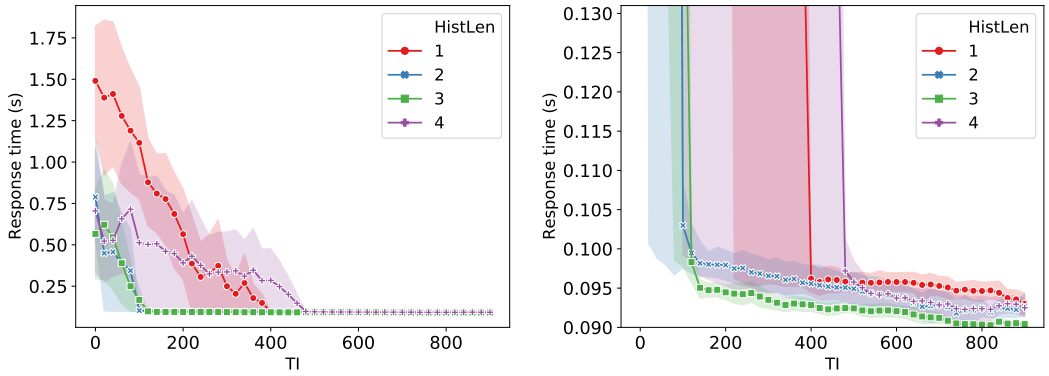
(3) The propagation latency between the agent deployed at node v and $C_{v'}$: $D(v, v')$;

(4) The queue length of $C_{v'}$;

(5) The number of requests sent from the agent to $C_{v'}$ during the previous time step;

(6) The total number of requests that $C_{v'}$ receives from the data plane during the previous time step.

In practice, the request arrival history is made up of a list of request arrival rates measured in the past few time steps by the agent. Intuitively, the longer the list, the easier it is for the agent to detect traffic change patterns and adjusts its request dispatching in consideration of future communication demand. Moreover, a state signal with a longer historical list provides more information of the past, which can better fulfill the Markov



(a) (b) Enlarged bottom area of (a)

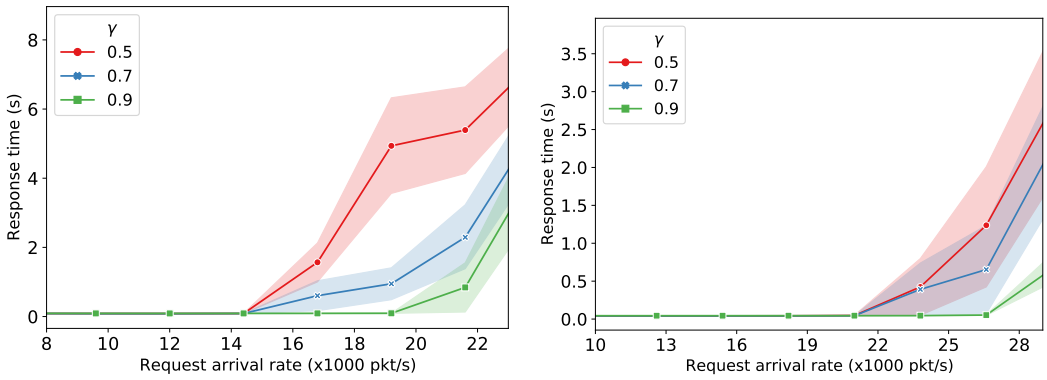
Figure 5.6: Influence of historical information.

property. The impact of the history length will be investigated in the next subsection.

Similar to $z_t^{v'}$, the global state s_t contains the arrival rate history from the data plane, all controllers' processing capacity, all controllers' queue length, and the propagation latency measured in $\boldsymbol{D}$.

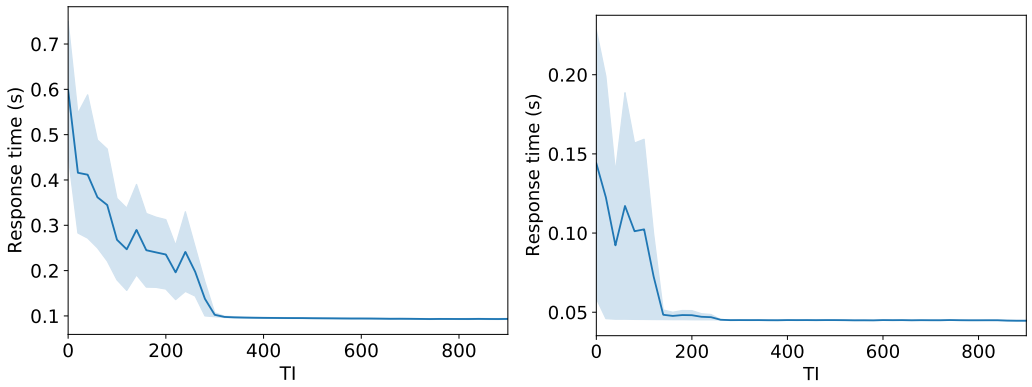
5.2.5.3 Simulation Result

The performance impact of history length: Similar to [142], we investigate the influence of historical information on the performance of our policy. As shown in Figure 5.6, regarding the list of historical request arrival rates contained in the agent's observation, its length needs to be set properly. With a larger history length, more information of the past is included in the agent's observation, which provides a better approximation of a Markov state. However, if the length is too large (e.g., 4), more learning samples are required for the network to improve its performance. On the other hand, when the length is too small (e.g., 1), the response time stops reducing after 400 TI. It appears that the most suitable length is 3 in our simulations for a good trade-off between sampling costs and perfor-



(a) South America (b) Asia

Figure 5.7: Influence of γ.



(a) South America (b) Asia

Figure 5.8: Training performance of SA-PPO.

mance.

The performance impact of γ: We also investigate the influence of γ on the performance of our policy. Figure 5.7 demonstrates the evaluation of the trained policies with different γ under a broad range of request arrival rates in two topologies. From Figure 5.7, we can see that the policy with $\gamma = 0.9$ consistently achieves the lowest response time compared to

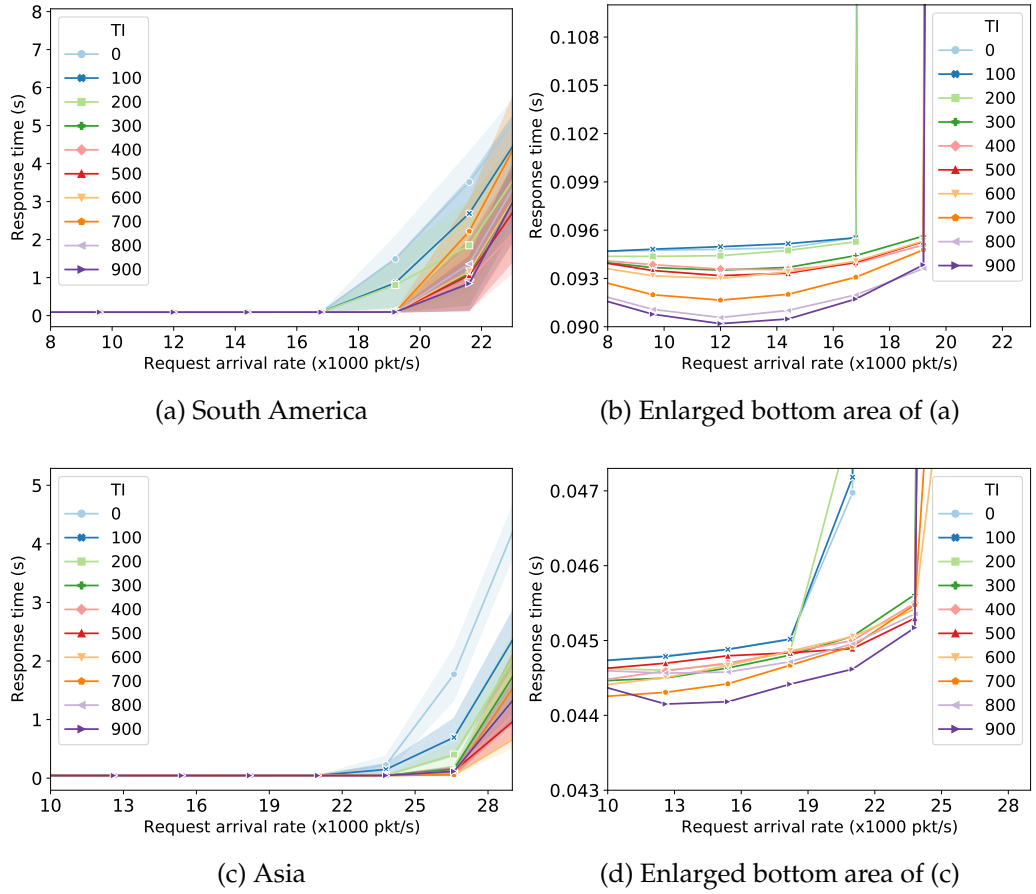
policies with $\gamma = 0.5$ and $\gamma = 0.7$ in both topologies. It confirms our theory that the agent should consider the influence of its actions on future network performance, which is vital to prevent any controllers from being overloaded due to accumulated requests over a long run. Thus, unless we explicitly point out, for the remaining simulation studies, γ is fixed to 0.9. Apart from that, we also observe that as the request arrival rate exceeds a certain value, the response time of all policies increases sharply regardless of the γ values. This is mainly due to the highly loaded control plane.

Training effectiveness: Figure 5.8 demonstrates how the network performance of our policy improves during the training process in two topologies. In general, we can see that the response time in both topologies quickly drops at the initial training stage and then converges. Specifically, a sharp decrease in the response time from around 0.6 s to less than 0.1 s within 300 TIs can be observed from Figure 5.8(a). Similar patterns can also be observed in Figure 5.8(b).

We also evaluate the policies obtained at different TIs as shown in Figure 5.9. In general, we can observe from Figure 5.9(a) and Figure 5.9(c) that in both topologies, all policies achieve low response time when the request arrival rate remains low, which is understandable because controllers are not likely to be overloaded. Nevertheless, in the enlarged figures (i.e., Figure 5.9(b) and Figure 5.9(d)), we can still spot the performance improvement in terms of the reduced response time as the training proceeds (i.e., the number of TIs increases).

Moreover, as the arrival rate grows (e.g., when the arrival rate is up to 20k pkt/s in Figure 5.9(a)), a sharp jump can be observed for the policies obtained at the earlier TIs while the latter policies can still maintain a low response time. Similar patterns can also be found in the Asia network (i.e., Figure 5.9(c)), which together with Figure 5.8 demonstrates the effectiveness of our policy training.

Policy design comparison: As shown in Figure 5.10(b) and Figure 5.10(d), our adaptive policy design SA-PPO achieves similar per-



(a) South America

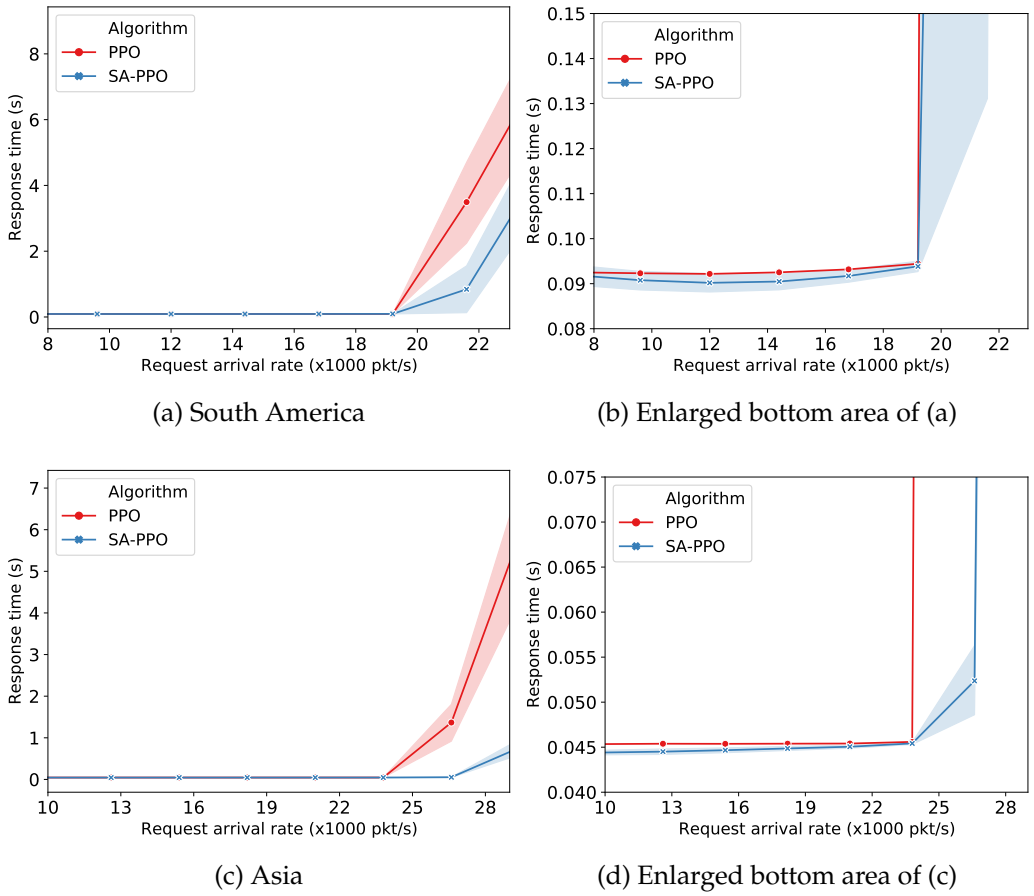
(b) Enlarged bottom area of (a)

(c) Asia

(d) Enlarged bottom area of (c)

Figure 5.9: Testing performance comparison during the learning process under different request arrival rates in different network topologies.

formance as PPO under low request arrival rates. However, from Figure 5.10(a), we can spot a sudden growth in response time for PPO as the request arrival rate increases while the response time of SA-PPO remains low. This is mainly because in the traditional policy representation, the NN is designed to directly output the request dispatching probabilities over all controllers given all controllers' state information as listed in Table 5.2. On the other hand, the NN used in SA-PPO is designed to estimate



(a) South America

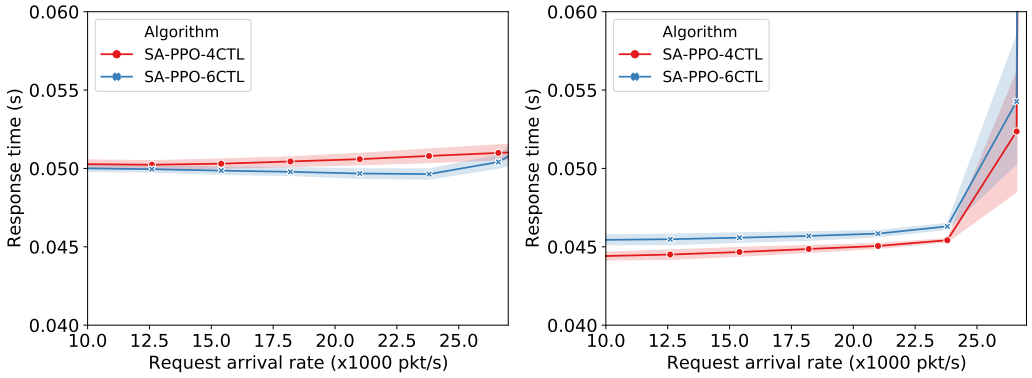
(b) Enlarged bottom area of (a)

(c) Asia

(d) Enlarged bottom area of (c)

Figure 5.10: Performance comparison of different algorithms in different network topologies.

a priority value with respect to one controller using the controller's state information. Given the larger dimensions of both inputs and outputs, the mapping learned in PPO is more complicated than in SA-PPO. Therefore, an NN with the same hidden layer configuration as SA-PPO may not be powerful enough to capture the mapping. This can be further evidenced in Asia topology with more network nodes (Figure 5.10(d)) where the performance difference at high request arrival rates is more significant than in



(a) Testing in 6CTL

(b) Testing in 4CTL

Figure 5.11: Performance comparison of policies on networks with different numbers of controllers. XCTL refers to a network with X controllers and "SA-PPO-XCTL" represents the policy trained by SA-PPO in XCTL.

the South America topology. Therefore, our new policy design can reduce the NN complexity without performance compromise.

Policy adaptiveness: Although our policy was only trained under two different workloads (50% and 80%), it can perform consistently well under different workloads, ranging from 30% up to 90% as demonstrated in Figure 5.10.

To demonstrate the adaptiveness of our trained policy with respect to different numbers of controllers, the policy trained in a network with 4 controllers (SA-PPO-4CTL) is evaluated in a network with 6 controllers. Its performance is compared with the policy trained with 6 controllers (SA-PPO-6CTL). From Figure 5.11(a), we can see that SA-PPO-4CTL can achieve similar performance compared to SA-PPO-6CTL. Similar conclusions can also be drawn from Figure 5.11(b) where SA-PPO-6CTL is compared with SA-PPO-4CTL in a network with 4 controllers. Our simulation results confirm that the policy can perform consistently well in networks with changing numbers of controllers.

Summary: From our simulation results, we can learn that:

(1) The performance of SA-PPO was affected by the amount of historical information provided in the agent's observation. The more historical information is given, the better the Markov state can be approximated, which further improves the network performance but also requires more training samples.

(2) The investigation of the discount factor γ showed that it is important to consider the influence of the agent's actions on future network performance.

(3) Compared with the traditional RL policy representation, our new policy design can reduce the NN architecture complexity while maintaining good performance and being adaptive to different numbers of controllers.

5.3 Multi-Agent Deep Reinforcement Learning for Request Dispatching

Although the policy design process can be modeled as SA-DRL, such a central approach may not be scalable [300]. This is mainly because the centralized agent can be a performance bottleneck and overwhelmed by the enormous traffic. Apart from that, it also introduces additional propagation latencies as all requests need to go through the centralized agent as we discussed in Section 5.1. Furthermore, the use of global information over the entire network in SA-DRL can cause substantial communication overhead and sometimes may not even be available. Thus, the SA-DRL approach is more suitable for small networks with relatively small geographic coverage.

To address the issues in SA-DRL, a new MA-DRL approach for learning adaptive policies for SDN switches is proposed in this section where multiple agents are involved. Rather than using a centralized agent, we

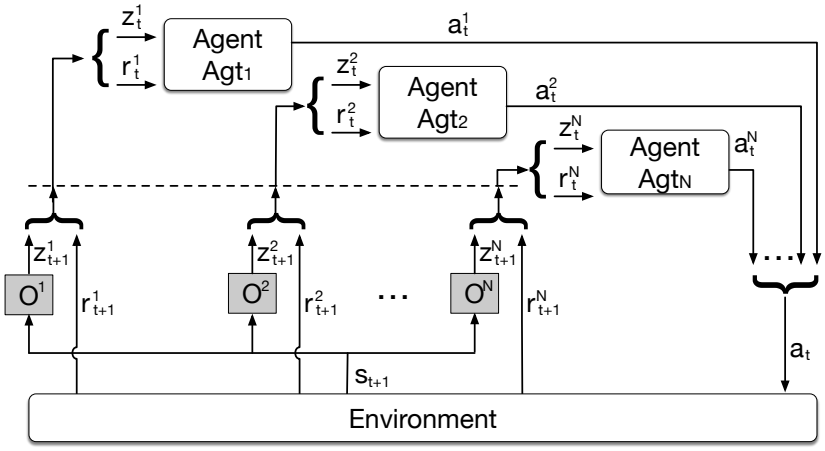


Figure 5.12: Modeling the RDPD problem as an MA-MDP.

equip each switch with a co-located agent and each agent makes request dispatching decisions for one switch. Without assuming fully observable agents supported by global network knowledge, a partially observable network is considered where each agent executes its policy based only on its local network observation.

Following the structures in Section 5.2, this section starts with demonstrating the mapping from the RDPD problem to a fully cooperative and partially observable MA-MDP in Subsection 5.3.1. Then a training algorithm called MA-PPO is proposed in Subsection 5.3.2. Subsection 5.3.3 presents the simulation results.

5.3.1 Modeling the RDPD Problem as an MA-MDP

In this subsection, the RDPD problem is modeled as a *fully cooperative* and *partially observable* MA-MDP with M agents that control the M SDN switches respectively. As shown in Figure 5.12, the overall network operating status is captured by a set of global states $\mathcal{S}$. At each time step t, every agent Agt_v receives a local observation $z_t^{v,v'}$ with respect to each controller $C_{v'}$. The relationship between $z_t^{v,v'}$ and s_t is determined by the agent's observation function $z_t^{v,v'} = \mathcal{O}^v(s_t, v')$. Based on their local obser-

vations, every agent Agt_v issues an action $a_t^v \in \mathcal{A}^v$ chosen from its policy $\pi_{\boldsymbol{\theta}^v}$ to jointly form the multi-agent action $\{a_t^v\}_{v \in V}$. Here $a_t^v = \{a_t^{v,v'}\}_{v' \in V'}$ specifies the priority $a_t^{v,v'}$ for Agt_v to dispatch its new requests to any controller $C_{v'}$ during time t and $t+1$. As a result of following the joint action, each agent obtains a reward r_t^v based on the responses it received from all controllers during time t and $t+1$. In line with the reward defined in SA-MDP, the reward r_t^v received by Agt_v is defined as below:

$$r_t^v = \varsigma X_t^v - \sum_{x=1}^{X_t^v} \tau_x^v, \tag{5.12}$$

where X_t^v is the total number of responses received between t and $t+1$ by agent Agt_v and τ_x^v is the response time of a particular request. ς is a *weight factor*. Clearly, all agents prefer to receive more responses with less request response time according to (5.12). For this purpose, each agent Agt_v learns one policy $\pi_{\boldsymbol{\theta}^v}$ parameterized by $\boldsymbol{\theta}^v$ with the adaptive design shown in Figure 5.1 that maps observation z_t^v to its action a_t^v. The goal of MA-MDP is hence to identify the optimal policies $\{\pi_{\boldsymbol{\theta}^v}^*\}_{v \in V}$ so as to maximize the *expected joint cumulative rewards*:

$$J(\{\pi_{\boldsymbol{\theta}^v}\}_{v \in V}) = \mathbb{E}_{\{a_t^v \sim \pi_{\boldsymbol{\theta}^v}\}_{v \in V}} \sum_{t=0}^{T} \gamma^t \sum_{v \in V} r_t^v(\{z_t^{v,v'}\}_{v' \in V'}, a_t^v) \tag{5.13}$$

5.3.2 MA-PPO for Adaptive Policy Training

Aiming at training a policy for each SDN switch in a network, one straightforward approach is to directly adopt the SA-DRL algorithm (e.g., SA-PPO in Subsection 5.2.4) for policy training. One DRL agent is placed on every SDN switch to continuously and independently learn its policy using SA-PPO while the other agents are treated as part of the environment. Despite its simplicity, the training process is vulnerable to the *non-stationary* environment problem [188]. In particular, the reward received by each agent and the global state transition do not depend solely on one agent's

individual actions. Instead, they are affected by the joint actions from all agents. Moreover, each agent's policy keeps being updated independently during the training process. Therefore, the environment observed by each agent becomes non-stationary (i.e., violating the Markov property in Subsection 2.2.6), affecting the convergence of SA-DRL algorithms [188]. Evaluation of the SA learning approach (denoted by SA-PPO-MA) will be reported in Subsection 5.3.3.2.

Without pursuing a learning system with SA-DRL any further, Multi-Agent Proximal Policy Optimisation (MA-PPO) is developed to fulfill the general principle of centralized training and decentralized execution [188], which is essential for reliable MA-DRL. MA-PPO is a multi-agent extension of SA-PPO. Its design follows the same idea as MADDPG [188], which extends DDPG [181] to the multi-agent context. MADDPG needs to learn a centralized Q-function with large function input that contains the multi-agent joint action space. In comparison to MADDPG, MA-PPO is more suitable for training policies since policy training relies only on the V-function with substantially reduced input dimensions. The V-function in MA-PPO is much easier to model as a DNN than the centrally trained Q-function in MADDPG. In view of this, we decide to use MA-PPO to train policies.

Similar to SA-PPO, experience replay is adopted where each state-transition sample u records both global states and agents' local observations:

$$u = \left\langle s_t, s_{t+1}, \{\boldsymbol{z}_t^v\}_{v\in V}, \{\boldsymbol{z}_{t+1}^v\}_{v\in V}, \{\boldsymbol{a}_t^v\}_{v\in V}, \{r_t^v\}_{v\in V} \right\rangle \tag{5.14}$$

where $\boldsymbol{z}_t^v = \{z_t^{v,v'}\}_{v'\in V'}$, $\boldsymbol{z}_{t+1}^v = \{z_{t+1}^{v,v'}\}_{v'\in V'}$, and $\boldsymbol{a}_t^v = \{a_t^{v,v'}\}_{v'\in V'}$.

Given the experience replay buffer, $\mathcal{V}_{\boldsymbol{\omega}}$ is trained using the mini-batches of samples to minimize the *Bellman loss*:

$$\mathcal{H}(\mathcal{V}_{\boldsymbol{\omega}}) = \frac{1}{\|\mathcal{B}\|} \sum_{\mathcal{B}} \left(\mathcal{V}_{\boldsymbol{\omega}}(s_t) - \sum_{v\in V} r_t^v - \gamma \mathcal{V}_{\boldsymbol{\omega}}(s_{t+1}) \right)^2 \tag{5.15}$$

In SA-PPO-MA, each agent simultaneously and independently learns its

own value function using local observation, which brings about the non-stationary environment issue. To mitigate this issue, MA-PPO learns a centrally maintained parametric value function $\mathcal{V}_{\boldsymbol{\omega}}$ with global state input $s_t \in \mathcal{S}$. The value function $\mathcal{V}_{\boldsymbol{\omega}}$ is then shared by all agents.

Guided by the trained $\mathcal{V}_{\boldsymbol{\omega}}$, each agent in MA-PPO continues to use the sampled mini-batches to update its policy $\pi_{\boldsymbol{\theta}^v}$ parameterized by $\boldsymbol{\theta}^v$ along the direction of estimated policy gradient $\nabla_{\boldsymbol{\theta}^v}\mathcal{L}(\pi_{\boldsymbol{\theta}^v})$. Similar to SA-PPO, $\nabla_{\boldsymbol{\theta}^v}\mathcal{L}(\pi_{\boldsymbol{\theta}^v})$ can be calculated as:

$$\nabla_{\boldsymbol{\theta}^v}\mathcal{L}(\pi_{\boldsymbol{\theta}^v}) \approx \frac{1}{\|\mathcal{B}\|}\sum_{\mathcal{B}}\frac{A_t(s_t,\{\boldsymbol{a}_t^i\}_{i\in V})}{\pi_{\boldsymbol{\theta}^v_{old}}(\boldsymbol{a}_t^v|\boldsymbol{z}_t^v)}\nabla_{\boldsymbol{\theta}^v}\pi_{\boldsymbol{\theta}^v}(\boldsymbol{a}_t^v|\boldsymbol{z}_t^v) \tag{5.16}$$

provided that $\frac{\pi_{\boldsymbol{\theta}^v}}{\pi_{\boldsymbol{\theta}^v_{old}}}$ falls in the range $(-\infty, 1+0.1)$ if $A_t(s_t,\{\boldsymbol{a}_t^i\}_{i\in V})) > 0$ or $(1-0.1,+\infty)$ if $A_t(s_t,\{\boldsymbol{a}_t^i\}_{i\in V})) < 0$, with respect to any $\{\boldsymbol{a}_t^i\}_{i\in V}$ and s_t. Otherwise, $\nabla_{\boldsymbol{\theta}^v}\mathcal{L}(\pi_{\boldsymbol{\theta}^v}) = 0$. In particular, $\nabla_{\boldsymbol{\theta}^v}\pi_{\boldsymbol{\theta}^v}(\boldsymbol{a}_t^v|\boldsymbol{z}_t^v)$ can be calculated using (5.9) and (5.10) given $\boldsymbol{a}_t^v$ and $\boldsymbol{z}_t^v$ from sample u_t.

The overall training process is summarized in Algorithm 7.

5.3.3 Simulation

To examine the performance of MA-PPO for automatic training of adaptive policies, network simulations have been carried out.

The policy trained by MA-PPO is compared with the policy obtained by each agent independently training its policy with SA-PPO in an MA setting (denoted by SA-PPO-MA) as we discussed in Subsection 5.3.2.

Apart from that, the policy trained by MA-PPO is also compared with a widely used man-made policy (denoted by CWRR which is introduced in Subsection 4.3.1), a GD-based policy (denoted by GD which is introduced in Subsection 4.3.2), and the centralized policy trained by SA-PPO (denoted by SA-PPO which is introduced in Subsection 5.2.4). All the algorithms that are evaluated in this subsection and their defining properties are summarized in Table 5.3.

Algorithm 7 MA-PPO for Policy Training

Initialize value function $\mathcal{V}_{\boldsymbol{\omega}_{0,N_{epo}}}$
Initialize M policies $\{\pi_{\boldsymbol{\theta}^v_{0,N_{epo}}}\}_{v\in V}$ for M switches
for Each training iteration $ti = 1 : I_{\max}$ **do**
  Initialize replay buffer$= \varnothing$
  for Each episode $epi = 1 : N_{epi}$ **do**
    Network warm-up
    Network enters initial state s_0
    Each agent Agt_v observes its local observation $\boldsymbol{z}^v_t, \forall v \in V$
    for $t = 1, ..., t_{\max}$ **do**
      Each agent Agt_v selects an action $\boldsymbol{a}^v_t = \pi_{\boldsymbol{\theta}^v_{ti-1,N_{epo}}}(\boldsymbol{z}^v_t)$ to jointly form the multi-agent action $\{\boldsymbol{a}^v_t\}_{v\in V}$
      Execute the joint action $\{\boldsymbol{a}^v_t\}_{v\in V}$
      Network enters new state s_{t+1}
      Each agent Agt_v receives a reward r^v_t and new observation $\boldsymbol{z}^v_{t+1}$
      Store state transition $u = \langle s_t, s_{t+1}, \{\boldsymbol{z}^v_t\}_{v\in V}, \{\boldsymbol{z}^v_{t+1}\}_{v\in V}, \{\boldsymbol{a}^v_t\}_{v\in V}, \{r^v_t\}_{v\in V}\rangle$ in replay buffer
    end for
  end for
  Estimate advantages A^π_t using GAE
  for Each epoch $epo = 1, ..., N_{epo}$ **do**
    Sample random minibatch of transitions $\mathcal{B}$ from replay buffer
    Update the centralized value function: $\boldsymbol{\omega}_{ti,epo+1} = \boldsymbol{\omega}_{ti,epo} + \alpha_{\boldsymbol{\omega}} \nabla_{\boldsymbol{\omega}} \mathcal{H}(\mathcal{V}_{\boldsymbol{\omega}})$ where $\mathcal{H}(\mathcal{V}_{\boldsymbol{\omega}})$ is defined in (5.15)
    Each agent updates its policy: $\boldsymbol{\theta}^v_{ti,epo+1} = \boldsymbol{\theta}^v_{ti,epo} + \alpha_{\boldsymbol{\theta}^v} \nabla_{\boldsymbol{\theta}^v} \mathcal{L}(\pi_{\boldsymbol{\theta}^v})$ where $\nabla_{\boldsymbol{\theta}^v} \mathcal{L}(\pi_{\boldsymbol{\theta}^v})$ is provided in (5.16)
  end for
end for

Table 5.3: An overview of the evaluated algorithms. Training refers to whether the DRL policy is trained using SA-DRL or MA-DRL. Decision making refers to whether the CS decision is made in a centralized or distributed way. Scheduling Implementation refers to how the CS decisions are enforced.

Algorithm	Training	Decision Making	Scheduling Implementation	Required Global Information	Section
CWRR	N/A	Distributed[33]	Distributed[34]	✗	Subsection 4.3.1
GD	N/A	Centralized[35]	Distributed	✓	Subsection 4.3.2
SA-PPO	SA	Centralized	Centralized[36]	✓	Subsection 5.2.4
SA-PPO-MA	SA	Distributed	Distributed	✗	Subsection 5.3.2
MA-PPO	MA	Distributed	Distributed	✗	Subsection 5.3.2

In terms of comparisons with other MA-DRL algorithms, MA-DDPG is closely related to MA-PPO. However, deterministic policy gradient used in MA-DDPG cannot be calculated with respect to our adaptive policy network design, rendering MA-DDPG inapplicable. Apart from that, the new technique developed in Subsection 5.2.4.1 to compute the gradient of our policy network can be utilized by any AC algorithms designed for training stochastic policies such as TRPO [240] and Asynchronous Advantage Actor-Critic (A3C) [201].

However, investigating the performance of different AC algorithms is not the main focus of this thesis. Furthermore, compared to PPO, TRPO has high computation complexity due to its use of both linear approximation of the learning objective and quadratic approximation of the constraint for policy update [59, 240]. On the other hand, A3C asynchronously executes multiple actors where each actor interacts with its own copy of

[33]Each decision making instance calculates the scheduling strategy for one switch.

[34]The CS decisions are enforced by multiple schedulers.

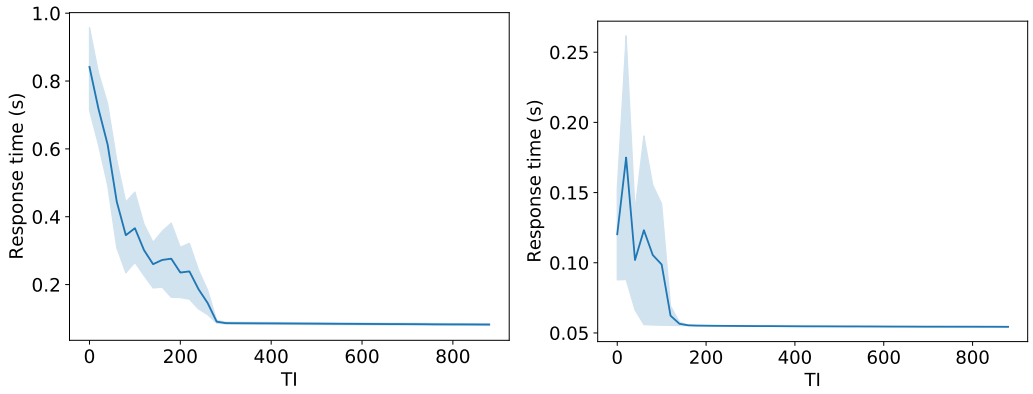
[35]A centralized decision making instance calculates the scheduling strategy for the entire data plane.

[36]The CS decisions are enforced by a centralized scheduler.

the environment. The rewards collected from all actors are used to update the shared policy and value function. However, due to the use of multiple actors, A3C requires more computation resources. Apart from that, the policy updates in A3C rely on the latest data collected from multiple actors without using memory replay, which results in high sampling costs. Therefore, both TRPO and A3C are not as suitable as PPO. Moreover, as a representative algorithm among all AC algorithms, studying the performance of PPO gives us an overall good understanding of other AC algorithms. In the future, combining our new policy design with different AC algorithms in an MA setting will be investigated when enough computation resources and time are provided.

5.3.3.1 Simulation Setting

We follow closely the algorithm implementation and network simulation setting in Subsection 5.2.5.1 and Subsection 5.2.5.2. However, there are a few exceptions for SA-PPO-MA and MA-PPO. Specifically, unlike SA-PPO where a centralized agent is selected for the whole network, both SA-PPO-MA and MA-PPO deploy a separate agent for each switch center in the network. At the beginning of any time step, all agents run their policies individually to calculate their respective priorities of dispatching any new requests to each controller in the network for the next time step. Apart from that, instead of providing the global state information to the policy, only local observation $z_t^{v,v'}$ with respect to the switch center Sw_v where agent Agt_v is placed on and controller $C_{v'}$ is available. For example, instead of the request arrival rate history of the whole data plane, only the arrival rate history of the switch center Sw_v is provided. Similarly, only the number of requests sent from Sw_v to $C_{v'}$ during the previous time step is available in $z_t^{v,v'}$.



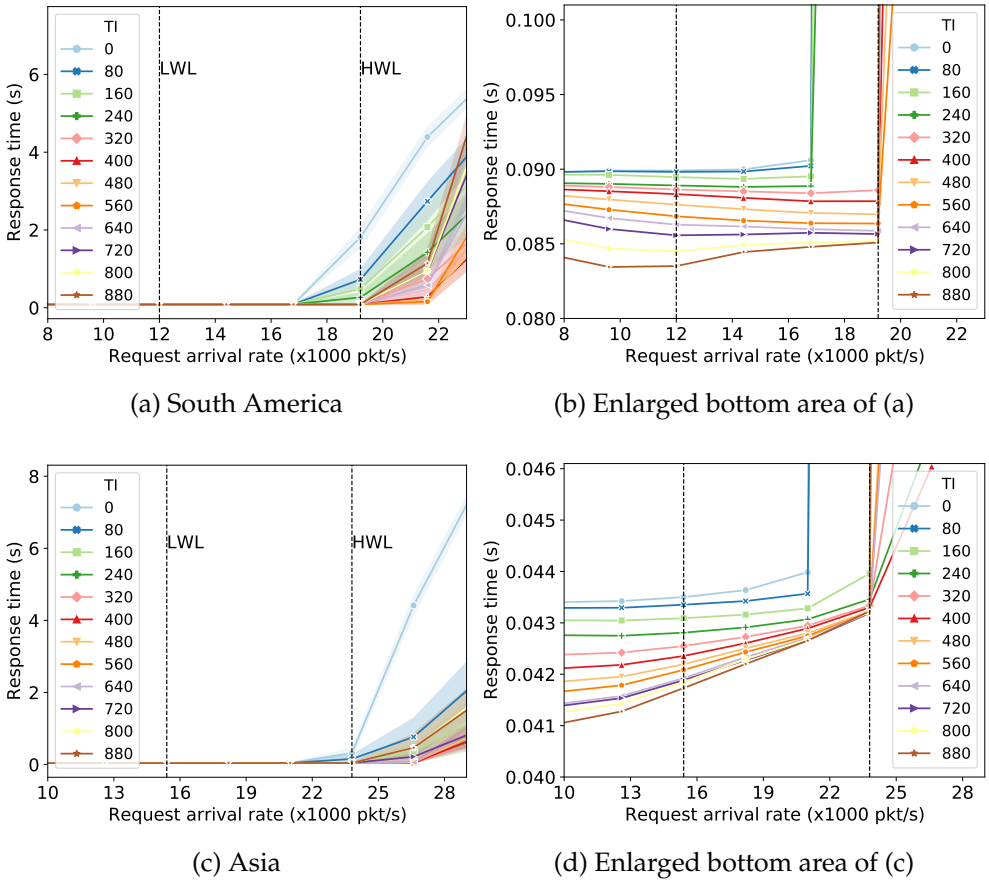
(a) South America (b) Asia

Figure 5.13: Training performance of MA-DRL.

5.3.3.2 Simulation Result

Training effectiveness: Following the evaluation of SA-PPO, we study the training performance of MA-PPO in two topologies. The learning curves showing the average response time across TIs are demonstrated in Figure 5.13. Similar to SA-PPO, the response time can rapidly converge in both topologies.

We also investigate how the performance improves as the training proceeds at different TIs, as shown in Figure 5.14. It can be observed that the policies obtained at the later TIs achieve lower response time compared to those at the earlier TIs, which implies that MA-PPO can effectively improve the performance with continued training of the policy. For example, in Figure 5.14(a) and Figure 5.14(b), the response time of the initialized policies (i.e., TI=0) jumps from 90 ms to 2 s when the arrival rate reaches 19k pkt/s. This is mainly because when the policy is randomly initialized, its behaviors are similar to a random policy which equally distributes requests among all controllers. Therefore, controllers with lower capacities are easily overloaded, resulting in high response time. In comparison, the policies obtained after 320 TIs keep the response time below 1 s under the



(a) South America

(b) Enlarged bottom area of (a)

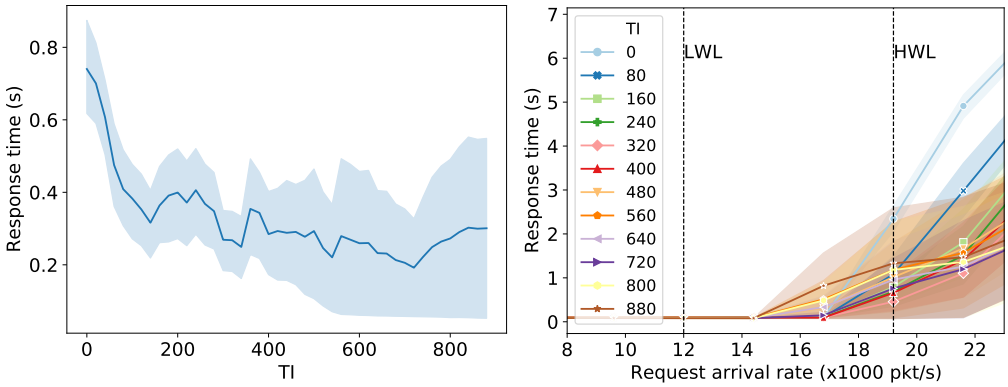
(c) Asia

(d) Enlarged bottom area of (c)

Figure 5.14: Testing performance comparison during the learning process under different request arrival rates in two topologies. In particular, "LWL" and "HWL" indicate the two request arrival rates (50% and 80% of the total control plane capacity) used during the training.

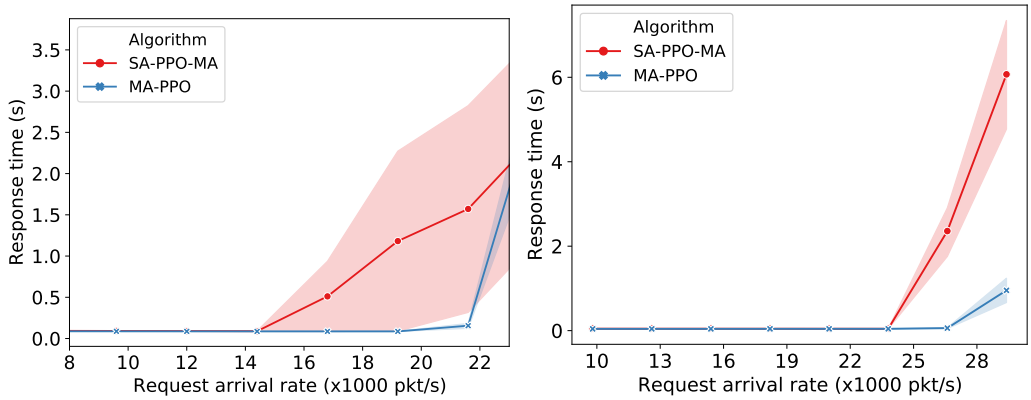
same arrival rate. Apart from avoiding overloading controllers, the training also consistently reduces the response time when the request arrival rate is low. Similar patterns can also be observed from Figure 5.14(c) and Figure 5.14(d).

SA vs. MA training in MA-DRL: To demonstrate the necessity of MA training, we compared training performance between MA-PPO and



(a) Training (b) Testing

Figure 5.15: Training and testing performance of SA-PPO-MA in South America Network.



(a) South America (b) Asia

Figure 5.16: Performance comparison between MA-PPO and SA-PPO-MA.

SA-PPO-MA. As discussed in Subsection 5.3.2, SA-PPO-MA trains a policy and a value function on each agent independently using SA-PPO. The training and testing performance of SA-PPO-MA are shown in Fig-

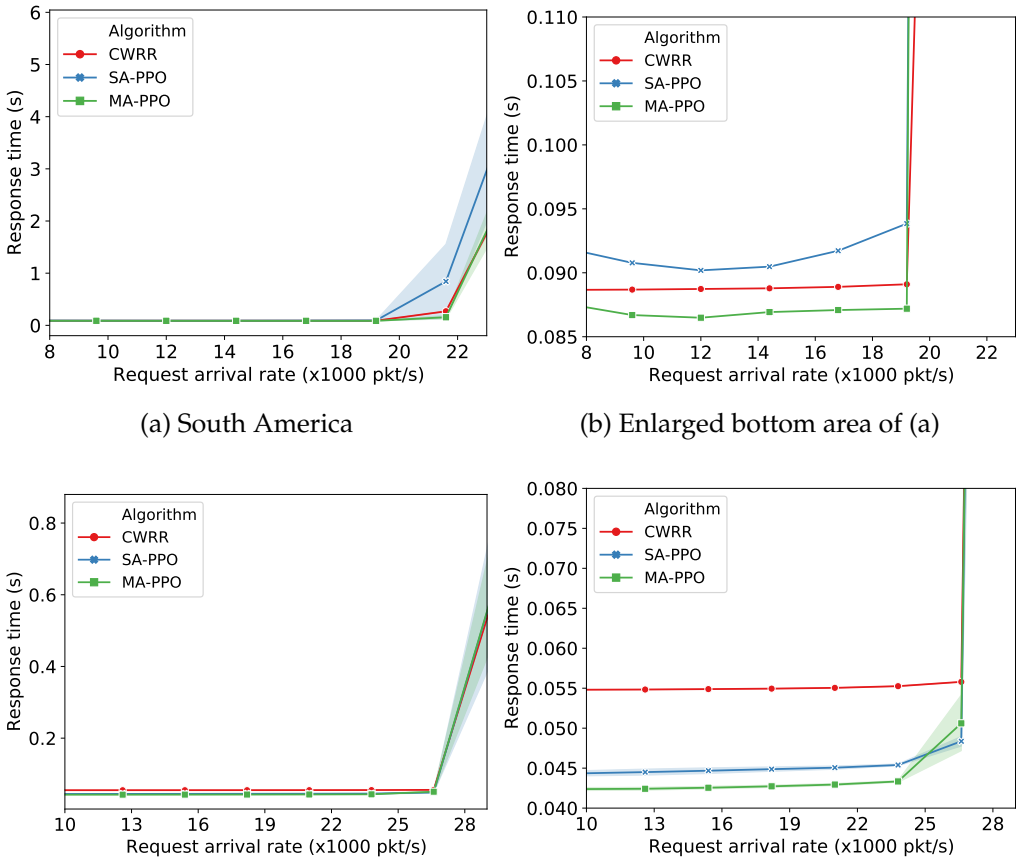
ure 5.15(a) and Figure 5.15(b) respectively.

We can observe a high variance in response time at the later TIs from Figure 5.15(a) which implies that the learning fails to converge. This observation confirms the non-stationary environment issue when SA-DRL algorithms are used in an MA environment as we discussed in Subsection 5.3.2. Correspondingly, we can also see from Figure 5.15(b) that as the number of iterations increases, SA-PPO-MA can keep the response time at a low level when the training arrival rate is low (i.e., the left dotted line LWL). However, it fails to avoid overloading controllers at the high training arrival rate (i.e., the right dotted line HWL). This is mainly because SA-PPO-MA ignores the impact of other agents during the training. As the request arrival rate increases, the importance of agent cooperation becomes significant and the deficiency of SA-PPO-MA becomes obvious.

We also compare the performance of the trained policies via MA-PPO and SA-PPO-MA respectively on two network topologies. Figure 5.16 confirms that polices trained by MA-PPO can effectively cope with increasing requests through better agent cooperation.

Performance comparison with existing policies: We compare MA-PPO with several policies as summarized in Table 5.3: (1) a widely used man-made policy (CWRR), a GD-based policy, and the centralized single-agent policy SA-PPO. The results are shown in Figure 5.17.

In particular, we can see that in both topologies (Figure 5.17(b) and Figure 5.17(d)), the response time of CWRR remains stable because the number of requests dispatched to each controller is proportional to its capacity, which effectively prevents overloading any controller at an early stage. However, solely sending requests based on the controller capacity may not achieve the optimal network performance. Especially, when the workload of the control plane is low, dispatching more requests to a closer controller without overloading it is a better option. In DRL, the relationship is learned during the interaction between the agents and the environment. Therefore, we can see from both Figure 5.17(b) and Figure 5.17(d)



(a) South America

(b) Enlarged bottom area of (a)

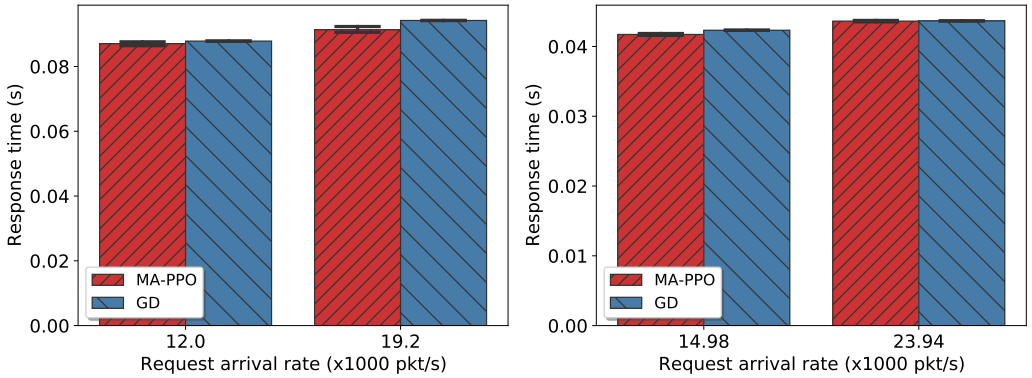
(c) Asia

(d) Enlarged bottom area of (c)

Figure 5.17: Performance comparison among CWRR, SA-PPO, and MA-PPO.

that MA-PPO achieves a lower response time compared to CWRR. Apart from that, we also notice that MA-PPO achieves a lower response time than SA-PPO, which confirms that using a centralized agent can introduce additional propagation latencies.

MA-PPO is also compared with a model-based optimization approach denoted by GD. As shown in Figure 5.18, MA-PPO can achieve slightly lower response time. This is mainly because GD optimizes the response



(a) South America (b) Asia

Figure 5.18: Performance comparison between MA-PPO and GD.

time for given network information. However, in general, network information such as request arrival rates can only be estimated. The inaccurate network information hinders GD achieving its optimal performance.

Even though both GD and MA-PPO achieve similar performance, MA-PPO has the advantage of low computation and communication overheads. During our simulation, we notice that the execution time of GD is 10 times longer than running the MA-PPO policy. The reason is that to obtain the request dispatching probabilities for the entire data plane (i.e., all switch centers), GD needs to iteratively perform gradient calculation. This can be computational intensive especially for a network with a large number of switch centers. In comparison, each agent in MA-PPO runs its policy individually (a forward pass from NN input layer to output layer) to calculate its request dispatching probability used by the switch center where the agent is placed on. Apart from that, for communication overheads, GD requires the information collected from the entire network while MA-PPO uses only local observation. Therefore, MA-PPO is more suitable for large-scale networks.

Summary: The simulation results of SA-PPO-MA confirmed that the

direct application of SA-DRL algorithms in an MA setting cannot train the policy effectively due to the non-stationarity issue. Also, comparing with existing policies (including man-made, optimization-based, and DRL-trained policies), the MA-PPO trained policy can effectively reduce the average request response time using local network information without introducing additional computation and communication overhead.

5.4 Chapter Summary

The overall goal of this chapter is to tackle the RM problem, particularly the CSP. This goal has been successfully achieved by developing two algorithms called SA-PPO and MA-PPO respectively, together with a new adaptive policy representation.

First, the RDPD problem has been formulated as an MDP under both SA-DRL and MA-DRL frameworks. Specifically, our SA-MDP formulated in Subsection 5.2.1 assumes a centralized agent in charge of request dispatching for all switches with the support of global network information (i.e., a fully observable environment). This assumption can be easily supported in a small-scale network with small geographic coverage (e.g., a local data center hosted by universities). On the other hand, for large-scale networks (e.g., WAN), the long propagation latency becomes inevitable which makes obtaining timely global information impractical. To address this issue, MA-DRL has been adopted in Subsection 5.3.1. In particular, the RDPD problem has been formulated as an MA-MDP where multiple agents have been involved to assist in making request dispatching decisions based on their local network information.

Second, a new adaptive DNN-based policy representation has been proposed in Subsection 5.2.2 to allow any switch to distribute its requests among all available controllers that may vary with time. In particular, a DNN serves as a priority function to calculate the priority of each controller given its state information. Based on the controller priorities, the

dispatching probabilities are calculated for any request arriving at a switch within a specific time period to be distributed to every eligible controller. Meanwhile, a controller filtering mechanism has been embedded into the probability calculation to prevent controller overloading as well as undesirable use of remote controllers.

Third, in line with the new policy representation, a new mathematical technique has been developed to calculate the gradient of the new policy. Armed with the new technique, DRL algorithms called SA-PPO and MA-PPO have been proposed in Subsections 5.2.4 and 5.3.2 to perform policy training under SA-DRL and MA-DRL settings respectively. In particular, to address the non-stationarity issue in an MA setting, MA-PPO follows the general principle of centralized training and decentralized execution.

Extensive simulations have been conducted in Subsections 5.2.5 and 5.3.3. Particularly, the effectiveness of the new policy representation has been evaluated under the SA-DRL setting in Subsection 5.2.5. This is mainly because the performance of MA-DRL not only relies on the policy design but also other factors, e.g., non-stationarity handling. Thus, it is easier to demonstrate the effectiveness of the new policy design using SA-DRL. The results have shown that compared to the policy designed in the traditional style, our new policy design can reduce the NN complexity without performance degradation. Apart from that, the new policy design can maintain good network performance with a changing number of controllers and different request arrival rates, which has demonstrated its good adaptiveness. Moreover, our simulation results on MA-DRL in Subsection 5.3.3 have shown that the policy trained via MA-PPO significantly outperformed man-made policy (weighted round robin) and the policy learned via SA-PPO in terms of response time. In comparison to the model-based policy (GD), MA-PPO policy can significantly reduce the computation and communication overheads.

With the help of our new policy design, the policies trained by MA-PPO can effectively adapt to networks with a changing number of con-

trollers. However, it may not apply directly to a network with a changing number of switches (or switch centers). This is mainly because in MA-PPO, each agent trains its policy with respect to a particular switch. Therefore, the policy trained on one switch may not perform consistently well when it is used by a different switch. To tackle this issue, additional policy training is required for any newly added switch. The policy trained by other switches can serve as the initialized policy to accelerate the training process. Since the network we considered in this chapter is a regional communication backbone, network topology changes are unlikely to happen. Therefore, policies trained by MA-PPO can perform consistently well. Nevertheless, multi-agent training of a shared adaptive policy will be investigated in the future where the policy is trained using inter-agent experience and shared by all switches.

Chapter 6

Conclusions and Future Work

This chapter summaries the thesis, outlines our major contributions, and points out the directions for our future work.

The goal of this thesis is to effectively manage the controller resources in distributed SDN controller architectures. This goal has been successfully achieved from both the architectural and algorithm design perspectives. From the architectural aspect, a new distributed controller architecture has been proposed to enable flexible CP and CS. Based on the new architecture, new EC-based algorithms have been designed to scalably solve the CPP. Apart from that, a new GD-based approach and new DRL-based approaches have been proposed to design effective policies to address the CSP. Both the proposed architecture and algorithms have been evaluated on a range of network topologies and compared with widely-used and state-of-the-art methods. The results showed that both the architecture and algorithms proposed in this thesis outperformed existing approaches and effectively improved the resource utilization of the control plane and network performance in terms of both response time and throughput.

The rest of this chapter is organized as follows. Section 6.1 provides the conclusions of each individual objective proposed in this thesis. Main findings are summarized in Section 6.2. Potential research directions for future work are pointed out in Section 6.3.

6.1 Achieved Objectives

The overall goal of this thesis is to effectively manage the controller resources in distributed SDN architectures. This goal has been successfully achieved by developing a comprehesive solution to address the challenges from both the architectural and algorithm design aspects. Specifically, the following research objectives have been fulfilled:

- This thesis has achieved the first research objective by proposing a new BindingLess Architecture for distributed SDN Controllers, namely BLAC (Chapter 3). BLAC aims to enable flexible and transparent CP and CS by removing the switch-controller binding constraint. Instead of directly connecting to controllers, switches send their requests to a newly introduced scheduling layer which is in charge of distributing requests among controllers so as to optimize network performance. A prototype of BLAC was implemented based on ONOS, a widely-used open source SDN controller architecture. The experimental results showed that BLAC outperformed existing SDN architectures in terms of average response time and throughput. Furthermore, BLAC eliminates the necessity of switch-controller re-association whenever the CP is changed. In addition, the newly introduced scheduling layer enables fine-grained workload distribution among controllers where a CS decision can be made on each request.

- The second research objective has been achieved by developing a new algorithm named CGA-CC (Chapter 4) to simultaneously address both the CPP and the CSP. Specifically, it should be noted that the network performance is not only related to CP but also CS. However, existing studies tend to oversimplify or completely ignore the CSP when addressing the CPP. This thesis introduced a new problem formulation to jointly tackle both the CPP and the CSP within a coherent framework. Based on the new formulation, CGA-CC

(Chapter 4) was developed to simultaneously address both the CPP and the CSP. Instead of directly applying GA in a large search space, network partitioning was adopted in CGA-CC to split the network into non-overlapping sub-networks so that GA can effectively identify CP within each sub-network. Moreover, to alleviate the impact of unexpected bursting requests, a new greedy mechanism was developed to strategically offload indigestible requests to adjacent sub-networks. Given CP, a GD-based scheduling algorithm was developed to optimize the probabilities of request distribution across multiple controllers. Extensive simulations have made it evident that CGA-CC significantly outperformed the widely-used and state-of-the-art algorithms (e.g., K-center [118] and MSPA [279]), and can effectively handle unexpected bursting traffic.

- This thesis developed a new MA-DRL approach (Chapter 5) to automatically design policies and successfully achieved the third objective. Compared to the GD-based optimization algorithm developed in Chapter 4, our MA-DRL approach does not require an explicit mathematical model of the underlying network environment. Apart from that, whenever the network condition changes (e.g., the request arrival rate, the number of controllers), GD needs to be rerun to calculate the request dispatching probabilities for all switches which can introduce additional computation overhead. By contrast, once the policy is trained by MA-DRL, it can perform consistently well without any retraining/modification under a changing network condition. Specifically, in order to accommodate the changes of numbers of controllers, a new policy representation was designed. In line with the new policy, a new training algorithm named MA-PPO was proposed armed with a new gradient calculation technique, enabling multiple agents to simultaneously learn their adaptive local policies. The simulation results showed that the policy trained using our approach can achieve significantly better performance compared with

man-made policies (e.g., weighted round robin scheduling) as well as policies learned via other RL algorithms.

6.2 Main Conclusions

In general, this thesis found that the RM problem in distributed SDN controller architectures can be effectively addressed from two aspects: the architectural design aspect and the algorithm design aspect. The newly proposed architecture and algorithms in this thesis have successfully outperformed the prior widely-used and state-of-the-art algorithms in terms of resource utilization and network performance.

The main conclusions for the three research objectives (Section 1.3) drawn from the three contribution chapters (Chapter 3 through Chapter 5) are summarized and discussed in this section.

6.2.1 Distributed SDN Controller Architectures

This thesis proposed a new distributed controller architecture for controller resource management (Chapter 3). To avoid the switch-controller binding constraint in existing architectures, the proposed architecture features a bindingless switch-controller association with a newly introduced scheduling layer, enabling requests from a switch to be processed by different controllers. The performance of our proposed architecture has been compared with the conventional static-binding and state-of-the-art dynamic-binding controller architectures, i.e., ONOS [45] and ElastiCon [80]. The results show that our architecture significantly outperformed the competitive architectures.

6.2.1.1 Bindingless Switch-controller Association

Compared to existing switch-controller binding based architectures, we found that without the switch-controller binding constraint, the network

performance can be significantly improved in terms of the average request response time and the overall throughput even though the requests are dispatched to controllers chosen independently and uniformly at random (Subsection 3.5.2).

Specifically, switch-controller binding restricts the requests from a switch to be processed by a predefined controller regardless of the controller's workload. Since each switch comes with different workload and its workload can be time-variant, the switch-controller binding renders the bound controller susceptible to either being overloaded or underloaded. Therefore, the existing controller resources cannot be effectively exploited. Motivated by this understanding, we proposed a bindingless design which provides the flexibility of request dispatching. In our design, a switch can flexibly select different controllers to process different requests based on the controllers' current status (e.g., workload and propagation latency). Therefore, compared to the switch-controller binding architectures, our bindingless design can more efficiently utilize the controller resources to improve the network performance.

6.2.2 Controller Placement

This thesis proposed a new algorithm called CGA-CC which simultaneously addressed both the CPP and the CSP (Chapter 4). Compared to the majority of existing methods, CGA-CC has three unique strengths including network partitioning, joint consideration of the CSP and the CPP, and a greedy load re-distribution mechanism which are discussed below. Simulations on multiple topologies and controller settings show that compared to the competitive methods (e.g., K-center [118]), CGA-CC can lower the response time by up to 70% and maintain high control plane utilization.

6.2.2.1 Network Partitioning

We observed (Subsection 4.5.4) that with the help of network partitioning, CGA-CC can effectively reduce the average response time by up to 60% compared to GA without partitioning and 75% compared to K-center. The performance improvement demonstrated that network partitioning can reduce the response time by effectively preventing long-distance communication. Furthermore, by partitioning the whole network into several sub-networks, the search space of GA is reduced, resulting in faster identification of high-quality solutions compared to the exploration of the whole search space in standard GA.

6.2.2.2 The Impact of the CSP on the CPP

This thesis verified the importance and influence of the CSP on the CPP. It has been observed (Subsections 4.5.3 and 4.5.4) that selecting controllers simply based on propagation latency without considering the CSP have performed poorly (e.g., K-center). This selection criteria is insufficient mainly because it overlooks the influence of the CSP on network performance. To address this issue, the proposed CGA-CC simultaneously addresses both the CPP and the CSP within a coherent framework by explicitly measuring the impact of the CSP on the response time. The simulation results in Subsections 4.5.3 and 4.5.4 have shown the effectiveness of quantifying the impact of the CSP for improving network performance.

6.2.2.3 Handling Unexpected Traffic Bursts

In general, enough controller resources should be provided for each sub-network using CP algorithms. However, there are special cases when a sub-network encounters unexpected traffic surge. Existing algorithms (e.g., MSPA [279]) treat each clustered sub-network as being completely independent. Therefore, for these algorithms, a dramatical growth in the average response time can be spotted in Subsections 4.5.5 and 4.5.6 before

deploying more controllers. On the contrary, a greedy mechanism was developed in this thesis to strategically offload indigestible requests to adjacent sub-networks. The simulation results reported in Subsections 4.5.5 and 4.5.6 have shown that our greedy offloading mechanism can effectively cope with the unexpected demand variations and keep the average response time low. Therefore, sub-network collaboration by sharing their controller resources has been found to be vital for handling unexpected traffic bursts.

6.2.3 Controller Scheduling

The bindingless switch-controller association provides the opportunity of distributing requests among controllers. However, how to distributing the requests depends on the scheduling approach. This thesis investigates and compares the performance of different scheduling approaches (Chapter 3 through Chapter 5) including heuristics (e.g., weighted round robin), mathematical optimization methods (e.g., GD-based scheduling approach), and learning-based approaches (e.g., policies designed by DRL). The main findings can be summarized as follows:

6.2.3.1 Input Information for the Policy

For policy design, the thesis showed the impact of input network information on the scheduling performance. In Chapter 3, with limited controller information (i.e., CPU utilization), weighted round-robin and improved random policies achieved significantly better performance compared to the purely random policy. However, scheduling requests solely based on CPU utilization information may not be sufficient, especially in a large-scale network where propagation latency contributes significantly to response time. This understanding was verified by our simulation studies in Subsections 4.5.2 and 5.2.5.3. Motivated by this, GD in Chapter 4 and the RL policies in Chapter 5 which take multiple network information

(including controller capacities, request arrival rate and switch-controller propagation latency) achieve lower average response time compared to weighted round-robin which distributes requests solely relying on the controller capacities.

6.2.3.2 Centralized vs. Distributed Scheduling

In general, CS can be performed in a centralized manner where the CS decision is made for the entire data plane (i.e., all switches) using the information collected from the entire networks. Depending on different implementations, the CS decisions can be enforced by a centralized scheduler (e.g., MSPA in Subsection 2.4.2.3 and SA-DRL in Section 5.2) or multiple schedulers (e.g., GD-based scheduling in Subsection 4.3.2). Alternatively, the CS decisions can also be made in a distributed way (i.e., MA-DRL in Section 5.3). In particular, each decision making instance/learning agent calculates the scheduling strategy for one or multiple switches using its limited local network information. The scheduling strategies obtained from all instances jointly form the CS decision for the whole data plane.

Clearly, centralized scheduling is more straightforward since it avoids the complexity caused by multiple instances (e.g., handling instance interaction). However, it also tends to present scalability problems due to the limited capacity of the centralized decision making instance. Apart from that, the use of global information in centralized scheduling may not be practical since obtaining timely global information over the entire network can introduce substantial communication overhead.

On the other hand, distributed scheduling can alleviate the scalability problems and does not require expensive global information. However, since the scheduling strategy is independently made by each decision making instance, instance cooperation or interference needs to be considered. For example, without considering cooperation, multiple instances may simultaneously schedule a large number of requests to a same controller, which can easily overload the controller and cause high request

response time. The cooperation is supported in our MA-DRL approach (Section 5.3) in two ways. First, each agent shares the same goal of minimizing the response time and increasing the control plane throughput. Second, we follow the framework of centralized training and decentralized execution where a centralized value function that explicitly considers the inter-agent interactions is trained. The trained centralized value function is shared among all agents and is used to improve their policy.

6.2.3.3 Adaptive Policy Representation

Instead of targeting at specific networks with fixed controllers, this thesis proposed a new policy representation that allows the trained policy to function and consistently perform well over different network settings. In terms of performance effectiveness, our approach can achieve comparable performance compared to existing policy representations that directly adopt an NN with fixed number of output nodes, as demonstrated in the results from Chapter 5. Apart from that, our method also provides an alternative way for DRL action generation. In terms of adaptiveness, it has been shown in Chapter 5 that the trained policy can perform consistently well without any modification or retraining when the number of controllers changes. This is mainly because the policy is not trained for a specific controller. Instead, the training data aggregating from all controllers helps the policy to generalize well.

6.2.3.4 DRL for Policy Design

This thesis proposed DRL approaches (Chapter 5) to automatically learn a policy for CS. Unlike the GD-based scheduling approach proposed in Chapter 4, no explicit model of the complex network system is required in our DRL approaches. Instead, the relationship between the input data and the request dispatching decisions are learned during the interaction between the agent(s) and the environment. Therefore, our DRL approaches

lower the requirement of domain expertise. Apart from that, whenever the value of the input variables changes (e.g., request arrival rates and the number of controllers), model-based algorithms (e.g., our GD-based scheduling approach) needs to be rerun which introduces additional computation costs. On the other hand, our DRL approach can remain to perform consistently well without being retrained. The results in Chapter 5 have also demonstrated the capability, effectiveness and adaptiveness of our DRL approaches on addressing the CSP.

6.3 Future Work

Several potential research directions for future work motivated by our studies are highlighted in this section, including real-world system integration, reliable controller placement, and online MA-DRL training.

6.3.1 System Integration and Deployment

This thesis developed an SDN controller resource management system that consists of three key components: a bindingless controller architecture, controller placement algorithms (e.g., GA, CGA, CGA-CC) and controller scheduling algorithms (e.g., GD and policies trained using DRL). The whole system can be naturally integrated and deployed to existing SDN networks (e.g., data centers and WANs run by the same organization). Due to hardware limitations, we followed a common practice among a majority of related research works [272, 279, 280] to evaluate our system in a simulated network environment. Apart from that, the controller capacity can be affected by many factors (e.g., the hardware device that it is running on and the network bandwidth). In this thesis, the controller capacity was set based on the measurement result reported in [17]. To ensure that our simulation results are accurate and reliable, a network simulator has been built to closely simulate the network behaviors. Real-

world network topologies and traffic traces have been utilized in our simulation runs. Furthermore, we also compared the results generated by our simulator with existing works using the same algorithm to ensure the correctness of our simulator. We are hence confident that our evaluation should closely reflect the real-world performance of the system. Nevertheless, to encourage the widespread use of our system, it is an important direction for our future research to further examine the practical usefulness and the commercial value of the our resource management system in large-scale production networks. To achieve this goal, comprehensive network system knowledge, substantial software implementation and reliable hardware support are critical.

6.3.2 Reliable Controller Placement

As demonstrated in this thesis (Chapter 4), the proposed CGA-CC algorithm on tackling the CPP is very promising. However, CGA-CC mainly focuses on maximizing the control plane utilization and simultaneously maintaining low network response time. Network failures (e.g., link failures and node failures) are not considered. However, a link failure can result in increasing propagation latency for related nodes. Moreover, the overall processing capacity of the control plane is reduced when the network node that a controller is deployed at encounters failures, which can significantly increase the request processing time and potentially overload the entire control plane. Therefore, it is important to consider network failures when tackling the CPP. In particular, a prediction model can be built for predicting future network failures which can be incorporated into our formulation. This can proactively avoid future failures or minimize the failure impact on network performance. Apart from that, how to improve our existing algorithm to reactively handle the failure when it happens also needs to be investigated.

6.3.3 Online vs. Offline Training

This thesis investigated the use of MA-DRL in addressing the CSP using the trained policies (Chapter 5). In general, existing DRL training methods can be widely divided into two categories: (1) Offline training: the model or policy is trained in advance, which remains fixed during real-world evaluation/execution; or, (2) Online training: the training is performed online where the model or policy is adjusted based on the interactions with the real-world environment. In this thesis, offline training is adopted due to the following two reasons: (1) During the training process, a policy can perform badly (e.g., overloading some controllers) especially at the beginning. However, a production network should always guarantee reasonable good performance. (2) To avoid convergence to a suboptimal policy, agents are encouraged to sufficiently explore the environment/network which can introduce high exploration cost during online training. Therefore, to ensure the performance of the production network and reduce the training cost, offline training is preferable where the policy is trained in a network simulator with real-world network topologies and traffic traces.

However, due to the fact that it is impractical to train a policy to accommodate all situations the agent may see in the real world, any unexpected perturbations or unseen situations can potentially deteriorate the performance of the trained policy [209]. To address this issue, adapting the pretrained policy to the changes in the real world using online training can be a potential solution. Although online training sounds promising, its application to real-world networks entails several challenges. For example, how to balance the exploration and exploitation given the high online exploration risk. Apart from that, how to further improve sample efficiency and enable quick adaptation given limited data samples is still an open research question.

6.3.4 Proactive vs. Reactive Mode

This thesis mainly focused on the use of reactive flow setup mode. Note that in reactive mode, controllers act after receiving the requests from switches, which enables controllers to flexibly handle network events in a timely manner [91]. In comparison, controllers in proactive mode populate flow rules in switches in advance, which potentially avoids the switch-controller communication time and reduces controller workload incurred by processing *Packet-in* requests. To take the advantages of both reactive and the proactive modes, it is important to investigate the use of the hybrid mode where certain flow rules are proactively installed in the controllers while maintaining the controllers' flexibility of reactively handling incoming traffic.

Although hybrid mode sounds promising, its implementation entails several challenges. For example, it is important to decide the granularity of the proactively installed flow rules. On one hand, wildcard rules can be installed in switches which can effectively avoid invoking controllers on new flow arrivals. However, the use of wildcard rules can easily lead to imbalanced network link utilization since all matching flows are routed over the same path. Apart from that, the statistics of all flows matching the same rule are aggregated into a single set of counters, hindering controllers' visibility of the flows in the network [75]. On the other hand, if fine-grained rules are preinstalled, the number of rules will significantly increase compared to wildcard rules, which requires large TCAM space consumption. Moreover, the large number of flow rules also results in long flow rule look-up and insertion time [60] as well as high TCAM power consumption [27]. Thus, identifying the suitable granularity of the preinstalled flow rules needs to be investigated.

Bibliography

[1] Cluster Coordination. https://goo.gl/Qaib5M.

[2] DE-CIX, Traffic Statistic. https://www.de-cix.net/en/locations/germany/frankfurt/statistics. Accessed: 12-Aug-2019. [Online].

[3] Digital 2019: Global Internet Use Accelerates. https://wearesocial.com/blog/2019/01/digital-2019-global-internet-use-accelerates.

[4] EZchip Announces OpenFlow 1.1 Implementations on its NP-4 100-Gigabit Network Processor. https://urlzs.com/EGiZ.

[5] Floodlight. http://floodlight.openflowhub.org.

[6] Google Tensor Processing Unit. https://goo.gl/L4AJe3.

[7] Internet usage worldwide - Statistics and Facts. https://www.statista.com/topics/1145/internet-usage-worldwide/.

[8] NoviSwitch: High Performance OpenFlow Switch. https://noviflow.com/wp-content/uploads/NoviSwitch-1132-Datasheet-V2_1.pdf.

[9] NTT Com's SD-WAN. https://www.nttict.com/sd-wan/.

[10] ONOS. https://github.com/opennetworkinglab/onos/tree/master/apps.

[11] Open vSwitch Database Management Protocol (OVSDB). https://tools.ietf.org/html/rfc7047.

[12] OpenFlow configuration protocols: Understanding OF-Config and OVSDB. https://goo.gl/DJnEaY.

[13] OpenFlow Management and Configuration Protocol. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow-config/of-config-1.2.pdf.

[14] OpenFlow Switch Specification Version 1.5.1. https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf.

[15] Ryu. https://osrg.github.io/ryu/.

[16] SDN Versus Traditional Networking Explained. https://www.cisco.com/c/dam/en_us/solutions/industries/docs/gov/software_defined_networking.pdf.

[17] Single-node ONOS Cbench. https://wiki.onosproject.org/display/ONOS/1.6%3A+Experiment+G+-+Single-node+ONOS+Cbench.

[18] Software-Defined Networking and Network Programmability: Use Cases for Defense and Intelligence Communities. https://www.ibm.com/services/network/sdn-versus-traditional-networking.

[19] Sprint network. https://www.sprint.net/performance/. Accessed: 15-Jan-2019. [Online].

[20] Sprint SLA Performance. https://www.sprint.net/sla_performance.php?network=sl. Accessed: 10-Nov-2019. [Online].

[21] TensorFlow. https://www.tensorflow.org/.

[22] Theano. http://deeplearning.net/software/theano/.

[23] World Internet Users Statistics and 2019 World Population Stats. https://www.internetworldstats.com/stats.htm.

[24] Dedicated Network Equipment. https://www.leaseweb.com/network-services/dedicated-network-equipment, 2020.

[25] ADLER, N. Competition in a deregulated air transportation market. *European Journal of Operational Research 129*, 2 (2001), 337–345.

[26] AHMADI, V., AND KHORRAMIZADEH, M. An adaptive heuristic for multi-objective controller placement in software-defined networks. *Computers & Electrical Engineering 66* (2018), 204–228.

[27] ALGHADHBAN, A., AND SHIHADA, B. Energy efficient sdn commodity switch based practical flow forwarding method. In *NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium* (2016), IEEE, pp. 784–788.

[28] ALPAYDIN, E. *Introduction to machine learning*. MIT press, 2020.

[29] APT, K. *Principles of constraint programming*. Cambridge university press, 2003.

[30] ARAB, A., AND ALFI, A. An adaptive gradient descent-based local search in memetic algorithm applied to optimal controller design. *Information Sciences 299* (2015), 117–142.

[31] ARORA, J. S. Introduction to optimum design.

[32] AUDET, C., HANSEN, P., JAUMARD, B., AND SAVARD, G. A branch and cut algorithm for nonconvex quadratically constrained quadratic programming. *Mathematical Programming 87*, 1 (2000), 131–152.

[33] BAIRD, L. Residual algorithms: Reinforcement learning with function approximation. In *Machine Learning Proceedings 1995*. Elsevier, 1995, pp. 30–37.

[34] BAKER, B. M., AND AYECHEW, M. A genetic algorithm for the vehicle routing problem. *Computers & Operations Research 30*, 5 (2003), 787–800.

[35] BANNOUR, F., SOUIHI, S., AND MELLOUK, A. Distributed sdn control: Survey, taxonomy, and challenges. *IEEE Communications Surveys & Tutorials 20*, 1 (2018), 333–354.

[36] BARBANCHO, J., LEÓN, C., MOLINA, F. J., AND BARBANCHO, A. A new qos routing algorithm based on self-organizing maps for wireless sensor networks. *Telecommunication Systems 36*, 1-3 (2007), 73–83.

[37] BARI, M. F., BOUTABA, R., ESTEVES, R., GRANVILLE, L. Z., PODLESNY, M., RABBANI, M. G., ZHANG, Q., AND ZHANI, M. F. Data center network virtualization: A survey. *IEEE Communications Surveys & Tutorials 15*, 2 (2013), 909–928.

[38] BARI, M. F., ROY, A. R., CHOWDHURY, S. R., ZHANG, Q., ET AL. Dynamic controller provisioning in software defined networks. In *Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013)* (2013), IEEE, pp. 18–25.

[39] BARRON, A. R. Approximation and estimation bounds for artificial neural networks. *Machine learning 14*, 1 (1994), 115–133.

[40] BARROS, R. C., DE CARVALHO, A. C., AND FREITAS, A. A. Evolutionary algorithms and hyper-heuristics. In *Automatic Design of Decision-Tree Induction Algorithms*. Springer, 2015, pp. 47–58.

[41] BELL, M. G., AND CASSIR, C. Risk-averse user equilibrium traffic assignment: an application of game theory. *Transportation Research Part B: Methodological 36*, 8 (2002), 671–681.

[42] BELLMAN, R. On the theory of dynamic programming. *Proceedings of the National Academy of Sciences of the United States of America 38*, 8 (1952), 716.

[43] BELLMAN, R. The theory of dynamic programming. Tech. rep., Rand corp santa monica ca, 1954.

[44] BENSON, T., AKELLA, A., AND MALTZ, D. A. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement* (2010), ACM, pp. 267–280.

[45] BERDE, P., GEROLA, M., HART, J., HIGUCHI, Y., ET AL. ONOS: towards an open, distributed SDN OS. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking* (2014), ACM, pp. 1–6.

[46] BEYER, H.-G., AND SCHWEFEL, H.-P. Evolution strategies–a comprehensive introduction. *Natural computing 1*, 1 (2002), 3–52.

[47] BIANCO, A., GIACCONE, P., MAHMOOD, A., ULLIO, M., AND VERCELLONE, V. Evaluating the sdn control traffic in large isp networks. In *2015 IEEE International Conference on Communications (ICC)* (2015), IEEE, pp. 5248–5253.

[48] BIANCO, A., GIACCONE, P., MASHAYEKHI, R., ULLIO, M., AND VERCELLONE, V. Scalability of onos reactive forwarding applications in isp networks. *Computer Communications 102* (2017), 130–138.

[49] BLUM, C., AND ROLI, A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM computing surveys (CSUR) 35*, 3 (2003), 268–308.

[50] BO, H., YOUKE, W., CHUAN'AN, W., AND YING, W. The controller placement problem for software-defined networks. In *Computer and Communications (ICCC), 2016 2nd IEEE International Conference on* (2016), IEEE, pp. 2435–2439.

[51] BOJOVIĆ, B., MESHKOVA, E., BALDO, N., RIIHIJÄRVI, J., AND PETROVA, M. Machine learning-based dynamic frequency and bandwidth allocation in self-organized lte dense small cell deployments. *EURASIP Journal on Wireless Communications and Networking 2016*, 1 (2016), 183.

[52] BUYYA, R. Economic-based distributed resource management and scheduling for grid computing. *arXiv preprint cs/0204048* (2002).

[53] CAI, Z., COX, A. L., AND NG, T. Maestro: A system for scalable openflow control. Tech. rep., 2010.

[54] CAO, X., MA, R., LIU, L., SHI, H., CHENG, Y., AND SUN, C. A machine learning-based algorithm for joint scheduling and power control in wireless networks. *IEEE Internet of Things Journal 5*, 6 (2018), 4308–4318.

[55] CELEBI, M. E., KINGRAVI, H. A., AND VELA, P. A. A comparative study of efficient initialization methods for the k-means clustering algorithm. *Expert systems with applications 40*, 1 (2013), 200–210.

[56] CELLO, M., XU, Y., WALID, A., WILFONG, G., CHAO, H. J., AND MARCHESE, M. Balcon: A distributed elastic sdn control via efficient switch migration. In *2017 IEEE International Conference on Cloud Engineering (IC2E)* (2017), IEEE, pp. 40–50.

[57] CHAPELLE, O., SCHOLKOPF, B., AND ZIEN, A. Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]. *IEEE Transactions on Neural Networks 20*, 3 (2009), 542–542.

[58] CHARILAS, D. E., AND PANAGOPOULOS, A. D. A survey on game theory applications in wireless networks. *Computer Networks 54*, 18 (2010), 3421–3430.

[59] CHEN, G., PENG, Y., AND ZHANG, M. An adaptive clipping approach for proximal policy optimization. *arXiv preprint arXiv:1804.06461* (2018).

[60] CHEN, H., AND BENSON, T. The case for making tight control plane latency guarantees in sdn switches. In *Proceedings of the Symposium on SDN Research* (2017), pp. 150–156.

[61] CHEN, H., CHENG, G., AND WANG, Z. A game-theoretic approach to elastic control in software-defined networking. *China Communications 13*, 5 (2016), 103–109.

[62] CHEN, K.-Y., JUNUTHULA, A. R., SIDDHRAU, I. K., XU, Y., AND CHAO, H. J. Sdnshield: Towards more comprehensive defense against ddos attacks on sdn control plane. In *2016 IEEE Conference on Communications and Network Security (CNS)* (2016), IEEE, pp. 28–36.

[63] CHEN, R. Y., SIDOR, S., ABBEEL, P., AND SCHULMAN, J. Ucb exploration via q-ensembles. *arXiv preprint arXiv:1706.01502* (2017).

[64] CHENG, G., CHEN, H., HU, H., AND LAN, J. Dynamic switch migration towards a scalable sdn control plane. *International Journal of Communication Systems* (2016).

[65] CHENG, G., CHEN, H., WANG, Z., AND CHEN, S. Dha: Distributed decisions on the switch migration toward a scalable sdn control

plane. In *2015 IFIP Networking Conference (IFIP Networking)* (2015), IEEE, pp. 1–9.

[66] CHENG, T. Y., WANG, M., AND JIA, X. Qos-guaranteed controller placement in sdn. In *2015 IEEE Global Communications Conference (GLOBECOM)* (2015), IEEE, pp. 1–6.

[67] CHINCHALI, S., HU, P., CHU, T., SHARMA, M., BANSAL, M., MISRA, R., PAVONE, M., AND SACHIN, K. Cellular network traffic scheduling with deep reinforcement learning. In *AAAI* (2018).

[68] CHOY, S., WONG, B., SIMON, G., AND ROSENBERG, C. The brewing storm in cloud gaming: A measurement study on cloud to end-user latency. In *Proceedings of the 11th annual workshop on network and systems support for games* (2012), IEEE Press, p. 2.

[69] COELLO, C. A. C. Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. *Computer methods in applied mechanics and engineering 191*, 11-12 (2002), 1245–1287.

[70] COHEN, N., SHARIR, O., AND SHASHUA, A. On the expressive power of deep learning: A tensor analysis. In *Conference on Learning Theory* (2016), pp. 698–728.

[71] CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L., AND STEIN, C. *Introduction to algorithms*. MIT press, 2009.

[72] COVER, T., AND HART, P. Nearest neighbor pattern classification. *IEEE transactions on information theory 13*, 1 (1967), 21–27.

[73] COWEN, L. J. Compact routing with minimum stretch. *Journal of Algorithms 38*, 1 (2001), 170–183.

[74] CUI, J., LU, Q., ZHONG, H., TIAN, M., AND LIU, L. A load-balancing mechanism for distributed sdn control plane using response time. *IEEE Transactions on Network and Service Management 15*, 4 (2018), 1197–1206.

[75] CURTIS, A. R., MOGUL, J. C., TOURRILHES, J., YALAGANDULA, P., SHARMA, P., AND BANERJEE, S. Devoflow: Scaling flow management for high-performance networks. *ACM SIGCOMM Computer Communication Review 41*, 4 (2011), 254–265.

[76] CYBENKO, G. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems 2*, 4 (1989), 303–314.

[77] DANG, H. T., SCIASCIA, D., CANINI, M., ET AL. Netpaxos: Consensus at network speed. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research* (2015), pp. 1–7.

[78] DEISENROTH, M. P., NEUMANN, G., PETERS, J., ET AL. A survey on policy search for robotics. *Foundations and Trends® in Robotics 2*, 1–2 (2013), 1–142.

[79] DIJKSTRA, E. W. A note on two problems in connexion with graphs. *Numerische mathematik 1*, 1 (1959), 269–271.

[80] DIXIT, A. A., HAO, F., MUKHERJEE, S., LAKSHMAN, T., AND KOMPELLA, R. Elasticon: an elastic distributed sdn controller. In *Proceedings of the Tenth ACM/IEEE Symposium on Architectures for Networking and Communications Systems* (2014), ACM, pp. 17–28.

[81] DOHAN, D., KARP, S., AND MATEJEK, B. K-median algorithms: theory in practice. Tech. rep., Working paper, Princeton, Computer Science, 2015.

[82] DORIGO, M., BIRATTARI, M., AND STUTZLE, T. Ant colony optimization. *IEEE computational intelligence magazine 1*, 4 (2006), 28–39.

[83] EIBEN, A. E., AND SMITH, J. From evolutionary computation to the evolution of things. *Nature 521*, 7553 (2015), 476.

[84] ERICKSON, D. The beacon openflow controller. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking* (2013), ACM, pp. 13–18.

[85] ERLANG, A. K. The theory of probabilities and telephone conversations. *Nyt. Tidsskr. Mat. Ser. B 20* (1909), 33–39.

[86] ESTER, M., KRIEGEL, H.-P., SANDER, J., XU, X., ET AL. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd* (1996), vol. 96, pp. 226–231.

[87] FARACI, G., AND SCHEMBRA, G. An analytical model for electricity-price-aware resource allocation in virtualized data centers. In *2015 IEEE International Conference on Communications (ICC)* (2015), IEEE, pp. 5839–5845.

[88] FATTAH, H., AND LEUNG, C. An overview of scheduling algorithms in wireless multimedia networks. *IEEE Wireless Communications 9*, 5 (2002), 76–83.

[89] FAWCETT, L., SCOTT-HAYWARD, S., BROADBENT, M., WRIGHT, A., AND RACE, N. Tennison: a distributed sdn framework for scalable network security. *IEEE Journal on Selected Areas in Communications 36*, 12 (2018), 2805–2818.

[90] FEAMSTER, N., REXFORD, J., AND ZEGURA, E. The road to sdn: an intellectual history of programmable networks. *ACM SIGCOMM Computer Communication Review 44*, 2 (2014), 87–98.

[91] FERNANDEZ, M. P. Comparing openflow controller paradigms scalability: Reactive and proactive. In *2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)* (2013), IEEE, pp. 1009–1016.

[92] FERNÁNDEZ-FERNÁNDEZ, A., CERVELLÓ-PASTOR, C., AND OCHOA-ADAY, L. Energy efficiency and network performance: A reality check in sdn-based 5g systems. *Energies 10*, 12 (2017), 2132.

[93] FOERSTER, K.-T., SCHMID, S., AND VISSICCHIO, S. Survey of consistent software-defined network updates. *IEEE Communications Surveys & Tutorials* (2018).

[94] FOUNDATION, O. N. Atomix. `https://atomix.io/`.

[95] FU, Q., RUTTER, B., LI, H., ZHANG, P., ET AL. Taming the wild: A scalable anycast-based cdn architecture (t-sac). *IEEE Journal on Selected Areas in Communications 36*, 12 (2018), 2757–2774.

[96] FU, Y., BI, J., GAO, K., CHEN, Z., WU, J., AND HAO, B. Orion: A hybrid hierarchical control plane of software-defined networking for large-scale networks. In *2014 IEEE 22nd International Conference on Network Protocols* (2014), IEEE, pp. 569–576.

[97] FUNAHASHI, K.-I. On the approximate realization of continuous mappings by neural networks. *Neural networks 2*, 3 (1989), 183–192.

[98] GAI, K., QIU, M., AND ZHAO, H. Cost-aware multimedia data allocation for heterogeneous memory using genetic algorithm in cloud computing. *IEEE transactions on cloud computing* (2016).

[99] GAO, J., LING, H., HU, W., AND XING, J. Transfer learning based visual tracking with gaussian processes regression. In *European conference on computer vision* (2014), Springer, pp. 188–203.

[100] GAO, X., KONG, L., LI, W., LIANG, W., CHEN, Y., AND CHEN, G. Traffic load balancing schemes for devolved controllers in mega data centers. *IEEE Transactions on Parallel and Distributed Systems 28*, 2 (2016), 572–585.

[101] GHANI, N., SHAMI, A., ASSI, C., AND RAJA, M. Intra-onu bandwidth scheduling in ethernet passive optical networks. *IEEE Communications Letters 8*, 11 (2004), 683–685.

[102] GHOSH, M., BEGUM, S., SARKAR, R., CHAKRABORTY, D., AND MAULIK, U. Recursive memetic algorithm for gene selection in microarray data. *Expert Systems with Applications 116* (2019), 172–185.

[103] GLOVER, F. Future paths for integer programming and links to ar tifi cial intelli g en ce. *Computers operations research 13*, 5 (1986), 533–549.

[104] GLOVER, F., AND LAGUNA, M. Tabu search. In *Handbook of combinatorial optimization*. Springer, 1998, pp. 2093–2229.

[105] GORTON, I. Hyperscalability–the changing face of software architecture. In *Software Architecture for Big Data and the Cloud*. Elsevier, 2017, pp. 13–31.

[106] GROSS, D. *Fundamentals of queueing theory*. John Wiley & Sons, 2008.

[107] GUDE, N., KOPONEN, T., PETTIT, J., PFAFF, B., CASADO, M., MCKEOWN, N., AND SHENKER, S. Nox: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review 38*, 3 (2008), 105–110.

[108] GUO, C., YUAN, L., XIANG, D., DANG, Y., HUANG, R., MALTZ, D., LIU, Z., WANG, V., PANG, B., CHEN, H., ET AL. Pingmesh: A large-scale system for data center network latency measurement and analysis. In *ACM SIGCOMM Computer Communication Review* (2015), vol. 45, ACM, pp. 139–152.

[109] GUO, M., AND BHATTACHARYA, P. Controller placement for improving resilience of software-defined networks. In *2013 Fourth International Conference on Networking and Distributed Computing* (2013), IEEE, pp. 23–27.

[110] HAMEED, K., ALI, A., NAQVI, M. H., JABBAR, M., JUNAID, M., AND HAIDER, A. Resource management in operating systems-a survey of scheduling algorithms. In *Int. Conf. on Innovative Computing (ICIC)* (2016).

[111] HARTMANN, S. A competitive genetic algorithm for resource-constrained project scheduling. *Naval Research Logistics (NRL) 45*, 7 (1998), 733–750.

[112] HASSAS YEGANEH, S., AND GANJALI, Y. Kandoo: a framework for efficient and scalable offloading of control applications. In *Proceedings of the first workshop on Hot topics in software defined networks* (2012), pp. 19–24.

[113] HASSELBRING, W., AND STEINACKER, G. Microservice architectures for scalability, agility and reliability in e-commerce. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)* (2017), IEEE, pp. 243–246.

[114] HASSELT, H. V. Double q-learning. In *Advances in neural information processing systems* (2010), pp. 2613–2621.

[115] HE, K., FISHER, A., WANG, L., GEMBER, A., AKELLA, A., AND RISTENPART, T. Next stop, the cloud: Understanding modern web service deployment in ec2 and azure. In *Proceedings of the 2013 conference on Internet measurement conference* (2013), ACM, pp. 177–190.

[116] HE, M., BASTA, A., BLENK, A., AND KELLERER, W. Modeling flow setup time for controller placement in sdn: Evaluation for dynamic flows. In *2017 IEEE International Conference on Communications (ICC)* (2017), IEEE, pp. 1–7.

[117] HEESS, N., WAYNE, G., SILVER, D., LILLICRAP, T., EREZ, T., AND TASSA, Y. Learning continuous control policies by stochastic

value gradients. In *Advances in Neural Information Processing Systems* (2015), pp. 2944–2952.

[118] HELLER, B., SHERWOOD, R., AND MCKEOWN, N. The controller placement problem. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networking* (2012), ACM, pp. 7–12.

[119] HIASSAT, A., DIABAT, A., AND RAHWAN, I. A genetic algorithm approach for location-inventory-routing problem with perishable products. *Journal of manufacturing systems 42* (2017), 93–103.

[120] HIGINO, W., CHAVES, A. A., AND DE MELO, V. V. Biased random-key genetic algorithm applied to the vehicle routing problem with private fleet and common carrier. In *IEEE CEC* (2018), pp. 1–8.

[121] HOCHBA, D. S. Approximation algorithms for np-hard problems. *ACM Sigact News 28*, 2 (1997), 40–52.

[122] HOCHBAUM, D. S., AND SHMOYS, D. B. A best possible heuristic for the k-center problem. *Mathematics of operations research 10*, 2 (1985), 180–184.

[123] HOCK, D., HARTMANN, M., GEBERT, S., JARSCHEL, M., ZINNER, T., AND TRAN-GIA, P. Pareto-optimal resilient controller placement in sdn-based core networks. In *Teletraffic Congress (ITC), 2013 25th International* (2013), IEEE, pp. 1–9.

[124] HOLLAND, J. H., ET AL. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[125] HOLLANDE, J. H. Adaptation in natural and artificial systems. *University of Michigan Press, Ann Arbor, MI, MI, USA 1* (1975), 8.

[126] HORNIK, K., STINCHCOMBE, M., WHITE, H., ET AL. Multilayer feedforward networks are universal approximators. *Neural networks 2*, 5 (1989), 359–366.

[127] HU, F., HAO, Q., AND BAO, K. A survey on software-defined network and openflow: From concept to implementation. *IEEE Communications Surveys & Tutorials 16*, 4 (2014), 2181–2206.

[128] HU, J., LIN, C., LI, X., AND HUANG, J. Scalability of control planes for software defined networks: Modeling and evaluation. In *2014 IEEE 22nd International Symposium of Quality of Service (IWQoS)* (2014), IEEE, pp. 147–152.

[129] HU, T., GUO, Z., YI, P., BAKER, T., AND LAN, J. Multi-controller based software-defined networking: A survey. *IEEE Access 6* (2018), 15980–15996.

[130] HU, T., LAN, J., ZHANG, J., AND ZHAO, W. Easm: Efficiency-aware switch migration for balancing controller loads in software-defined networking. *Peer-to-Peer networking and applications 12*, 2 (2019), 452–464.

[131] HU, Y., LUO, T., BEAULIEU, N. C., AND DENG, C. The energy-aware controller placement problem in software defined networks. *IEEE Communications Letters 21*, 4 (2016), 741–744.

[132] HU, Y., WANG, W., GONG, X., QUE, X., AND CHENG, S. Balanceflow: controller load balancing for openflow networks. In *2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems* (2012), vol. 2, IEEE, pp. 780–785.

[133] HU, Y., WANG, W., GONG, X., QUE, X., AND CHENG, S. On reliability-optimized controller placement for software-defined networks. *China Communications 11*, 2 (2014), 38–54.

[134] HU, Y., WENDONG, W., GONG, X., QUE, X., AND SHIDUAN, C. Reliability-aware controller placement for software-defined networks. In *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on* (2013), IEEE, pp. 672–675.

[135] HUA, Y., LI, R., ZHAO, Z., CHEN, X., AND ZHANG, H. Gan-powered deep distributional reinforcement learning for resource management in network slicing. *IEEE Journal on Selected Areas in Communications* (2019).

[136] HUANG, V., CHEN, G., AND FU, Q. Effective scheduling function design in sdn through deep reinforcement learning. In *2019 IEEE International Conference on Communications (ICC)* (2019), IEEE, pp. 1–7.

[137] HUANG, V., CHEN, G., FU, Q., AND WEN, E. Optimizing controller placement for software-defined networks. In *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)* (2019), IEEE, pp. 224–232.

[138] HUANG, V., FU, Q., CHEN, G., WEN, E., AND HART, J. BLAC: A Bindingless Architecture for Distributed SDN Controllers. In *2017 IEEE 42nd Conference on Local Computer Networks (LCN)* (2017), IEEE, pp. 146–154.

[139] ISHIGAKI, G., GOUR, R., YOUSEFPOUR, A., SHINOMIYA, N., AND JUE, J. P. Cluster leader election problem for distributed controller placement in sdn. In *GLOBECOM 2017-2017 IEEE Global Communications Conference* (2017), IEEE, pp. 1–6.

[140] JAIN, S., KUMAR, A., MANDAL, S., ONG, J., ET AL. B4: Experience with a globally-deployed software defined wan. *ACM SIGCOMM Computer Communication Review 43*, 4 (2013), 3–14.

[141] JALILI, A., AHMADI, V., KESHTGARI, M., AND KAZEMI, M. Controller placement in software-defined wan using multi objective genetic algorithm. In *2015 2nd International Conference on Knowledge-Based Engineering and Innovation (KBEI)* (2015), IEEE, pp. 656–662.

[142] JAY, N., ROTMAN, N., GODFREY, B., SCHAPIRA, M., AND TAMAR, A. A deep reinforcement learning perspective on internet congestion control. In *ICML* (2019), pp. 3050–3059.

[143] JIMENEZ, Y., CERVELLO-PASTOR, C., AND GARCIA, A. J. On the controller placement for designing a distributed sdn control layer. In *Networking Conference, 2014 IFIP* (2014), IEEE, pp. 1–9.

[144] JOACHIMS, T. Transductive inference for text classification using support vector machines. In *Icml* (1999), vol. 99, pp. 200–209.

[145] JOHNSON, S. C. Hierarchical clustering schemes. *Psychometrika 32*, 3 (1967), 241–254.

[146] JOO, C., AND SHROFF, N. B. Performance of random access scheduling schemes in multi-hop wireless networks. *IEEE/ACM Transactions on Networking 17*, 5 (2009), 1481–1493.

[147] KANUNGO, T., MOUNT, D. M., NETANYAHU, N. S., PIATKO, C. D., SILVERMAN, R., AND WU, A. Y. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE transactions on pattern analysis and machine intelligence 24*, 7 (2002), 881–892.

[148] KARAKUS, M., AND DURRESI, A. A survey: Control plane scalability issues and approaches in software-defined networking (sdn). *Computer Networks 112* (2017), 279–293.

[149] KENNEDY, J. Swarm intelligence in handbook of nature-inspired and innovative computing: Integrating classical models with emerging technologies.

[150] KENNEDY, J., AND EBERHART, R. Particle swarm optimization. In *Proceedings of ICNN'95-International Conference on Neural Networks* (1995), vol. 4, IEEE, pp. 1942–1948.

[151] KHORRAMIZADEH, M., AND AHMADI, V. Capacity and load-aware software-defined network controller placement in heterogeneous environments. *Computer Communications 129* (2018), 226–247.

[152] KILLI, B. P. R., AND RAO, S. V. Capacitated next controller placement in software defined networks. *IEEE Transactions on Network and Service Management 14*, 3 (2017), 514–527.

[153] KILLI, B. P. R., AND RAO, S. V. On placement of hypervisors and controllers in virtualized software defined network. *IEEE Transactions on Network and Service Management 15*, 2 (2018), 840–853.

[154] KILLI, B. P. R., REDDY, E. A., AND RAO, S. V. Cooperative game theory based network partitioning for controller placement in sdn. In *2018 10th International Conference on Communication Systems & Networks (COMSNETS)* (2018), IEEE, pp. 105–112.

[155] KIM, H., BENSON, T., AKELLA, A., AND FEAMSTER, N. The evolution of network configuration: a tale of two campuses. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference* (2011), pp. 499–514.

[156] KIM, H., REICH, J., GUPTA, A., SHAHBAZ, M., FEAMSTER, N., AND CLARK, R. Kinetic: Verifiable dynamic network control. In *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)* (2015), pp. 59–72.

[157] KINGMA, D. P., AND BA, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[158] KOPONEN, T., CASADO, M., GUDE, N., STRIBLING, J., POUTIEVSKI, L., ET AL. Onix: A distributed control platform for large-scale production networks. In *OSDI* (2010), vol. 10, pp. 1–6.

[159] KOTHARI, P. Lecture note on advanced algorithm design: (lecture 17) games, min-max and equilibria, Fall 2016.

[160] KOTSIANTIS, S. B., ZAHARAKIS, I., AND PINTELAS, P. Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering 160* (2007), 3–24.

[161] KREUTZ, D., RAMOS, F. M., VERISSIMO, P. E., ROTHENBERG, C. E., AZODOLMOLKY, S., AND UHLIG, S. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE 103*, 1 (2015), 14–76.

[162] KSENTINI, A., BAGAA, M., TALEB, T., AND BALASINGHAM, I. On using bargaining game for optimal placement of sdn controllers. In *Communications (ICC), 2016 IEEE International Conference on* (2016), IEEE, pp. 1–6.

[163] KUIPERS, F., WANG, H., AND VAN MIEGHEM, P. The stability of paths in a dynamic network. In *Proceedings of the 2005 ACM conference on Emerging network experiment and technology* (2005), pp. 105–114.

[164] KUMARI, A., AND SAIRAM, A. S. A survey of controller placement problem in software defined networks. *arXiv preprint arXiv:1905.04649* (2019).

[165] LANGE, S., GEBERT, S., ZINNER, T., TRAN-GIA, P., HOCK, D., JARSCHEL, M., AND HOFFMANN, M. Heuristic approaches to the controller placement problem in large scale sdn networks. *IEEE Transactions on Network and Service Management 12*, 1 (2015), 4–17.

[166] LANTZ, B., HANDIGOL, N., HELLER, B., AND JEYAKUMAR, V. Introduction to Mininet. `https://github.com/mininet/mininet/wiki/Introduction-to-Mininet`, 2018.

[167] LANTZ, B., HELLER, B., AND MCKEOWN, N. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks* (2010), ACM, p. 19.

[168] LEE, D.-H. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *Workshop on challenges in representation learning, ICML* (2013), vol. 3, p. 2.

[169] LENG, M., AND PARLAR, M. Game theoretic applications in supply chain management: a review. *INFOR: Information Systems and Operational Research 43*, 3 (2005), 187–220.

[170] LEVIN, D., WUNDSAM, A., HELLER, B., HANDIGOL, N., AND FELDMANN, A. Logically centralized?: state distribution trade-offs in software defined networks. In *Proceedings of the first workshop on Hot topics in software defined networks* (2012), ACM, pp. 1–6.

[171] LEVINE, S., FINN, C., DARRELL, T., AND ABBEEL, P. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research 17*, 1 (2016), 1334–1373.

[172] LI, A., YANG, X., KANDULA, S., AND ZHANG, M. Cloudcmp: comparing public cloud providers. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement* (2010), ACM, pp. 1–14.

[173] LI, H., LI, P., GUO, S., AND NAYAK, A. Byzantine-resilient secure software-defined networks with multiple controllers in cloud. *IEEE Transactions on Cloud Computing 2*, 4 (2014), 436–447.

[174] LI, R., ZHAO, Z., SUN, Q., CHIH-LIN, I., YANG, C., CHEN, X., ZHAO, M., AND ZHANG, H. Deep reinforcement learning for resource management in network slicing. *IEEE Access 6* (2018), 74429–74441.

[175] LI, T., XU, Z., TANG, J., AND WANG, Y. Model-free control for distributed stream data processing using deep reinforcement learning. *Proceedings of the VLDB Endowment 11*, 6 (2018), 705–718.

[176] LI, Y., AND CHEN, M. Software-defined network function virtualization: A survey. *IEEE Access 3* (2015), 2542–2553.

[177] LIANG, C., GUO, Y., AND LIU, Y. Is random scheduling sufficient in p2p video streaming? In *Distributed Computing Systems, 2008. ICDCS'08. The 28th International Conference on* (2008), IEEE, pp. 53–60.

[178] LIANG, C., KAWASHIMA, R., AND MATSUO, H. Scalable and crash-tolerant load balancing based on switch migration for multiple open flow controllers. In *2014 Second International Symposium on Computing and Networking* (2014), IEEE, pp. 171–177.

[179] LIAO, J., SUN, H., WANG, J., QI, Q., LI, K., AND LI, T. Density cluster based approach for controller placement problem in large-scale software defined networkings. *Computer Networks 112* (2017), 24–35.

[180] LIAO, L., AND LEUNG, V. C. Genetic algorithms with particle swarm optimization based mutation for distributed controller placement in sdns. In *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)* (2017), IEEE, pp. 1–6.

[181] LILLICRAP, T. P., HUNT, J. J., PRITZEL, A., HEESS, N., EREZ, T., TASSA, Y., SILVER, D., AND WIERSTRA, D. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).

[182] LIU, J., LIU, J., AND XIE, R. Reliability-based controller placement algorithm in software defined networking. *Computer Science and Information Systems 13*, 2 (2016), 547–560.

[183] LIU, L., CHENG, Y., CAI, L., ZHOU, S., AND NIU, Z. Deep learning based optimization in wireless network. In *2017 IEEE international conference on communications (ICC)* (2017), IEEE, pp. 1–6.

[184] LIU, L., YIN, B., ZHANG, S., CAO, X., AND CHENG, Y. Deep learning meets wireless network optimization: Identify critical links. *IEEE Transactions on Network Science and Engineering* (2018).

[185] LIU, N., LI, Z., XU, J., XU, Z., LIN, S., QIU, Q., TANG, J., AND WANG, Y. A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)* (2017), IEEE, pp. 372–382.

[186] LIU, S., STEINERT, R., AND KOSTIC, D. Flexible distributed control plane deployment. In *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium* (2018), IEEE, pp. 1–7.

[187] LIVNI, R., SHALEV-SHWARTZ, S., AND SHAMIR, O. On the computational efficiency of training neural networks. In *Advances in neural information processing systems* (2014), pp. 855–863.

[188] LOWE, R., WU, Y., TAMAR, A., HARB, J., ABBEEL, O. P., AND MORDATCH, I. Multi-agent actor-critic for mixed cooperative-competitive environments. In *NeurIPS* (2017), pp. 6379–6390.

[189] LU, Z., PU, H., WANG, F., HU, Z., AND WANG, L. The expressive power of neural networks: A view from the width. In *Advances in neural information processing systems* (2017), pp. 6231–6239.

[190] MADNI, S. H. H., LATIFF, M. S. A., COULIBALY, Y., ET AL. Resource scheduling for infrastructure as a service (iaas) in cloud computing: Challenges and opportunities. *Journal of Network and Computer Applications 68* (2016), 173–200.

[191] MAHMOOD, K., CHILWAN, A., ØSTERBØ, O., AND JARSCHEL, M. Modelling of openflow-based software-defined networks: the multiple node case. *IET Networks 4*, 5 (2015), 278–284.

[192] MAO, B., FADLULLAH, Z. M., TANG, F., KATO, N., AKASHI, O., INOUE, T., AND MIZUTANI, K. Routing or computing? the paradigm shift towards intelligent computer network packet transmission based on deep learning. *IEEE Transactions on Computers 66*, 11 (2017), 1946–1960.

[193] MAO, H., ALIZADEH, M., MENACHE, I., AND KANDULA, S. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks* (2016), ACM, pp. 50–56.

[194] MAO, H., SCHWARZKOPF, M., VENKATAKRISHNAN, S. B., MENG, Z., AND ALIZADEH, M. Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM Special Interest Group on Data Communication* (2019), ACM, pp. 270–288.

[195] MAO, Y., ZHANG, J., AND LETAIEF, K. B. A lyapunov optimization approach for green cellular networks with hybrid energy supplies. *IEEE Journal on Selected Areas in Communications 33*, 12 (2015), 2463–2477.

[196] MARTIN, J., AND NILSSON, A. On service level agreements for ip networks. In *IEEE INFOCOM* (2002), vol. 2, pp. 855–863.

[197] MCCALL, J. Genetic algorithms for modelling and optimisation.

Journal of Computational and Applied Mathematics 184, 1 (2005), 205–222.

[198] METAWA, N., HASSAN, M. K., AND ELHOSENY, M. Genetic algorithm based model for optimizing bank lending decisions. *Expert Systems with Applications 80* (2017), 75–82.

[199] MIKHAYLOV, K., PETAEJAEJAERVI, J., AND HAENNINEN, T. Analysis of capacity and scalability of the lora low power wide area network technology. In *European Wireless 2016; 22th European Wireless Conference* (2016), VDE, pp. 1–6.

[200] MITZENMACHER, M. D. *The Power of Two Random Choices in Randomized Load Balancing*. PhD thesis, PhD thesis, Graduate Division of the University of California at Berkley, 1996.

[201] MNIH, V., BADIA, A. P., MIRZA, M., GRAVES, A., LILLICRAP, T., HARLEY, T., SILVER, D., AND KAVUKCUOGLU, K. Asynchronous methods for deep reinforcement learning. In *ICML* (2016), pp. 1928–1937.

[202] MNIH, V., KAVUKCUOGLU, K., SILVER, D., RUSU, A. A., VENESS, J., BELLEMARE, M. G., GRAVES, A., RIEDMILLER, M., FIDJELAND, A. K., OSTROVSKI, G., ET AL. Human-level control through deep reinforcement learning. *Nature 518*, 7540 (2015), 529–533.

[203] MOAZZENI, S., KHAYYAMBASHI, M. R., MOVAHHEDINIA, N., AND CALLEGATI, F. On reliability improvement of software-defined networks. *Computer Networks 133* (2018), 195–211.

[204] MOENS, H., AND DE TURCK, F. Vnf-p: A model for efficient placement of virtualized network functions. In *10th International Conference on Network and Service Management (CNSM) and Workshop* (2014), IEEE, pp. 418–423.

[205] MOGHADAM, N., LI, H., ZENG, H., AND LIU, L. Lyapunov scheduling and optimization in network coded wireless multicast network. *IEEE Transactions on Vehicular Technology 67*, 6 (2018), 5135–5145.

[206] MOHANASUNDARAM, J. G., TRUONG-HUU, T., AND GURUSAMY, M. Game theoretic switch-controller mapping with traffic variations in software defined networks. In *2018 IEEE Global Communications Conference (GLOBECOM)* (2018), IEEE, pp. 1–6.

[207] MÜLLER, L. F., OLIVEIRA, R. R., LUIZELLI, M. C., GASPARY, L. P., AND BARCELLOS, M. P. Survivor: An enhanced controller placement strategy for improving sdn survivability. In *2014 IEEE Global Communications Conference* (2014), IEEE, pp. 1909–1915.

[208] NACHUM, O., NOROUZI, M., XU, K., AND SCHUURMANS, D. Bridging the gap between value and policy based reinforcement learning. In *Advances in Neural Information Processing Systems* (2017), pp. 2775–2785.

[209] NAGABANDI, A., CLAVERA, I., LIU, S., FEARING, R. S., ABBEEL, P., LEVINE, S., AND FINN, C. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. *arXiv preprint arXiv:1803.11347* (2018).

[210] NAZARI, M., OROOJLOOY, A., SNYDER, L., AND TAKÁC, M. Reinforcement learning for solving the vehicle routing problem. In *NeurIPS* (2018), pp. 9839–9849.

[211] NEELY, M. J. Stability and probability 1 convergence for queueing networks via lyapunov optimization. *Journal of Applied Mathematics 2012* (2012).

[212] NGUYEN, S., ZHANG, M., JOHNSTON, M., AND TAN, K. C. Automatic design of scheduling policies for dynamic multi-objective job

shop scheduling via cooperative coevolution genetic programming. *IEEE Transactions on Evolutionary Computation 18*, 2 (2014), 193–208.

[213] NGUYEN, S., ZHANG, M., AND TAN, K. C. Adaptive charting genetic programming for dynamic flexible job shop scheduling. In *Proceedings of the Genetic and Evolutionary Computation Conference* (2018), ACM, pp. 1159–1166.

[214] NUNES, B. A. A., MENDONCA, M., NGUYEN, X.-N., OBRACZKA, K., AND TURLETTI, T. A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys & Tutorials 16*, 3 (2014), 1617–1634.

[215] OBADIA, M., BOUET, M., ROUGIER, J.-L., AND IANNONE, L. A greedy approach for minimizing sdn control overhead. In *Network Softwarization (NetSoft), 2015 1st IEEE Conference on* (2015), IEEE, pp. 1–5.

[216] OKTIAN, Y. E., LEE, S., LEE, H., AND LAM, J. Distributed sdn controller system: A survey on design choice. *computer networks 121* (2017), 100–111.

[217] OLIVEIRA, R. V., ZHANG, B., AND ZHANG, L. Observing the evolution of internet as topology. In *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications* (2007), pp. 313–324.

[218] PAN, S. J., AND YANG, Q. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering 22*, 10 (2009), 1345–1359.

[219] PARK, J., MEI, Y., NGUYEN, S., CHEN, G., AND ZHANG, M. Investigating a machine breakdown genetic programming approach for dynamic job shop scheduling. In *EuroGP* (2018), Springer, pp. 253–270.

[220] PARTRIDGE, C., MENDEZ, T., AND MILLIKEN, W. RFC 1546: Host anycasting service. *InterNet Network Working Group* (1993).

[221] PENG, Y. *Policy Direct Search for Effective Reinforcement Learning*. PhD thesis, Victoria University of Wellington, New Zealand, 2019.

[222] PFAFF, B., PETTIT, J., KOPONEN, T., JACKSON, E., ZHOU, A., RAJAHALME, J., GROSS, J., WANG, A., STRINGER, J., SHELAR, P., ET AL. The design and implementation of open vswitch. In *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)* (2015), pp. 117–130.

[223] PHEMIUS, K., BOUET, M., AND LEGUAY, J. Disco: Distributed sdn controllers in a multi-domain environment. In *2014 IEEE Network Operations and Management Symposium (NOMS)* (2014), IEEE, pp. 1–2.

[224] POLLARD, D. Asymptopia: an exposition of statistical asymptotic theory. `http://www.stat.yale.edu/pollard/Books/Asymptopia`, 2000.

[225] POPA, L., KUMAR, G., CHOWDHURY, M., KRISHNAMURTHY, A., RATNASAMY, S., AND STOICA, I. Faircloud: Sharing the network in cloud computing. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication* (2012), pp. 187–198.

[226] RASMUSSEN, R. V., AND TRICK, M. A. Round robin scheduling–a survey. *European Journal of Operational Research 188*, 3 (2008), 617–636.

[227] RATH, H. K., REVOORI, V., NADAF, S. M., AND SIMHA, A. Optimal controller placement in software defined networks (sdn) using a non-zero-sum game. In *Proceeding of IEEE International Symposium*

on a World of Wireless, Mobile and Multimedia Networks 2014 (2014), IEEE, pp. 1–6.

[228] ROS, F. J., AND RUIZ, P. M. On reliable controller placements in software-defined networks. *Computer Communications 77* (2016), 41–51.

[229] ROTHLAUF, F. *Design of modern heuristics: principles and application*. Springer Science & Business Media, 2011.

[230] RUDER, S., PETERS, M. E., SWAYAMDIPTA, S., AND WOLF, T. Transfer learning in natural language processing. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Tutorials* (2019), pp. 15–18.

[231] RUIZ-RIVERA, A., CHIN, K.-W., AND SOH, S. Greco: An energy aware controller association algorithm for software defined networks. *IEEE communications letters 19*, 4 (2015), 541–544.

[232] RUNARSSON, T. P., AND YAO, X. Stochastic ranking for constrained evolutionary optimization. *IEEE Transactions on evolutionary computation 4*, 3 (2000), 284–294.

[233] SABAR, N. R., TURKY, A., AND SONG, A. A genetic programming based iterated local search for software project scheduling. In *Proceedings of the Genetic and Evolutionary Computation Conference* (2018), ACM, pp. 1364–1370.

[234] SAFAVIAN, S. R., AND LANDGREBE, D. A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics 21*, 3 (1991), 660–674.

[235] SAHOO, K. S., SAHOO, S., SARKAR, A., SAHOO, B., AND DASH, R. On the placement of controllers for designing a wide area software

defined networks. In *TENCON 2017-2017 IEEE Region 10 Conference* (2017), IEEE, pp. 3123–3128.

[236] SAKIC, E., SARDIS, F., GUCK, J. W., AND KELLERER, W. Towards adaptive state consistency in distributed sdn control plane. In *2017 IEEE International Conference on Communications (ICC)* (2017), IEEE, pp. 1–7.

[237] SALLAHI, A., AND ST-HILAIRE, M. Optimal model for the controller placement problem in software defined networks. *IEEE communications letters 19*, 1 (2014), 30–33.

[238] SALLAHI, A., AND ST-HILAIRE, M. Expansion model for the controller placement problem in software defined networks. *IEEE Communications Letters 21*, 2 (2016), 274–277.

[239] SALOT, P. A survey of various scheduling algorithm in cloud computing environment. *International Journal of Research in Engineering and Technology 2*, 2 (2013), 131–135.

[240] SCHULMAN, J., LEVINE, S., ABBEEL, P., JORDAN, M., AND MORITZ, P. Trust region policy optimization. In *ICML* (2015), pp. 1889–1897.

[241] SCHULMAN, J., MORITZ, P., LEVINE, S., JORDAN, M., AND ABBEEL, P. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438* (2015).

[242] SCHULMAN, J., WOLSKI, F., DHARIWAL, P., RADFORD, A., AND KLIMOV, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[243] SCHWEFEL, H.-P. *Numerical optimization of computer models*. John Wiley & Sons, Inc., 1981.

[244] SCOTT-HAYWARD, S., O'CALLAGHAN, G., AND SEZER, S. Sdn security: A survey. In *Future Networks and Services (SDN4FNS), 2013 IEEE SDN For* (2013), IEEE, pp. 1–7.

[245] SELVI, H., GÜR, G., AND ALAGÖZ, F. Cooperative load balancing for hierarchical sdn controllers. In *2016 IEEE 17th International Conference on High Performance Switching and Routing (HPSR)* (2016), IEEE, pp. 100–105.

[246] SEZER, S., SCOTT-HAYWARD, S., CHOUHAN, P. K., FRASER, B., LAKE, D., FINNEGAN, J., VILJOEN, N., MILLER, M., AND RAO, N. Are we ready for sdn? implementation challenges for software-defined networks. *IEEE Communications Magazine 51*, 7 (2013), 36–43.

[247] SHALEV-SHWARTZ, S., AND BEN-DAVID, S. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[248] SHEELA, K. G., AND DEEPA, S. N. Review on methods to fix number of hidden neurons in neural networks. *Mathematical Problems in Engineering 2013* (2013).

[249] SHOHAM, Y., AND LEYTON-BROWN, K. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press, 2008.

[250] SILVER, D., HUANG, A., MADDISON, C. J., GUEZ, A., SIFRE, L., VAN DEN DRIESSCHE, G., SCHRITTWIESER, J., ANTONOGLOU, I., PANNEERSHELVAM, V., LANCTOT, M., ET AL. Mastering the game of go with deep neural networks and tree search. *nature 529*, 7587 (2016), 484.

[251] SIM, K., AND HART, E. A combined generative and selective hyper-heuristic for the vehicle routing problem. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016* (2016), pp. 1093–1100.

[252] SONG, P., LIU, Y., LIU, T., AND QIAN, D. Flow stealer: lightweight load balancing by stealing flows in distributed sdn controllers. *Science China Information Sciences 60*, 3 (2017), 032202.

[253] SORIA-ALCARAZ, J. A., ÖZCAN, E., SWAN, J., KENDALL, G., AND CARPIO, M. Iterated local search using an add and delete hyper-heuristic for university course timetabling. *Applied Soft Computing 40* (2016), 581–593.

[254] SRIDHARAN, V., GURUSAMY, M., AND TRUONG-HUU, T. Multi-controller traffic engineering in software defined networks. In *2017 IEEE 42nd Conference on Local Computer Networks (LCN)* (2017), IEEE, pp. 137–145.

[255] SRIDHARAN, V., GURUSAMY, M., AND TRUONG-HUU, T. On multiple controller mapping in software defined networks with resilience constraints. *IEEE Communications Letters 21*, 8 (2017), 1763–1766.

[256] SRINIVAS, M., AND PATNAIK, L. M. Genetic algorithms: A survey. *computer 27*, 6 (1994), 17–26.

[257] SUFIEV, H., AND HADDAD, Y. A dynamic load balancing architecture for sdn. In *2016 IEEE International Conference on the Science of Electrical Engineering (ICSEE)* (2016), IEEE, pp. 1–3.

[258] SUH, D., AND PACK, S. Low-complexity master controller assignment in distributed sdn controller environments. *IEEE Communications Letters 22*, 3 (2017), 490–493.

[259] SUTTON, R. S., AND BARTO, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

[260] SUTTON, R. S., MCALLESTER, D. A., SINGH, S. P., AND MANSOUR, Y. Policy gradient methods for reinforcement learning with function

approximation. In *Advances in Neural Information Processing Systems* (2000), pp. 1057–1063.

[261] TALBI, E.-G. *Metaheuristics: from design to implementation*, vol. 74. John Wiley & Sons, 2009.

[262] TALVITIE, E. Self-correcting models for model-based reinforcement learning. In *Thirty-First AAAI Conference on Artificial Intelligence* (2017).

[263] TANHA, M., SAJJADI, D., RUBY, R., AND PAN, J. Capacity-aware and delay-guaranteed resilient controller placement for software-defined wans. *IEEE Transactions on Network and Service Management 15*, 3 (2018), 991–1005.

[264] TELGARSKY, M. Benefits of depth in neural networks. *arXiv preprint arXiv:1602.04485* (2016).

[265] TESAURO, G., JONG, N. K., DAS, R., AND BENNANI, M. N. A hybrid reinforcement learning approach to autonomic resource allocation. In *Autonomic Computing, 2006. ICAC'06. IEEE International Conference on* (2006), IEEE, pp. 65–73.

[266] THRUN, S., AND SCHWARTZ, A. Issues in using function approximation for reinforcement learning. In *Proceedings of the 1993 Connectionist Models Summer School Hillsdale, NJ. Lawrence Erlbaum* (1993).

[267] TIELEMAN, T., AND HINTON, G. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning 4*, 2 (2012), 26–31.

[268] TOOTOONCHIAN, A., AND GANJALI, Y. Hyperflow: A distributed control plane for openflow. In *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking* (2010), pp. 3–3.

[269] TOOTOONCHIAN, A., GORBUNOV, S., GANJALI, Y., CASADO, M., AND SHERWOOD, R. On controller performance in software-defined networks. *Hot-ICE 12* (2012), 1–6.

[270] TORODE, C. Data center facility of the future is a hybrid. `https://searchcio.techtarget.com/feature/Data-center-facility-of-the-future-is-a-hybrid`. Published: 12-Dec-2012. [Online].

[271] UL HUQUE, M. T. I., JOURJON, G., AND GRAMOLI, V. Revisiting the controller placement problem. In *Local Computer Networks (LCN), 2015 IEEE 40th Conference on* (2015), IEEE, pp. 450–453.

[272] UL HUQUE, M. T. I., SI, W., JOURJON, G., AND GRAMOLI, V. Large-scale dynamic controller placement. *IEEE Transactions on Network and Service Management 14*, 1 (2017), 63–76.

[273] VAN HASSELT, H., GUEZ, A., AND SILVER, D. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence* (2016).

[274] VAN LAARHOVEN, P. J., AND AARTS, E. H. Simulated annealing. In *Simulated annealing: Theory and applications*. Springer, 1987, pp. 7–15.

[275] VAPNIK, V. *The nature of statistical learning theory*. Springer science & business media, 2013.

[276] VAZIRANI, V. V. *Approximation algorithms*. Springer Science & Business Media, 2013.

[277] VIZARRETA, P., MACHUCA, C. M., AND KELLERER, W. Controller placement strategies for a resilient sdn control plane. In *2016 8th International Workshop on Resilient Networks Design and Modeling (RNDM)* (2016), IEEE, pp. 253–259.

[278] WANG, C., HU, B., CHEN, S., LI, D., AND LIU, B. A switch migration-based decision-making scheme for balancing load in sdn. *IEEE Access 5* (2017), 4537–4544.

[279] WANG, G., ZHAO, Y., HUANG, J., AND WU, Y. An effective approach to controller placement in software defined wide area networks. *IEEE Transactions on Network and Service Management 15*, 1 (2018), 344–355.

[280] WANG, T., LIU, F., GUO, J., AND XU, H. Dynamic sdn controller assignment in data center networks: Stable matching with transfers. In *Proc. of INFOCOM* (2016).

[281] WANG, T., LIU, F., AND XU, H. An efficient online algorithm for dynamic sdn controller assignment in data center networks. *IEEE/ACM Transactions on Networking 25*, 5 (2017), 2788–2801.

[282] WANG, Z., SCHAUL, T., HESSEL, M., VAN HASSELT, H., LANCTOT, M., AND DE FREITAS, N. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581* (2015).

[283] WATKINS, C. J., AND DAYAN, P. Q-learning. *Machine learning 8*, 3-4 (1992), 279–292.

[284] WHITESON, S. Evolutionary computation for reinforcement learning. In *Reinforcement Learning*. Springer, 2012, pp. 325–355.

[285] WILLIAMS, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning 8*, 3-4 (1992), 229–256.

[286] WU, J., GAO, Z., AND SUN, H. Effects of the cascading failures on scale-free traffic networks. *Physica A: Statistical Mechanics and its Applications 378*, 2 (2007), 505–511.

[287] WU, J., XU, X., ZHANG, P., AND LIU, C. A novel multi-agent reinforcement learning approach for job scheduling in grid computing. *Future Generation Computer Systems 27*, 5 (2011), 430–439.

[288] XIA, W., WEN, Y., FOH, C. H., NIYATO, D., AND XIE, H. A survey on software-defined networking. *IEEE Communications Surveys & Tutorials 17*, 1 (2014), 27–51.

[289] XIE, J., GUO, D., HU, Z., QU, T., AND LV, P. Control plane of software defined networks: A survey. *Computer Communications 67* (2015), 1–10.

[290] XU, Y., CELLO, M., WANG, I.-C., WALID, A., WILFONG, G., WEN, C. H.-P., MARCHESE, M., AND CHAO, H. J. Dynamic switch migration in distributed software-defined networks to achieve controller load balance. *IEEE Journal on Selected Areas in Communications 37*, 3 (2019), 515–529.

[291] YAO, G., BI, J., LI, Y., AND GUO, L. On the capacitated controller placement problem in software defined networks. *IEEE Communications Letters 18*, 8 (2014), 1339–1342.

[292] YAO, H., QIU, C., ZHAO, C., AND SHI, L. A multicontroller load balancing approach in software-defined wireless networks. *International Journal of Distributed Sensor Networks 11*, 10 (2015), 454159.

[293] YAP, K.-K., MOTIWALA, M., RAHE, J., PADGETT, S., HOLLIMAN, M., BALDUS, G., HINES, M., KIM, T., NARAYANAN, A., JAIN, A., ET AL. Taking the edge off with espresso: Scale, reliability and programmability for global internet peering. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (2017), pp. 432–445.

[294] YE, X., CHENG, G., AND LUO, X. Maximizing sdn control resource

utilization via switch migration. *Computer Networks 126* (2017), 69–80.

[295] YEGANEH, S. H., TOOTOONCHIAN, A., AND GANJALI, Y. On scalability of software-defined networking. *IEEE Communications Magazine 51*, 2 (2013), 136–141.

[296] YU, M., REXFORD, J., FREEDMAN, M. J., AND WANG, J. Scalable flow-based networking with difane. *ACM SIGCOMM Computer Communication Review 40*, 4 (2010), 351–362.

[297] ZENG, D., TENG, C., GU, L., YAO, H., AND LIANG, Q. Flow setup time aware minimum cost switch-controller association in software-defined networks. In *2015 11th International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness (QSHINE)* (2015), IEEE, pp. 259–264.

[298] ZHANG, B., WANG, X., AND HUANG, M. Multi-objective optimization controller placement problem in internet-oriented software defined network. *Computer Communications 123* (2018), 24–35.

[299] ZHANG, B., WU, Y., LU, J., AND DU, K.-L. Evolutionary computation and its applications in neural and fuzzy systems. *Applied Computational Intelligence and Soft Computing 2011* (2011).

[300] ZHANG, K., YANG, Z., LIU, H., ZHANG, T., AND BAŞAR, T. Fully decentralized multi-agent reinforcement learning with networked agents. *arXiv preprint arXiv:1802.08757* (2018).

[301] ZHANG, Q., ZHANI, M. F., BOUTABA, R., AND HELLERSTEIN, J. L. Harmony: Dynamic heterogeneity-aware resource provisioning in the cloud. In *2013 IEEE 33rd International Conference on Distributed Computing Systems* (2013), IEEE, pp. 510–519.

[302] ZHANG, Q., ZHU, Q., ZHANI, M. F., ET AL. Dynamic service placement in geographically distributed clouds. *IEEE Journal on Selected Areas in Communications 31*, 12 (2013), 762–772.

[303] ZHANG, T., GIACCONE, P., BIANCO, A., AND DE DOMENICO, S. The role of the inter-controller consensus in the placement of distributed sdn controllers. *Computer Communications 113* (2017), 1–13.

[304] ZHAO, J., QU, H., ZHAO, J., LUAN, Z., AND GUO, Y. Towards controller placement problem for software-defined network using affinity propagation. *Electronics Letters 53*, 14 (2017), 928–929.

[305] ZHENG, K., WANG, L., YANG, B., SUN, Y., AND UHLIG, S. Lazyctrl: A scalable hybrid network control plane design for cloud data centers. *IEEE Transactions on Parallel and Distributed Systems 28*, 1 (2017), 115–127.

[306] ZHONG, Q., WANG, Y., LI, W., AND QIU, X. A min-cover based controller placement approach to build reliable control network in sdn. In *NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium* (2016), IEEE, pp. 481–487.

[307] ZHOU, Y., WANG, Y., YU, J., BA, J., AND ZHANG, S. Load balancing for multiple controllers in sdn based on switches group. In *2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS)* (2017), IEEE, pp. 227–230.

[308] ZHOU, Y., ZHENG, K., NI, W., AND LIU, R. P. Elastic switch migration for control plane load balancing in sdn. *IEEE Access 6* (2018), 3909–3919.

[309] ZHU, X., AND GOLDBERG, A. B. Introduction to semi-supervised learning. *Synthesis lectures on artificial intelligence and machine learning 3*, 1 (2009), 1–130.