CONTROL DESIGNS AND REINFORCEMENT LEARNING-BASED MANAGEMENT FOR SOFTWARE DEFINED NETWORKS

ZIYAO ZHANG

A Thesis Submitted in Fulfilment of Requirements for the Degree of

Doctor of Philosophy of Imperial College London and

Diploma of Imperial College

Communications and Signal Processing Group

Department of Electrical and Electronic Engineering

Imperial College London

2020

Declaration of Originality

This is to certify that to the best of my knowledge, the content of this thesis is my own work. This thesis has not been submitted for any degree or other purposes.

I certify that the intellectual content of this thesis is the product of my own work and that all the assistance received in preparing this thesis and sources have been acknowledged.

Ziyao Zhang

Abstract

In this thesis, we focus our investigations around the novel software defined networking (SDN) paradigm. The central goal of SDN is to smoothly introduce centralised control capabilities to the otherwise distributed computer networks. This is achieved by abstracting and concentrating network control functionalities in a logically centralised control unit, which is referred to as the SDN controller. To further balance between centralised control, scalability and reliability considerations, distributed SDN is introduced to enable the coexistence of multiple physical SDN controllers. For distributed SDN, networking elements are grouped together to form various domains, with each domain managed by an SDN controller. In such a distributed SDN setting, SDN controllers of all domains synchronise with each other to maintain logically centralised network views, which is referred to as controller synchronisation. Centred on the problem of SDN controller synchronisation, this thesis specifically aims at addressing two aspects of the subject as follows. First, we model and analyse the performance enhancements brought by controller synchronisation in distributed SDN from a theoretical perspective. Second, we design intelligent controller synchronisation policies by leveraging existing and creating new Reinforcement Learning (RL) and Deep Learning (DL)-based approaches.

In order to understand the performance gains of SDN controller synchronisation from a fundamental and analytical perspective, we propose a two-layer network model based on graphs to capture various characteristics of distributed SDN networks. Then, we develop two families of analytical methods to investigate the performance of distributed SDN in relationship to network structure and the level of SDN controller synchronisation. The significance of our analytical results is that

they can be used to quantify the contribution of controller synchronisation level, in improving the network performance under different network parameters. Therefore, they serve as fundamental guidelines for future SDN performance analyses and protocol designs.

For the designs of SDN controller synchronisation policies, most existing works focus on the engineering-centred system design aspect of the problem for ensuring anomaly-free synchronisation. Instead, we emphasise on the performance improvements with respect to (w.r.t.) various networking tasks for designing controller synchronisation policies. Specifically, we investigate various scenarios with diverse control objectives, which range from routing related performance metric to other more sophisticated optimisation goals involving communication and computation resources in networks. We also take into consideration factors such as the scalability and robustness of the policies developed. For this goal, we employ machine learning techniques to assist our policy designs. In particular, we model the SDN controller synchronisation policy as serial decision-making processes and resort to RL-based techniques for developing the synchronisation policy. To this end, we leverage a combination of various RL and DL methods, which are tailored for handling the specific characteristics and requirements in different scenarios. Evaluation results show that our designed policies consistently outperform some already in-use controller synchronisation policies, in certain cases by considerable margins. While exploring existing RL algorithms for solving our problems, we identify some critical issues embedded within these algorithms, such as the enormity of the state-action space, which can cause inefficiency in learning. As such, we propose a novel RL algorithm to address these issues, which is named state action separable reinforcement learning (sasRL). Therefore, the sasRL approach constitutes another major contribution of this thesis in the field of RL research.

To my family
for their endless love and support

Acknowledgments

First of all, I would like to extend my sincere gratitude and appreciation to my PhD supervisor Professor Kin K. Leung for his guidance and supports in the past few years. From his dedicated mentorship and teachings, I learned not only how to address specific issues in my area of research, but also the way to approach, frame, break down, and present results of complex research problems in general. In addition to his academic role, Kin has been a father figure to me, who always supports and looks after me. Kin treats me as his junior colleague and interacts with me in a candid, understanding, and respectful manner. Another very important thing Kin taught me was how to behave professionally in the scientific research community. He sets an excellent example of an honest, responsible, courteous, and trustworthy researcher. All these precious qualities I learned from Kin shall accompany me moving into future careers.

I would also like to express my sincere thanks to Dr. Liang Ma, who was a research scientist at IBM T.J. Watson Research Center in New York. Throughout my PhD years, Liang has been my mentor and main collaborator. Liang taught me how to conduct research. From tedious and detailed things such as the software programs to draw diagrams or manage paper references, to high-level and delicate tasks such as problem formulation and positioning, Liang taught me step by step. In particular, I have been very impressed by the clear logic and mathematical rigour he possesses, while approaching research problems. I benefited tremendously from Liang's strong analytical abilities, as I learned to turn wild ideas into concrete and tractable research problems. I would also like to express my sincere thanks to him for hosting two unforgettable research internships for me at IBM T.J. Watson Research Center in

New York during 2018 and 2019. Liang is a kind, caring, and considerate person. Over the years, we have established personal friendship, which I value and cherish.

I would also like to take this opportunity to thank Dr. Mudhakar Srivatsa, who was my manager during my internships with IBM. I really enjoyed the technical discussions I had with Mudhakar, and I appreciate his advice to me on various subjects. Moreover, special thanks to my other collaborators, including Dr. Konstantinos Poularakis and Professor Leandros Tassiulas at Yale University, Mr. Jeremy Tucker at UK Defence Science and Technology Laboratory, Dr. Ananthram Swami at US Army Research Laboratory, Dr. Franck Le at IBM T.J. Watson Research Center, and Dr. Sastry Kompella at US Naval Research Laboratory. It has been a pleasure and a great honour to work with you all and publish research papers of high calibre.

Thanks to Dr. Deniz Gunduz from Imperial College London and Dr. George Parisis from University of Sussex, who served as my viva examiners, for their thorough examination of the thesis and insightful comments and suggestions.

With much appreciation, I would like to acknowledge the funding source of my PhD programme, namely the International Technology Alliance in Distributed Analytics and Information Sciences (DAIS-ITA) research project. The UK-US DAIS-ITA project has been a great platform, which provided me with precious opportunities to work with many other world-class researchers and peers.

Finally, my family has always been my backbone and the fundamental source of energy in my life. I cannot imagine completing my PhD programme without their endless love, encouragement and companionship. It is a great pleasure to dedicate this thesis to my mother, Xuemei Li, my father, Renhua Zhang, and my fiancée, Xintong Ren.

Ziyao Zhang

Imperial College London

August, 2020

Contents

External Publications

- **Journal Papers**
  1. **Z. Zhang**, L. Ma, K. K. Leung, F. Le, S. Kompella, L. Tassiulas, "How Advantageous Is It? An Analytical Study of Controller-Assisted Path Construction in Distributed SDN", *IEEE/ACM Transactions on Networking, Vol. 27, No. 4, pp. 1643-1656, July 2019*.
  2. **Z. Zhang**, L. Ma, K. K. Leung, K. Poularakis, L. Tassiulas, J. Tucker, A. Swami, "Controller Synchronization by Multi-Agent Reinforcement Learning and Temporal Data Enhancement for Distributed SDN", *preparing for submission*.
  3. **Z. Zhang**, L. Ma, K. K. Leung, F. Le, "More Is Not Always Better: An Analytical Study of Controller Synchronizations in Distributed SDN", *submitted to IEEE/ACM Transactions on Networking*.
  4. L. Ma, **Z. Zhang**, M. Srivatsa, "Neural Network Tomography", *submitted to IEEE/ACM Transactions on Networking*.

- **Conference Papers**
  1. **Z. Zhang**, L. Ma, K. K. Leung, K. Poularakis, M. Srivatsa, "State Action Separable Reinforcement Learning", *to appear in Proc. of IEEE International Conference on Big Data (IEEE BigData)*, 2020, online.
  2. **Z. Zhang**, L. Ma, K. K. Leung, K. Poularakis, M. Srivatsa, Franck Le, "Efficient Reinforcement Learning with Implicit Action Space", *in Proc. of Annual Fall Meeting of International Technology Alliance*, 2020, online.
  3. **Z. Zhang**, L. Ma, K. Poularakis, K. K. Leung, J. Tucker, A. Swami, "MACS:

Deep Reinforcement Learning based SDN Controller Synchronization Policy Design", *in Proc. of IEEE nternational Conference on Network Protocols (IEEE ICNP)*, 2019, Chicago, IL, United States. (**Travel Grant Award**)

4. **Z. Zhang**, L. Ma, K. Poularakis, K. K. Leung, L. Wu, "DQ Scheduler: Deep Reinforcement Learning Based Controller Synchronization in Distributed SDN", *in Proc. of IEEE International Conference on Communications (IEEE ICC)*, 2019, Shanghai, China. (**Best Paper Award**)
5. **Z. Zhang**, Q. Qin, L. Ma, K. Poularakis, F. Le, K. K. Leung, S. Kompella, L. Tassiulas, "DRL based SDN Controller Synchronization Policy Design for joint Communication and Computation Resource Optimisations", *in Proc. of Annual Fall Meeting of International Technology Alliance*, 2019, Weybridge, United Kingdom.
6. **Z. Zhang**, L. Ma, K. Poularakis, K. K. Leung, L. Tassiulas, J. Tucker, "Q-placement: Reinforcement-Learning-Based Service Placement in Software-Defined Networks", *in Proc. of International Conference on Distributed Computing Systems (IEEE ICDCS)*, 2018, Vienna, Austria. (**Travel Grant Award**)
7. **Z. Zhang**, L. Ma, K. Poularakis, K. K. Leung, J. Tucker, A. Swami, "Routing Performance in Distributed SDN under Synchronization Constraint", *in Proc. of Annual Fall Meeting of International Technology Alliance*, 2018, Armonk, NY, United States.
8. L. Ma, **Z. Zhang**, B. Ko, M. Srivatsa, K. K. Leung, "Resource management in distributed SDN using reinforcement learning", *in Proc. SPIE 10635, Ground/Air Multisensor Interoperability, Integration, and Networking for Persistent ISR IX, 106350M*, 2018, Orlando, FL, United States.
9. **Z. Zhang**, L. Ma, K. K. Leung, "On Quantifying Performance Enhancement of Distributed SDN Architecture", *in Proc. of Annual Fall Meeting of International Technology Alliance*, 2017, London, United Kingdom.

List of Figures

Abbreviations

AI	Artificial Intelligence
APC	Average Path Cost
AS	Autonomous System
CS	Complete Synchronization
DL	Deep Learning
DNN	Deep Neural Network
DRL	Deep Reinforcement Learning
ISP	Internet Service Provider
LSTM	Long-Short Term Memory
MDP	Markov Decision Process
MRP	Markov Reward Process
MS	Minimum Synchronization
POMDP	Partially Observable Markov Decision Process
PS	Partial Synchronization
RCPC	Routing Cluster-based Path Construction
RDPN	Randomised Degree-Preserving Network
RL	Reinforcement Learning
SAV	State-Action-Value
SDN	Software-Defined Networking/ Software-Defined Network
SS	Self-domain Synchronization
STV	State-Transition-Value

CHAPTER 1

Introduction

1.1 Overview and Motivations

1.1.1 Software-Defined Networking (SDN) and the Controller Synchronisation Problem

Software-Defined Networking (SDN) [1], a newly-deployed networking architecture [2, 3], significantly improves the network performance due to its programmable network management, easy reconfiguration, and on-demand resource allocation, which has therefore attracted considerable research interests. One key attribute that differentiates SDN from classic networks is the separation of the SDN's data and control planes. Specifically, in SDN, all control functionalities are implemented and abstracted on the control plane for operational decision making, e.g., flow construction and resource allocation, while the data plane only passively executes the instructions received from the control plane. For a canonical SDN architecture, all network control decisions are made in the control plane by a logically centralised control entity, called *SDN controller*. Since the logically centralised SDN controller has the full knowledge of network status, it is able to make globally optimal decisions. Yet, such centralised control suffers from major scalability issues. In particular, as networks grow, the number of flow requests and operational constraints are likely to increase drasticly. The high requirement on computation and communication may impose substantial burden on the SDN controller, potentially resulting in significant

performance degradation (e.g., delays) or even network failures [4].

In this regard, distributed SDN is proposed [5, 6, 7, 8, 9] to balance the centralised and distributed controls. Specifically, a distributed SDN network is composed of a set of subnetworks, referred to as *domains*, each managed by an independent SDN controller. Moreover, each domain contains several gateways connecting to some other domains; such inter-connected domains then form the distributed SDN architecture. In the distributed SDN, if controllers do not communicate with each other regarding the network status of their own domains, then, the distributed SDN can be regarded as reduced to the classical multi-AS (Autonomous Systems) network, where the network flows are managed by distributed protocols, such as the Interior Gateway Protocol (IGP) and Border Gateway Protocol (BGP) protocols. Nevertheless, to take advantage of centralised network in the distributed SDN architecture, controllers are expected to exchange information via proactive probing or passive listening. Such additional status information at each controller, called the *synchronised information*, can assist in enhancing decision making for inter-domain tasks. In distributed SDN, network performance relies heavily on the inter-controller synchronisation level, which refers to the amount of information controllers exchange with each other and the frequency of such exchanges. Since complete synchronisation among controllers, i.e., each controller knows the network status in all other domains, will incur high synchronisation overheads especially in large networks, practical distributed SDN networks can only afford partial inter-domain synchronisation and allow temporary inconsistency in controllers' network views, which is known as the *eventual consistency model* [10].

1.1.2 Limitation of Existing Work and Our Motivation

For partial synchronisation under the eventual consistency model, this thesis aims at addressing two subjects which are largely lacking in existing literatures. First,

we realise that most existing works focus on engineering-based system designs and implementations for distributed SDN. For instance, information sharing algorithms are proposed in [6, 7] for implementing common traffic policies among various domains. Similarly, frameworks are designed in [8, 9], aiming to enable inter-domain routing selection via network status exchanges. However, one fundamental question regarding the distributed SDN architecture has generally been ignored: *How does the network performance in distributed SDN relate to controller synchronisation levels and network structural properties?* It is possible that under certain network conditions, the benefits of increasing the synchronisation level is only marginal. Without such fundamental understandings, it is difficult to justify the existing proposals for controller synchronisation policy design in distributed SDN, which may involve complicated mechanisms and protocols. We, therefore, exploit analytical methods to investigate this unsolved yet critical problem in the distributed SDN paradigm, aiming at quantifying the performance metric and its relationship with various network parameters and controller synchronisation levels.

Second, regarding the eventual consistency model, existing works have identified and addressed some serious anomalies arising from controllers' inconsistent network views, such as loopholes [11], blackholes [12], and other problems caused by policy inconsistencies [13]. Yet, despite these efforts aimed at eliminating inconsistency-caused anomalies, *we have not seen any notable proposals on fine-grained controller synchronisation policy designs, which are tailored for SDN applications with specific performance metrics.* The urgency to fill this gap is especially pronounced when SDN technologies are discussed in a wider range of contexts, where advanced applications are developed on top of SDN-enabled 5G, smart grid, and ISP networks; all these cases require the support of new and finer-grained controller synchronisation models [14]. In this regard, we approach the controller synchronisation problem with the aim of developing fine-grained controller synchronisation policies for

enhancing given performance metrics. Complementary to the existing works that eliminate critical errors in the controller synchronisation process; our work in this thesis is performance-focused, for which we look at how controller(s) should synchronise with each other at certain time steps, so that the given performance metric is maximised.

To this end, we leverage a combination of RL [15] and DL [16] techniques to design control and management strategies for distributed SDN. Based on our experiences, harnessing the power of artificial intelligence (AI) for network control and management in SDN is especially appealing for the reasons as follows. First, the abundance of network data made available by SDN switches through the OpenFlow protocol [17] builds up a pool of past experiences which are the ideal "trial-and-error" data, upon which RL algorithms learn. Second, distributed SDN domains can be highly heterogeneous; as such, SDN networks are complex systems. Therefore, accurately modelling such systems becomes extremely challenging, as we shall show in the first part of the thesis. In addition, due to the modelling challenges and potentially stringent assumptions required, coupled with the fact that network conditions are constantly evolving, traditional optimisation techniques are less suitable for optimising control and management policies in distributed SDN. In light of this, the model-free RL-based approaches are especially attractive, as they come without any constraints on network's structure or its dynamicity, thus adaptable for handling real-world SDN networks.

1.2 Research Objectives and Methodologies

1.2.1 Theoretical Performance Quantification for SDN Controller Synchronisation

To quantify the performance of distributed SDN and understand how it relates to controller synchronisation levels and network structural properties, we first propose a graph based network topological model to capture the intra-/inter-domain connections in distributed SDN. Based on this network topological model, we further associate preference levels to links, which, in practice, reflects the quality of the link with respect to (w.r.t) the networking applications. Such graph based network model is generic in that it only requires node degree/link preference distributions and the number of gateways in each domain as the input parameters, i.e., they are independent of any specific graph models. Using this network model, we then derive analytical expressions of the network performance focusing on the average cost of the constructed paths, i.e. average path cost (APC), w.r.t. random flow requests. The concept of APC is a generalisation, which captures a wide range of additive performance metrics, see Section 2.3 for more discussions. In particular, we develop two families of analytical methods to quantify the performance metric as a function of controller synchronisation levels and other network structural properties.

For the first family of analytical methods (Approach I), we define four canonical controller synchronisation scenarios, i.e., Minimum synchronisation (MS), Self-domain synchronisation (SS), Partial synchronisation (PS), and Complete synchronisation, for quantising the levels of controller synchronisation to assist our analysis. Then, we base our main analysis on the Randomised Degree-Preserving Network (RDPN), which is a special surrogate network generated by combining various network domains to make our analysis tractable. In addition, to capture different levels of granularity of network information, we also define two families of networks, i.e.,

network with uniform and non-uniform link preferences. The main difference between them is the dynamicity of link preference, where in the former case controllers do not specify any preference for links due to the lack of network status information, thus all links have equal link preferences; in the latter case, however, controllers assign preferences to links to achieve control objectives based on up-to-date network status information collected. Based on the preparatory definitions mentioned above, our analysis first establishes an asymptotic expression to highlight the relationship between the performance metric and dominant parameters. Then, detailed analyses are conducted to derive fine-grained characterisations of the performance metric for the two families of networks under the four canonical synchronisation levels. Analytical results reveal the relative contributions of different parameters to the performance metric. For instance, we conclude that controller synchronisation plays key role in improving inter-domain routing when there are moderate amount of inter-domain connections, and the benefit of controller synchronisation dwindles as synchronisation level increases.

For the second family of analytical methods (Approach II), we aim at achieving the same goal as Approach I does. However, the main differences here lie in the methods used and the granularities of the results obtained. In particular, instead of defining four canonical controller synchronisation levels, in Approach II we develop analytical methods that can deal with arbitrary controller synchronisation levels. The trade-off here is that the results obtained with Approach II are coarser-grained, which are in the form of bounds on the performance metric, compared to the finer-grained but complex expressions developed with Approach I. Specifically, Approach II develops the analytical lower bounds for the performance metric, i.e., the APC. We further quantify the tightness of the developed APC lower bound expressions, which shows that it can approach the actual APC value under some conditions. Then, based on the analytical results, again we draw some revelations that relate the quality of

the constructed paths to various network structural parameters and the controller synchronisation levels. By all these theoretical results, we prove that the contribution of network synchronisation levels depends on specific network parameter settings, which therefore provides insights into real network protocol design. Finally, to validate the accuracy of the derived analytical expressions, they are compared against evaluation results obtained using simulation networks constructed using both real and synthetic network datasets.

1.2.2 Reinforcement Learning-based Control and Management in SDN

The second half of this thesis is dedicated to developing control and management policies in the distributed SDN networks by employing DL and RL techniques. Indeed, our work in this thesis confirm that the RL-based approaches are suitable candidates for designing control and management policies for SDN, which includes problems ranging from relatively simple inter-domain routing task to scheduling controller synchronisations for optimising communication, and/or computation resources, as illustrated as follows.

First, we start our investigation by exploring how RL-based controller synchronisation policy design can assist inter-domain routing tasks. To this end, we design *Deep-Q (DQ) Scheduler*, an RL-based algorithm implemented using deep neural networks (DNN), whose goal is to generate controller synchronisation schedules for enhancing inter-domain routing performance.

Although evaluations show that DQ Scheduler renders promising results compared to other schemes which do not involve learning, there are several simplifying assumptions which limit its applicability in more complex tasks. As a follow-up to DQ Scheduler, we then develop *multi-armed cooperative synchronisation (MACS)*, to address the limitations of DQ Scheduler. Compared to the former, MACS's capa-

bilities improve significantly in the following aspects. First, MACS allows synchronisation of finer-grained information among controllers. Second, instead of only focusing on inter-domain routing tasks, MACS enables more networking applications to take advantage of controller synchronisation. As the name suggested, the improvements brought by MACS is achieved by the specially designed neural network structure, which is able to handle the increased complexity in training data.

The MACS operates on the assumption that a central node (one of the controllers or an independent control unit) in the SDN network is responsible for learning the synchronisation policy based on necessary information gathered from all controllers in the network. It is also assumed that the central node can always obtain needed information for the RL task. However, such a centralised learning scheme can be vulnerable since it creates a single point of failure. On the other hand, it is possible that the central node may not be able to receive all information required from other controllers, due to network constraints or anomalies. Therefore, in the follow-up work we address these two vulnerabilities and further improve MACS's scalability and robustness. In this regard, we propose an upgraded system of MACS, named *synchronisation via multi-agent reinforcement-learning and temporal-data-enhancement (SMART)*. For SMART, we conduct a theoretical analysis to identify the conditions upon which a decentralised policy can perform comparably as the centralised counterpart does. Moreover, to counter unstable network environments under which the loss of data may occur, we employ deep learning techniques for data enhancement, which is used in conjunction with our designed deep reinforcement learning networks for policy generations.

Finally, although the aforementioned work suggests that RL techniques are very suitable and useful for developing SDN control and management policies, we identify several issues that are embedded in existing DRL algorithms. Specifically, for conventional RL formulations, Markov decision process (MDP) and state-action-

value (SAV) function are the basis for problem modelling and policy evaluation. With such a formulation, a major challenge we face is that the enormity of state/action space causes inefficiency and difficulties for the neural network in accurately approximating the SAV function. We, therefore, propose a new learning paradigm, State Action Separable Reinforcement Learning (sasRL), wherein the action space is decoupled from the value function learning process for higher efficiency. Then, a light-weight transition model is learned to assist the agent to determine the action that triggers the associated state transition. In the high-level, sasRL is designed to improve learning efficiency by breaking down a complicated model-free RL problem into a simpler model-free RL problem and a supervised learning problem. sasRL is a generic RL paradigm, which can work with several RL problem beyond the controller synchronisation problem we investigated in SDN.

1.3 Summary of Contributions

To the best of our knowledge, for the two focused research areas of this thesis, i.e., theoretical performance quantifications w.r.t. controller synchronisation and the RL-based control and management policy designs for distributed SDN, our work contributes novel approaches and ideas in the respective fields. Specifically, our contributions are summarised as follows.

1) We propose a generic two-layer network model to capture intra-/inter-domain connections, link preference, and other properties of the distributed SDN networks. On top of the network model, we use the APC of the constructed paths as the performance metric to develop the asymptotic expression of the APC under any given synchronisation levels. Then, for each of the four canonical controller synchronisation scenarios defined, we integrate dynamic link preference levels and develop the corresponding fine-grained analytical expression of the APC.

2) In addition to the fine-grained but complex analytical expressions of APC which we derive, we also develop the analytical lower bounds of the APC, which are linear or even logarithmic functions of network structural or synchronisation-related parameters. Moreover, we quantify the tightness of the APC lower bound expressions developed, which shows that the APC lower bound can approach the actual APC value under some conditions. Based on the analytical results, we draw revelations that relate the qualities of the constructed paths to various network structural parameters and the controller synchronisation levels. Insights for protocol design are also provided.

3) We design *Deep-Q (DQ) Scheduler*, a DRL-based algorithm to determine controller synchronisation schedules for enhancing inter-domain routing performance in distributed SDN.

4) To address the limitation of DQ Scheduler, we design customised DRL structure tailored to enable and support finer-grained controller synchronisation applications, which is called MACS.

5) For higher scalability and robustness of the controller synchronisation policies developed by RL-based approaches, we conduct a theoretical analysis to identify the conditions upon which a distributed policy can perform as well as the centralised counterpart does. We also propose to employ deep learning techniques for data enhancement, to counter unstable network environments under which the loss of training data may occur. Based on these considerations, we further propose a controller synchronisation policy derivation framework for distributed SDN, which is named SMART.

6) We design a new generic RL paradigm, named state action separable reinforcement learning (sasRL), to address the issue of learning inefficiency with existing RL algorithms. We conduct experiments on real-world scenarios to compare the performance of sasRL to other state-of-the-art RL algorithms, which demonstrate its

superior performance.

1.4 Thesis Organisation

The organisation of the rest of the thesis is as follows.

In Chapter 2, we propose a graph-based network model for modelling the distributed SDN architecture. Chapters 3-4 present our analytical results on quantifying the network performance of SDN and its relationship to controller synchronisation levels and network structural properties. Starting from Chapter 5, the focus shifts to control and management of distributed SDN via RL and DL techniques. In particular, Chapters 5-7 centre on the designs of controller synchronisation policies with various control and management objectives. In Chapter 8, we propose a new RL paradigm for addressing the learning inefficiency issues commonly seen in existing RL algorithms. Finally, Chapter 9 concludes the thesis and discusses plans for future work.

The proofs for lemmas, theorems, and corollaries in Chapter 3, Chapter 4 and Chapter 8 are documented in Appendix A.1, Appendix A.2, and Appendix A.3, respectively. Appendix B provides details on the experiments in Chapter 8. Note that the numbers at the end of each reference item in the Bibliography are the page numbers of the pages where the reference is cited.

CHAPTER 2

The Modelling and Formulation of Distributed SDN

2.1 Centralised and Distributed SDN

In the high-level, the central idea of the SDN paradigm is to introduce centralised control over distributed networks, to enable flexible network management and support novel networking applications. This is achieved by adding central control unit to the network, called the SDN controller, which can interact with all networking elements in the network. With such a setting, SDN controller collects status information from networking elements in the network, which is then used to assist the controller in making control decisions. In a fully centralised SDN network, the only physical SDN controller is responsible for managing all networking elements in the network, which scales poorly and creates a single node of failure. Fig. 2.1 is an example of a centralised SDN with one physical SDN controller.

To address the scalability issue created by the centralised SDN paradigm, distributed SDN is therefore proposed. Instead of having only one physical SDN controller in the network, distributed SDN pools networking elements into different groups, with each group managed by one physical SDN controller. The networking elements that are managed by the same physical SDN controller together form a *domain*. All SDN domains then constitute the *data plane*, whereas all *domain controllers* constitute the *control plane*. For instance, Fig. 2.2 shows an distributed SDN

with four domains in the data plane and four domain controllers in the control plane. See Section 3.2.2 for more discussions on related work for distributed SDN.

Under the distributed SDN architecture, the physically distributed controllers synchronise with each other to maintain a logically centralised network view, which is referred to as *controller synchronisation*. Since complete synchronisation among controllers, i.e., all controllers always maintain the same global view, incurs high costs especially in large networks[18, 19], most practical distributed SDN networks can only afford partial inter-controller synchronisations and allow temporary inconsistency in controllers' network view, which is known as the *eventual consistency model* [10]. In this thesis, except for the analysis of full synchronisation among SDN controllers (i.e., CS in Chapter 3), our investigations assume the eventual consistency model.

The types of information being synchronised among domain controllers depend on the requirements of networking tasks and applications. Due to SDN's exceptional ability in fine-grained control and information gathering, potentially many types of network status information can be synchronised among controllers. In the following chapters, we specifically define how controllers synchronise with each other to assist the corresponding networking tasks.

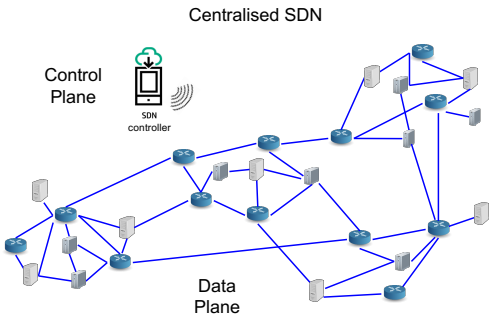


Figure 2.1: Centralised SDN

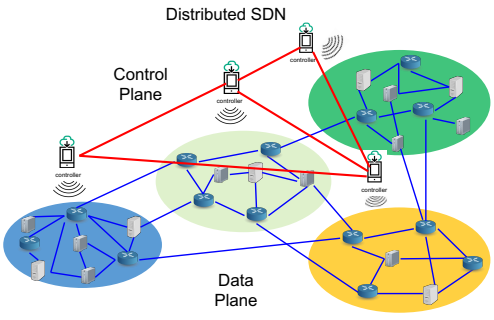


Figure 2.2: Distributed SDN

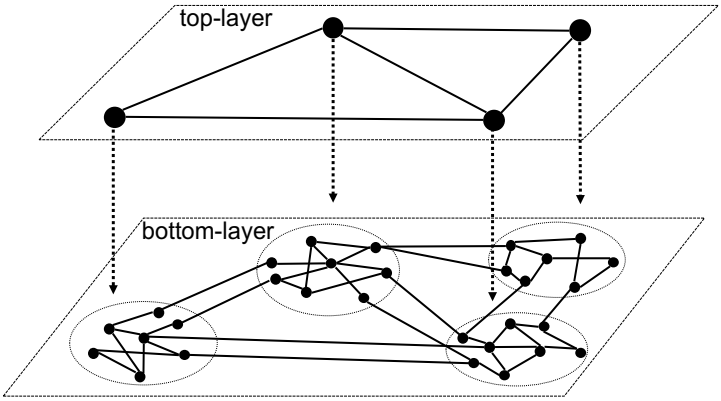


Figure 2.3: Two-layer network model: Top-/bottom-layers abstract domain-wise topology/all physical connections, respectively.

2.2 Network Model for Distributed SDN

This section describes the network model we employ for modelling and constructing the distributed SDN, which lays the foundation for the analyses in Chapter 3 and Chapter 4.

We formulate the distributed SDN network as an undirected graph according to a two-layer network model (Fig. 2.3), where (i) the top-layer abstracts the inter-domain connections, and (ii) the bottom-layer characterises physical connections among all network elements under the inter-domain connection constraint in the top-layer.

First, each domain is characterised by an undirected graph with n nodes in the bottom-layer; these n nodes are connected following a given *intra-domain degree distribution*, which is the distribution of the number of neighbouring nodes of an arbitrary node within the same domain. [1] We also assume that such intra-domain degrees across all domains are independently and identically distributed (i.i.d.). The graph of each domain is referred to as *intra-domain topology*.

Then, the top-layer, which characterises the connections among domains in distributed SDN network, is a graph $\mathcal{G}_d = (V_d, E_d)$ (V_d/E_d: set of vertices/edges): (i)

[1] In one domain, some nodes may have connections to other domains; such external connections are not considered in the concept of intra-domain degree.

each vertex in V_d represents a domain, (ii) two vertices are connected by an edge if and only if their corresponding domains are directly connected by communication links. We refer to $\mathcal{G}_d$ as the *domain-wise topology* in the sequence. Similar to the model for intra-domain topology, it is assumed that the domain-wise topology, i.e., the top-layer graph, is formed following a given domain-wise degree distribution. A *domain-wise path* from a vertex to another vertex is the sequence of vertices traversed in the top-layer graph during the routing process. The existence of an edge in E_d connecting two vertices $v_1, v_2 \in V_d$ in the domain-wise topology implies that the two network domains corresponding to v_1 and v_2 are connected.

Next, we describe how inter-domain connections are established under our network model. In particular, for each $e \in E_d$ with end-points corresponding to domains $\mathcal{A}_i$ and $\mathcal{A}_j$, we (i) uniformly randomly select two nodes w_1 from $\mathcal{A}_i$ and w_2 from $\mathcal{A}_j$ and connect these two nodes if link w_1w_2 does not exist, and (ii) repeat such link construction process between $\mathcal{A}_i$ and $\mathcal{A}_j$ β times. By this link construction process, the bottom-layer network topology $\mathcal{G} = (V, E)$ is therefore formed (V/E: set of nodes/links in $\mathcal{G}$, $|V| = n|V_d|$) (Fig. 2.3). Without loss of generality, we assume that all inter/intra-domain topologies are *connected graphs*.

Note that the above process indicates that the i-th selected link may overlap with existing links (i.e., the same end-points); therefore, parameter β represents the maximum number of links between any two domains. In each domain, nodes having connections to other domains are called *gateways*. Hence, if domains $\mathcal{A}_i$ and $\mathcal{A}_j$ are connected in the domain-wise topology, then the expected number of gateways in $\mathcal{A}_i$ connecting to $\mathcal{A}_j$ is denoted by γ. According to the inter-domain connection method described above, the probability that a node is not selected after β selections is $(1-1/n)^\beta$. Therefore, the expectation of the number of gateway nodes connecting to the other domain is approximated by $\gamma \approx n(1-(1-1/n)^\beta)$.

Remark: Our two-layer network model is generic in that the input can be any

domain-wise topology and node degree distribution, empirical or extracted from real networks. Note that in practice, gateways are a fixed set of nodes in a domain; our random gateway selection only indicates that the gateway locations can be anywhere in the context of network graphs.

2.2.1 Data Plane

We exploit graph $\mathcal{G}$ generated by the two-layer network model in Section 2.2 to represent the data plane of the distributed SDN. Specifically, a node/link exists in $\mathcal{G}$ if and only if it can be used for data transmission in the network. In addition to the ability to transmit data, a data plane node can also be a general purpose server with computation and storage capabilities to provide virtualised network services. Under the SDN paradigm, the networking elements in the data plane have two basic functions. First, they communicate with their corresponding domain controller for reporting their status information to assist the latter in making control decisions. This can be achieved by SDN's northbound APIs (e.g., via the OpenFlow protocol [20]). Second, they receive and passively execute the instructions given by the corresponding domain controller, which is facilitated by the controller's southbound interface.

2.2.2 Control Plane

As discussed in Section 2.1, each domain contains one logical SDN controller, referred to as the domain controller, which carries out control operations and facilitates information sharing. Each domain controller can be one or a collection of existing nodes inside domains that are equipped with the controlling functionality (i.e., in-band control [21]) or external controlling entities operating on top of a network domain. (i.e., out-of-band control [1]). All domain controllers together with inter/intra-domain controlling channels form the control plane. For our analytical

studies in Chapter 3 and Chapter 4, it is assumed that under the distributed SDN architecture, to construct a path between a pair of source and destination nodes for the given networking task, the corresponding routing path is logically determined by the domain controllers in the source, the destination, and all intermediate domains collectively, using the synchronised information among them. However, the performance of the constructed paths may vary, depending on the network status information at each involved controller. Next, we discuss the performance metric employed for our analytical studies.

2.3 Link Preference and Path Cost for Routing in SDN

In the distributed SDN architecture, a routing path construction between a pair of nodes is determined by all involved controllers. To reach an optimised routing decision, controllers take into account the traffic status, load balancing, and other policy-related factors. To this end, controllers can proactively assign a weight to each link to indicate the link preference based on the collected network information, i.e., the smaller the link weight, the better the link is for path construction, so that the end-to-end accumulated weight of any path matches its corresponding path construction preference. Therefore, the goal for constructing an optimised end-to-end inter-domain path under a given network status is reduced to finding the end-to-end path with the minimum accumulated weight under the given link weight assignment. We refer to such accumulated path weight as the *path cost*. In real networks, the performance of routing can be measured by many metrics, such as delay and congestion level, depending on the goal of network management. In order to make our analytical work sufficiently generalised to capture the performance metric that is important to most network management tasks, we employ the average path cost (APC), measured by the average accumulated link weight of the constructed path, as the performance

metric for our analyses and evaluations in Chapter 3 and Chapter 4, and Chapter 5.

Remark: Under distributed SDN, link preference assignment is adjusted dynamically by the domain controller according to the current network status and the routing performance metric used. For example, when routing performance is delay (additive metric), the domain controller simply assigns as link weights the delays of intra-domain links under the given traffic levels. In another example, if the routing objective is to find the least congested path (non-additive metric), then the weight assignments should reflect the preference for links with low load levels. We assume that controllers assign such link preferences according to their control objectives; the exact mechanism of link preference assignment subject to different routing objectives is beyond the scope of this thesis, and thus not discussed.

Since the link preference (weight) can be dynamic, in this thesis, we conduct our analysis in two types of networks which we call *network with uniform link preference (Type-1 Network)* and *network with non-uniform link preference (Type-2 Network)*. For Type-1 Networks, all link preferences are static and equal; therefore, without loss of generality, all link weights in Type-1 Networks are set to 1. By contrast, in Type-2 Networks, random variables are used to capture the dynamicity of link preference. Specifically, for Type-2 Networks, we assume that intra-domain link preferences across all domains are at least 1 and are independent and identically distributed (i.i.d.). Furthermore, in real distributed SDN environments, unlike the intra-domain links which are potentially wireless, inter-domain gateway-to-gateway links are likely to be wired with high bandwidth, thus more stable. In this regard, we characterise all inter-domain link weights by a non-negative constant. Without loss of generality, we assume that the link preference levels for all inter-domain links are 1; all our theoretical results can be easily extended to other policy-based inter-domain setups, if the behaviours of such policy-based setups can be captured by random variables with certain distributions.

2.4 Capability Requirements for Distributed SDN

In this section, we briefly discuss the practical SDN capability requirements in order to meet the assumptions and problem scenarios in later chapters of the thesis.

For the distributed SDN paradigm, there are generally two types of logical communication interfaces, i.e., east-west interface and north-south interface. The former enables the communication among SDN controllers, whereas the latter abstracts the interactions between SDN controller in the control plane and networking elements in the data plane.

In this thesis, the east-west interface is used for controller synchronisation and for coordinating inter-domain networking tasks. For example, the controller synchronisation process in Chapter 3 and Chapter 4, as defined in Definition 1, involves the exchange of minimum path costs between intra-domain node pairs by the corresponding SDN controllers. In addition, the implementation of inter-domain routing mechanism, such as the RCPC defined in Section 3.4, requires similar capability to enable the first domain controller within an routing cluster (RC) to communicate the path construction decision to all other controllers within the RC. In practice, this type of inter-controller communication is realised through the east-west interfaces of distributed SDN's control plane. Due to the high programmability and agility of SDN, this can be easily implemented as software-based communication interfaces on controllers. For instance, one protocol named Communication Interface for Distributed Control plane (CIDC), which is proposed in [22], can handle all east-west communication discussed in this thesis.

On the other hand, the north-south interface is mainly used for implementing controller's control policies (southbound) and gathering status information from networking elements in the data plane (northbound). Specifically, in the context of inter-domain routing tasks discussed in Chapter 3-5, controller installs routing rules in SDN switches for directing packets routing. For the service path construction tasks

discussed in Chapter 6 and Chapter 7, controller installs service request forwarding rules in SDN switches in a similar manner. In addition, the RL-based network control tasks (Chapter 5-Chapter 7) also use SDN's northbound interface for collecting status information of networking elements in the data plane to assist the controller in learning network control policies. Currently, there is no unified standard on the implementation of SDN's northbound interface [23], various controller architectures provide their own high-level Application Programmable Interfaces (APIs). For example, the intent-based [23] northbound API of the ONOS [24] controller supports a wide range of communication modes between data plane elements and controller, which enables the status data collection process in Chapter 5-Chapter 7.

CHAPTER 3

Analysis of Performance Enhancements by Controller Synchronisation - Approach I

3.1 Introduction

This chapter presents the first family of analytical methods we develop (Approach I), which renders fine-grained analytical results revealing the performance of inter-domain routing tasks in SDN and its relationships to controller synchronisation levels and other network structural parameters. Based on the network model presented in Section 2.2, we derive analytical expressions of the network performance focusing on the APC of the constructed paths w.r.t. random flow requests. In particular, we first establish an asymptotic expression to highlight the relationship between the performance metric and dominant parameters. Then, such a performance metric is investigated under four canonical synchronisation levels, ranging from the minimum to the maximum level of synchronisation (see Section 3.3.1), i.e., between Minimum Synchronisation (MS) and *Complete Synchronisation (CS)*. If a given synchronisation scenario cannot be described by any of these four cases, then its performance can be bounded by our analytical results corresponding to the two extreme cases (i.e., maximum/minimum synchronisation).

Analytical results reveal the relative contributions of different parameters to the

performance metric. For example, the performance metric scales linearly with the average domain-wise distance; whereas it scales logarithmically with the number of nodes in each domain. Moreover, the performance gain declines with the increasing synchronisation level and the number of gateways. To validate the accuracy of the derived analytical expressions, they are compared against evaluation results using both real and synthetic networks. Note that the proofs for lemmas, theorems, and corollaries in Chapter 3 are documented in Appendix A.1.

3.2 Related Work

3.2.1 Information Sharing for Routing Quality Improvement

Researchers have looked into better understanding the performance of hierarchical routing where the internal structure of each domain is not revealed to outside nodes. For example, [25] shows that hierarchical routing where the topologies of the clusters are hidden can lead to suboptimal routing, and forwarding loops; [26] proposes solutions to aggregate topologies with theoretical bounds. Moreover, [27] analyses the effectiveness of hierarchical routing (e.g., ATM PNNI [28], Nimrod [29]). However, most of these early works are either driven by simulations or with different focuses from ours. In contrast, we conduct a rigorous mathematical analysis to understand the benefits of controller synchronisation in distributed SDN. It should be noted that although some of the theoretical results presented in this thesis could be applied to the analysis of legacy networks under certain conditions, our work is SDN-focused because many of the assumptions we have for modelling can only be realised through fine-grained control under SDN. For example, SDN's state update process enables fine-grained domain information exchange, which is crucial in our definition of controller synchronisation. On the other hand, SDN also makes it possible for joint-decision making and implementation of routing policies, which is in

the core of our analysis.

3.2.2 Distributed SDN

The distributed SDN architecture, which integrates the advantages in early hierarchical networks, have stimulated many research efforts in this area. Specifically, the feasibility of deploying SDN-based mechanisms incrementally to the current BGP-glued Internet is considered in [5], where routing control planes of multiple domains are outsourced to form centralised control planes for optimizing routing decisions. Similarly, [30] investigates the problem of SDN upgrade in ISP (Internet Service Provider) networks under the constraint of migration costs. In addition, protocols and systems, such as HyperFlow [31], DISCO [32], and ONOS [24], are proposed to realise logically centralised but physically distributed SDN architecture. Devoflow [33] and Kandoo [34] are designed to reduce the overheads introduced by the interaction between the control and data planes; while DIFANE [35] and Fibbing [36] are conceived for limiting the level of centralisation and addressing robustness issues, respectively.

Moreover, Google's B4 [2] and Espresso [37], and Facebook's Edge Fabric [38] are the examples of tackling real-world network control problems, such as data centre management, traffic engineering, using SDN-based techniques. However, again, most of these works are experiment-based. By contrast, our goal is to investigate distributed SDN from the perspective of fundamental analytics.

3.2.3 Performance Analytics

Since all theoretical results in this thesis are obtained based on a graph model, our work is related to the area of graphical analysis of complex networks. Most works in this area are dedicated to the study of specific graph properties, e.g., small-world effect [39], network motif [40], scale-free [41], etc. On the other hand, models

in [42, 39, 43, 44] are purely randomised, which cannot differentiate intra-/inter-domain links in the context of distributed SDN. Thus, they are substantially different from our work. Our approach is also comparable to [45] as they also consider a layered-network model for the study of communication networks. However, the authors are mainly concerned with modelling the co-existing connectivity.

3.3 Problem Formulation

3.3.1 Synchronisation Among SDN Controllers

Recall that we associate link preference level (weight) (see Section 2.3) to links in the network model developed (see Section 2.2). Since for inter-domain routing path construction tasks, link preference (weight) captures the controller's view of the current domain, i.e., network status information, the process of controller synchronisation involves the exchange of such information, which we formally define below.

Definition 1. Domain $\mathcal{A}_i$ is *synchronised* with domain $\mathcal{A}_j$ if and only if the SDN controller in $\mathcal{A}_i$ knows the minimum path cost between any two nodes in $\mathcal{A}_j$.

By Definition 1, clearly there exist a significant number of synchronisation cases. Moreover, in real networks, it is usually the case that synchronisation difficulty is high when two SDN controllers are far apart. In this thesis, we therefore categorise inter-domain synchronisations into the following cases, sorted by their corresponding synchronisation difficulties.

*a) Minimum Synchronisation **(MS)***: Under MS, no domains synchronise with any other domains. As a result, each controller only knows its own intra-domain topology and the domain-wise topology, but the controller does not assign link preference levels (all links have an equal link preference of 1) due to the lack of network status information. This scenario captures IGP routing protocols that do not take into account any link weights but select routes purely based on the hop count (e.g., Routing

Information Protocol (RIPv2)). Note that MS corresponds to the minimum network knowledge that is always available, including scenarios in *b)–d)* ;

b) Self-domain Synchronisation ***(SS)***: In addition to the information under MS, each controller under SS knows its intra-domain and out-going inter-domain link preference levels. With this additional information, one controller can find the optimal intra-domain path for any intra-domain flow requests, within its own domain;

c) Partial Synchronisation ***(PS)***: PS refers to any synchronisation levels between SS and the following complete synchronisation (CS), where some controllers exchange the views of their own domains gained through SS. See an example of PS in Fig. 3.1 where there are five domains, among which domain pairs $\{\mathcal{A}_1, \mathcal{A}_2\}$, and $\{\mathcal{A}_3, \mathcal{A}_4\}$ are respectively synchronised; $\mathcal{A}_5$ is not synchronised with the other four domains. Under PS, SDN and legacy routing policies could coexist, e.g., those synchronised domains may operate on SDN routing utilising the synchronised information (see Section 3.4 for details), whereas those not synchronised operate on a fully distributed inter-domain routing protocol such as BGP. PS is the most realistic scenario in distributed SDN, as it balances the benefits and costs of controller synchronisation;

d) Complete Synchronisation ***(CS)***: Under CS, every pair of domains $\mathcal{A}_i$ and $\mathcal{A}_j$ synchronise with each other. As such, there is effectively one logically centralised controller, which can make globally optimal decisions. Among all these synchronisation scenarios, CS experiences the highest synchronisation difficulty.

3.3.2 Problem Statement and Objective

Given the distributed SDN network model in Chapter 2, our goal is to study the performance of the paths constructed by a basic and representative path construction mechanism (see Section 3.4 for details) under various synchronisation scenarios. To this end, we use the APC defined in Section 2.3 as the performance metric. Here

APC is a natural generalised performance metric, as link weights are dynamically adjusted by controllers based on the current network status to reflect time-varying link preference. Formally, our research objective is:

Objective: Suppose (i) each network realisation under the two-layer network model exists with the same probability, and (ii) the source-destination node pair belonging to two different domains in a given network realisation also exists with the same probability. Our goal is to derive mathematical expressions of APC under each of the four synchronisation scenarios, i.e., MS, SS, PS, and CS, in both Type-1/Type-2 Networks (networks with uniform/non-uniform link preference).

Remark: In this thesis, we are only interested in studying the cross-domain routing, since controllers can easily find the optimal intra-domain paths without relying on inter-controller synchronisations. Note that our two-layer network model is a random graph model, i.e., there exist multiple network realisations satisfying the same set of input parameters. Therefore, APC is an expected value over not only random source/destination node pairs but also random network realisations. All our theoretical results on APC are based on the given network parameters (e.g., degree and weight distributions) rather than a specific network realisation.

3.4 Path Construction Mechanism

We describe a path construction mechanism for 4 synchronisation scenarios introduced in Section 3.3.1. The intuition behind the path construction mechanism is that given a particular synchronisation level, the synchronised controllers attempt to use the synchronised information and make joint decision to minimise the overall accumulated cost of the constructed path in their domains. Then the selected path segments in all participating domains between the source/destination nodes concatenate into a cross-domain, end-to-end path. Before presenting the path construction

mechanism, we first introduce several definitions as follows.

Definition 2. *a)* In the domain-wise topology $\mathcal{G}_d$, the vertex corresponding to domain $\mathcal{A}$ in $\mathcal{G}$ is denoted by $\vartheta(\mathcal{A})$. Given a pair of source and destination nodes v_1 and v_2 with[1] $v_1 \in \mathcal{A}_1$, $v_2 \in \mathcal{A}_2$, and $\mathcal{A}_1 \neq \mathcal{A}_2$, the *domain-wise path* w.r.t. v_1 and v_2 is a path in $\mathcal{G}_d$ starting at vertex $\vartheta(\mathcal{A}_1)$ and terminating at vertex $\vartheta(\mathcal{A}_2)$;

b) The *domain-wise distance* w.r.t. domains $\mathcal{A}_1$ and $\mathcal{A}_2$ is the length of the shortest path from the vertex corresponding to $\mathcal{A}_1$ to the vertex corresponding to $\mathcal{A}_2$ in the domain-wise topology $\mathcal{G}_d$.

Based on Definition 2, we then define synchronisation radius to capture different levels of synchronisations as follows.

Definition 3. The *synchronisation radius* τ ($\tau \geq 1$) is an *integer* such that (i) *any* two domains with their domain-wise distance less than or equal to $\tau - 1$ are synchronised, and (ii) *no* two domains with their domain-wise distance greater than $\tau - 1$ are synchronised.

According to the definition of synchronisation radius, $\tau = 1$ for MS or SS, depending on link preference status; $\tau = \phi$ for CS, where ϕ is the maximum domain-wise distance between any two domains in the network. Any value of τ between 1 and ϕ falls in the category of PS. As such, we use a given τ to capture the PS scenario. Under a specified synchronisation level, the synchronised controllers leverage the shared information to jointly make routing decisions on any domain-wise paths between source/destination nodes. Formally, we have the following definition.

Definition 4. The group of domain(s) on the domain-wise path where routing decisions are jointly made by their synchronised controller(s) is referred to as a *routing cluster (RC)*. Specifically, given a domain-wise path between the source and destination nodes, for all domains on this domain-wise path:

[1] In this thesis, for graph $\mathcal{G} = (V, E)$, by abusing graph theory notations, we use vertex $v \in \mathcal{G}$ to denote $v \in V$ and edge $e \in \mathcal{G}$ to denote $e \in E$.

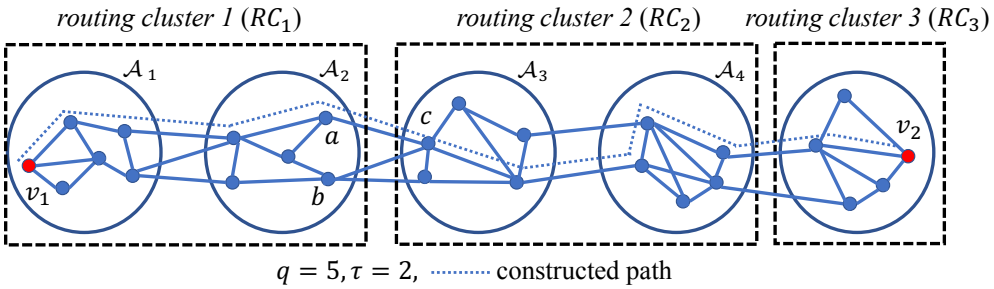


Figure 3.1: Path construction w.r.t. v_1 and v_2, whose shortest domain-wise path traverses $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$, $\mathcal{A}_4$, and $\mathcal{A}_5$.

a) Under MS or SS ($\tau = 1$), each domain constitutes an RC;

b) Under PS ($1 < \tau < \phi$), starting from the source domain, every τ domains form an RC such that each domain belongs to one and only one RC, and only the RC including the destination domain may have less than τ domains;

c) Under CS ($\tau = \phi$), all domains on the domain-wise path form an RC, where ϕ is the maximum domain-wise distance between any two domains in the network.

According to Definition 3 and 4, for any domain pairs inside an RC, there must be at least one domain-wise path connecting them s.t. all intermediate domains on the domain-wise path between them are within the same RC; otherwise, jointly optimal routing decisions cannot be guaranteed between any two nodes within the RC, due to the presence of external domain(s) en route, whose information is not known to RC domains. Based on Definition 4, let q and μ denote the number of domains and the number of RCs on the domain-wise path, respectively. For PS with synchronisation radius τ, the RC that includes the destination domain has $q - \tau(\mu - 1)$ domains, whereas all other RCs have τ domains. Now, we are ready to introduce the path construction mechanism between two arbitrary nodes v_1 and v_2 in the following steps:

Step 1) Select the shortest domain-wise path w.r.t. v_1 and v_2, which consists of q domains, with ties (if any) broken arbitrarily. That is, no domain-wise path w.r.t. v_1 and v_2 traverses less than q domains.

Step 2) Based on the given synchronisation status of all involved domains on the above domain-wise path, partition these domains into μ RCs ($\mu = q$ for MS and SS, $\mu = 1$ for CS, and $\mu = \lceil q/\tau \rceil$ for PS);

Step 3) For each RC_i (RCs are sequentially labelled from the source to the destination, $i = 1, 2, \ldots, \mu$), a path segment starting from the entering node (which is v_1 if $i = 1$, or is specified by RC_{i-1}) and terminating at one of the exiting nodes (which are gateways connecting to RC_{i+1}, or node v_2 if $i = \mu$) with the minimum cost is constructed.[2] Such path segment is denoted by $\mathcal{P}_i$ in RC_i. Also let $e_{i,i+1}$ be the edge leading from $\mathcal{P}_i$ in RC_i to connect to the entering node in RC_{i+1} if $i \leq \mu - 1$;

Step 4) The final v_1-to-v_2 path $\mathcal{P}$ is

$$\mathcal{P} = \mathcal{P}_1 + e_{1,2} + \mathcal{P}_2 + e_{2,3} + \ldots + \mathcal{P}_{\mu-1} + e_{\mu-1,\mu} + \mathcal{P}_\mu. \tag{3.1}$$

Remark: ***Step 1)*** is similar to the BGP protocol used for inter-domain routing in the Internet.[3]. We further justify the selection of the shortest domain-wise path in Theorem 3.6 and Corollary 3.7. The path construction mechanism described above relies on routing clusters as the basic routing unit, it is therefore referred to as *routing cluster-based path construction (RCPC)* in the sequel. Fig. 3.1 shows a PS example with $q = 5$ and $\tau = 2$ under RCPC. After the selection of a domain-wise path which consists of domains $\mathcal{A}_1 - \mathcal{A}_5$, the domains are partitioned into 3 RCs according to ***Step 2)***, as shown in the figure. Then, by ***Step 3)***, routing decision is made jointly by controllers in each RC to minimise the corresponding path cost. For example, assume that all link preferences are 1 in Fig. 3.1, the controllers of $\mathcal{A}_1$ and $\mathcal{A}_2$ jointly choose node a as the exit point and thus construct a path segment between v_1 and a in RC_1. The core of RCPC is that the synchronised SDN controllers jointly decide the

[2]Note that ECMP or similar schemes could be applied within RCs, since equal intra-RC costs would be incurred.

[3]For mathematical tractability, we do not consider some uncontrollable random factors, such as the LOCAL PREFERENCE attribute in BGP.

routing policies according to the link preferences in their domains, i.e., to minimise accumulated end-to-end link weights. For mathematical tractability, other routing-related factors, such as the LOCAL PREFERENCE in BGP, are reflected in the controller-assigned link weights.

Note that the intention in this thesis is not to design a new routing mechanism; instead, the goal is to use a basic routing mechanism, RCPC, to understand the network performance in distributed SDN. For improved routing mechanisms, our RCPC-based analytical results serve as performance bounds.

3.5 Asymptotic APC for Various Synchronisation Levels

Before the discussion of fine-grained analytical results on APC, we first present the asymptotic analysis of APC (called *asymptotic APC*) under various synchronisation scenarios in this section. The basic idea here is that we highlight, in the form of directly observable expressions, the interactions among different parameters in determining the overall APC.

The basic intuition behind the derivation of the asymptotic APC is that we first compute the average domain-wise distance w.r.t. two arbitrary source/destination nodes. Then, with the given synchronisation level (τ), the domains on the domain-wise path form RCs according to RCPC. Finally, we calculate the APC inside individual RCs and add up these APCs to obtain the accumulated end-to-end APC. When the number of domains inside an RC is more than one, we employ a special graph, called the *randomised Degree-Preserving Network (RDPN)*, to help us derive its APC. In essence, RDPN is obtained by aggregating the topologies of all domains inside an RC to a single graph, for which the aim is to make the derivation of APC tractable (see Definition 6 for details).

Table 3.1: Main Notations and Abbreviation

Notation	Meaning
m	number of domains in the network
n	number of nodes in each domain
β	inter-domain connection parameter
γ	$\gamma = n(1-(1-1/n)^{\beta})$, average number of gateways in a domain connecting to a neighbouring domain
z_1, z_2	average number of nodes that are 1-/2-hop away from a randomly chosen node within a domain
z_1', z_2'	average number of domains that are 1-/2-hop away from a randomly chosen domain in the domain-wise topology
τ	synchronisation radius
ζ_i	average number of vertices which are i-hop away from a random vertex in a RDPN (Definition 6)
Δ	$\Delta = \frac{\log(m/z_1')}{\log(z_2'/z_1')} + 1$ is the average domain-wise distance w.r.t. two arbitrary domains (Section 3.6)
MS	minimum synchronisation
SS	self-domain synchronisation
PS	partial synchronisation
CS	complete synchronisation

Let m and n be the number of domains and the number of nodes in each domain in the network, and γ the average number of gateways connecting two neighbouring domains ($\gamma = n(1-(1-1/n)^{\beta})$). Next, within a domain $\mathcal{A}$, let z_i denote the average number of vertices that are i-hop away from a random vertex within $\mathcal{A}$. Similarly, in the top-layer $\mathcal{G}_d$ of our two-layer model, let z_i' denote the average number of vertices (here each vertex represents a domain) that are i-hop away from a random vertex in $\mathcal{G}_d$ (Refer to Section 3.6 for details of z_1', z_2', and γ). In addition, let ζ_i be the average number of vertices which are i-hop away from a random vertex in an RDPN. Main notations and abbreviations used in this thesis are summarised in Table 3.1. Under all above definitions and path construction mechanisms, we present the asymptotic APC in the following theorem.

Theorem 3.1. Given the synchronisation radius τ, the asymptotic APC (denoted by $\mathcal{L}$) in the two-layer network model is

$$\mathcal{L} = \begin{cases} O\Big(\frac{(\Delta-1)\log(\frac{n\tau'}{\zeta_1\gamma})}{\tau\log(\zeta_2/\zeta_1)} + \frac{\log(n\tau'/\zeta_1)}{\log(\zeta_2/\zeta_1)}\Big) & \text{if } \gamma \leq \frac{n\tau'+1}{\zeta_1+1}, \\ O\Big(\frac{\Delta-1}{\tau} + \frac{\log(n\tau'/\zeta_1)}{\log(\zeta_2/\zeta_1)}\Big) & \text{otherwise,} \end{cases} \tag{3.2}$$

where $\tau' = \min\{\tau, \Delta+1\}$; see Table 3.1 for other notations.

Theorem 3.1 directly shows how the synchronisation level (τ) affects the APC. Specifically, when τ is small, there are two dominant terms, which are both logarithmic functions, in (3.2). However, with the increase of τ, when the network achieves CS, only the second logarithmic function is dominant, and the two cases under different values of γ in (3.2) are merged into one unified expression, i.e., $\mathcal{L} = O\Big(\frac{\log(n\tau'/\zeta_1)}{\log(\zeta_2/\zeta_1)}\Big)$, with $\tau' \approx \Delta+1$. To better observe these trends, we consider a sample two-layer network with the *Erdös-Rényi (ER)* model (see Section 3.14.1.2) as the graph model in each layer with the following parameters: $m = 200$, $n = 500$, $p = 2/199$ for the domain-wise topology, and $p = 3/499$ for the intra-domain topology (see Section 3.14.1.2 for parameter p) and visualise the corresponding expression of (3.2) in Fig. 3.2. Clearly, $\mathcal{L}$ steadily descends with a diminishing amount every time τ increases by 1, thus implying the declining benefit of the increased synchronisation level. In addition, we also observe that having more gateways (larger β) results in a smaller $\mathcal{L}$. However, the performance gain of larger β also gradually diminishes as the synchronisation level grows. Thus, there is a cost/benefit trade-off that needs to be considered in practical network design. The asymptotic APC's ability to reveal the relationship between APC and other parameters are validated in Section 3.14.

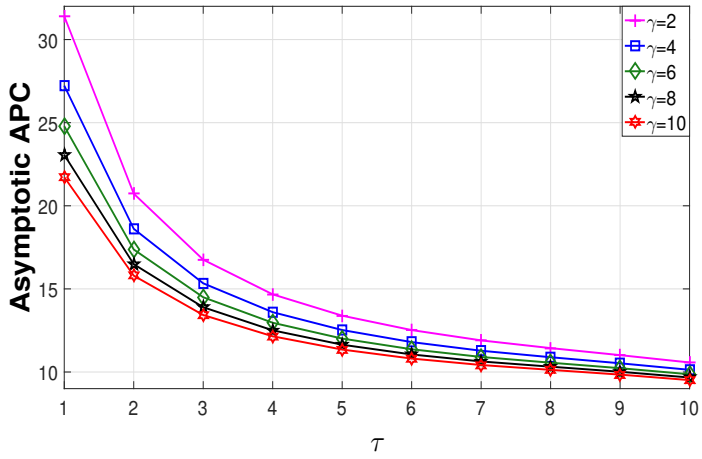


Figure 3.2: $\mathcal{L}$ in (3.2) in a sample network with varying τ and γ.

3.6 APC under MS in Type-1 Networks

In this section, we study the APC under MS in Type-1 Networks (all links are of equal preference, i.e., link preference levels are 1 for all links) based on the path constructions mechanism RCPC introduced in Section 3.4.

To this end, we first present the results in the existing work [44] to assist our mathematical analysis.

Proposition 3.2. [44] In an undirected connected graph $\mathcal{H}$ with n_0 vertices and the vertex degree satisfying a given distribution, let x_i be the average number of vertices that are i-hop away from a random vertex in $\mathcal{H}$. Suppose all edge weights are 1, and $x_2 \gg x_1$ [4]. Then *a)*

$$x_i = (x_2/x_1)^{i-1} x_1; \tag{3.3}$$

b) APC in $\mathcal{H}$ is

$$\frac{\log(n_0/x_1)}{\log(x_2/x_1)} + 1. \tag{3.4}$$

In our two-layer model, the top-layer graph $\mathcal{G}_d$ (domain-wise topology with m vertices) itself is a random graph following a given domain-wise degree distribution.

[4]This is a valid assumption because, according to our observations of real network datasets, the number of two-hops nodes is (exponentially) larger than the number of immediate neighbour nodes in most cases, i.e., $x_2 > x_1^2$, in both intra-domain and domain-wise topologies.

Therefore, similar to [44], let z_i' denote the average number of vertices that are i-hop away from a random vertex in $\mathcal{G}_d$. For two arbitrary nodes v_1 and v_2 with $v_1 \in \mathcal{A}_1$, $v_2 \in \mathcal{A}_q$, and $\mathcal{A}_1 \neq \mathcal{A}_q$, let Δ denote the average distance of the shortest domain-wise path from domain $\mathcal{A}_1$ to domain $\mathcal{A}_q$. Assuming $z_2' \gg z_1'$, then according to (A.7), we have

$$\Delta = \frac{\log(m/z_1')}{\log(z_2'/z_1')} + 1. \tag{3.5}$$

With (3.5), we know that the average number of domains for MS under RCPC is $\Delta + 1$. If we further know the average cost of $\mathcal{P}_i$ associated with the traversed domain $\mathcal{A}_i$, then we can estimate the average cost of $\mathcal{P}$. To this end, let $|\mathcal{P}|$ denote the *number of hops* on path $\mathcal{P}$. Then, $|\mathcal{P}| = |\mathcal{P}_1| + |\mathcal{P}_2| + \ldots + |\mathcal{P}_{\Delta+1}| + \Delta$ according to (3.1), where $|\mathcal{P}_i|$ is a r.v. The expectation of $|\mathcal{P}|$ is:

$$\begin{aligned} \mathbb{E}[|\mathcal{P}|] &= \mathbb{E}[|\mathcal{P}_1| + |\mathcal{P}_2| + \ldots + |\mathcal{P}_{\Delta+1}|] + \Delta \\ &= \mathbb{E}[|\mathcal{P}_1|] + \mathbb{E}[|\mathcal{P}_2|] + \ldots + \mathbb{E}[|\mathcal{P}_{\Delta+1}|] + \Delta. \end{aligned} \tag{3.6}$$

According to the path construction procedure for MS, $\mathbb{E}[|\mathcal{P}_1|] = \mathbb{E}[|\mathcal{P}_2|] = \ldots = \mathbb{E}[|\mathcal{P}_\Delta|]$ for two reasons. First, all domains have the same statistical properties. Second, in each domain $\mathcal{A}_i$ ($i \leq \Delta$), the routing mechanism selects a gateway (from a set of gateway options) that is closest to the ingress node. By contrast, in domain $\mathcal{A}_{\Delta+1}$, the routing mechanism only selects the minimum-cost path from the ingress node to the destination node v_2. Thus, (3.6) is simplified as

$$\mathbb{E}[|\mathcal{P}|] = \Delta \cdot \mathbb{E}[|\mathcal{P}_1|] + \mathbb{E}[|\mathcal{P}_{\Delta+1}|] + \Delta. \tag{3.7}$$

In a domain $\mathcal{A}$ with n nodes, let z_i denote the average number of intra-domain nodes

that are i-hop away from an arbitrary node v ($v \in \mathcal{A}$). Then again by (A.7), we have

$$\mathbb{E}[|\mathcal{P}_{\Delta+1}|] = \frac{\log(n/z_1)}{\log(z_2/z_1)} + 1, \tag{3.8}$$

assuming $z_2 \gg z_1$. Hence, to compute $\mathbb{E}[|\mathcal{P}|]$ in (3.7), it suffices to consider only $\mathbb{E}[|\mathcal{P}_1|]$ associated with domain $\mathcal{A}_1$.

In $\mathcal{A}_1$, on average, there are $\gamma = n(1-(1-1/n)^\beta)$ gateways connecting to $\mathcal{A}_2$. Suppose $\mathcal{A}_1$ contains exactly γ gateways, denoted by set S. Then regarding path $\mathcal{P}_1$ from the starting point v_1 in $\mathcal{A}_1$ to set S, there are two cases. First, $v_1 \in S$, then $\mathcal{P}_1$ is a degenerate path containing only one node v_1, i.e., $|\mathcal{P}_1| = 0$. Second, $v_1 \notin S$, which complicates the computation of $|\mathcal{P}_1|$. For the second case, let $l := \mathbb{E}[|\mathcal{P}_1| \; | v_1 \notin S]$, i.e., the expectation of $|\mathcal{P}_1|$ conditioned on $v_1 \notin S$. Regarding the gateway set S, there are up to γz_i non-gateways that are i-hop away from the closest gateways. Let $l_{\max} := \arg\max_i z_i$ s.t. $\gamma + \sum_{j \le i} z_j \le n$. According to (A.6), z_i increases exponentially with i. In other words, the majority of non-gateways are $l_{\max}$-hop away from the closest gateways; therefore, we use $l_{\max}$ to approximate l. Thus, $z_l \approx z_{l_{\max}} \approx n - \gamma \approx n + 1 - \gamma$ when n is large. By solving $z_l = n + 1 - \gamma$, we obtain

$$l = \frac{\log(\frac{n+1-\gamma}{z_1\gamma})}{\log(z_2/z_1)} + 1, \tag{3.9}$$

where $\gamma = n(1-(1-1/n)^\beta)$. By close examination of (3.9), we notice that it is also needed to guarantee $l \ge 1$. Hence, (3.9) can be calibrated as follows.

$$l = \begin{cases} \frac{\log(\frac{n+1-\gamma}{z_1\gamma})}{\log(z_2/z_1)} + 1 & \text{if } \gamma \le \frac{n+1}{z_1+1}, \\ 1 & \text{otherwise.} \end{cases} \tag{3.10}$$

We can verify that when $\gamma = 1$, (3.10) reduces to (3.8) as expected. A key threshold $\gamma_0 = (n+1)/(z_1+1)$ is revealed in (3.10). When $\gamma \le \gamma_0$, the distance from an arbitrary non-gateway to the closest gateway is relatively large; when $\gamma > \gamma_0$, there

are sufficiently many gateways randomly distributed in one domain, causing each non-gateway to have a gateway neighbour with high probability. Hence,

$$\begin{aligned}\mathbb{E}[|\mathcal{P}_1|] &= \mathbb{E}[|\mathcal{P}_1| \; |v_1 \notin S]\text{Pr}(v_1 \notin S) \\ &\quad + \mathbb{E}[|\mathcal{P}_1| \; |v_1 \in S]\text{Pr}(v_1 \in S) + 1 = (\frac{n-\gamma}{n})l + 1,\end{aligned} \tag{3.11}$$

where $\frac{n-\gamma}{n}$ is the percentage of non-gateway nodes in a domain. Putting (3.5), (3.8), and (3.11) into (3.7), the final expression of APC under MS is summarised in the following theorem.

Theorem 3.3. The APC in Type-1 Networks under MS (denoted by $L_{\text{MS}}^{\text{Type-1}}$) is

$$L_{\text{MS}}^{\text{Type-1}} = \begin{cases} \Delta\Big(\big(\frac{n-\gamma}{n}\big)\big(\frac{\log(\frac{n+1-\gamma}{z_1\gamma})}{\log(z_2/z_1)} + 1\big) + 2\Big) & \\ + \frac{\log(n/z_1)}{\log(z_2/z_1)} + 1 & \text{if } \gamma \leq \frac{n+1}{z_1+1}, \\ \Delta(\frac{n-\gamma}{n} + 2) + \frac{\log(n/z_1)}{\log(z_2/z_1)} + 1 & \text{otherwise,} \end{cases} \tag{3.12}$$

see Table 3.1 for notations.

It can be observed that the domain-wise distance (Δ) and the number of gateways (γ) in domains are the most influential factors in shaping the APC for MS. Specifically, $L_{\text{MS}}^{\text{Type-1}}$ is logarithmic in domain structural parameters n, γ, z_1, and z_2, and it is near linear in Δ.

Since SS coincides with MS in Type-1 Networks, we therefore discuss synchronisation scenario PS in the next section.

3.7 APC under PS in Type-1 Networks

In this section, we consider the partial synchronisation (PS) model[5] as defined in Definition 3. RCs are created as basic routing units according to Definition 4 under PS. Note that the network graph of an RC is no longer a random graph, because multiple domains are connected via inter-domain connections in a specific way as dictated by the network model. As such, we cannot directly apply the results obtained in Section 3.6 for the APC expression under MS in Type-1 Networks. Regarding such difficulties, in this section, we instead derive the APC lower bound for PS with the assistance of an auxiliary network called the *randomised Degree-Preserving Network (RDPN)* (see Definition 5). Here is the sketch of our methodology.

Sketch of Analytical Methodology:

a) Given a domain-wise path, we identify all RCs along the path according to Definition 4;

b) We construct the RDPN associated with each RC;

c) We compute the path cost incurred in RDPNs, and prove it is a lower bound of the actual path cost incurred in its original RC;

d) Adding up RDPN path costs and the number of inter-RC connections, we get the lower bound of APC under PS.

Based on this methodology, we next describe the details on how the APC lower bound under PS is derived.

[5]Such PS model enables an efficient analytical method for understanding the routing performance under different partial synchronisation levels (quantified by the synchronisation radius). Other PS models are left for future work.

3.7.1 The Line Network and its Randomised Degree-Preserving Network (RDPN)

We first formally define the following terms: (i) the *line network* that generalises RCs; and (ii) the randomised Degree-Preserving Network (RDPN) of a line network. These concepts are also used in the analysis of Type-2 Networks.

Definition 5. A *line network* with k domains is a special graph generated via the two-layer network model, consists of k domains, where its domain-wise topology is a connected linear graph (i.e., a connected tree where no vertex has degree 3 or more). The domains with inter-domain degree being 1 and 2 in a line network are called *end-domains* and *transit-domains*, respectively.

Definition 6. For a line network (denoted by $\mathcal{F}$) with k domains and n nodes in each domain, the corresponding *randomised Degree-Preserving Network (RDPN)* of $\mathcal{F}$, denoted by $\mathcal{F}_R$, is a randomly generated network with kn nodes such that $\mathcal{F}_R$ and $\mathcal{F}$ have the same degree distribution.

Remark: Although $\mathcal{F}$ and $\mathcal{F}_R$ have the same degree distribution and the number of nodes, they differ significantly from the perspective of randomness. In particular, $\mathcal{F}$, as a line network, is constrained to certain structural properties, i.e., the domain-wise topology must be a linear graph with k vertices. The RDPN $\mathcal{F}_R$, however, is purely random without such constrains. Thus, let $S_{\mathcal{F}}$ and $S_{\mathcal{F}_R}$ be the sets of all graph instances of $\mathcal{F}$ and $\mathcal{F}_R$, respectively. Then, $S_{\mathcal{F}} \subseteq S_{\mathcal{F}_R}$.

3.7.2 Path Cost in RDPN

With the concept of RDPN, we now show the relationships between path costs in the line network and its corresponding RDPN. Specifically, we discuss the minimum path cost between a randomly chosen vertex and a vertex set in a line network and its corresponding RDPN. To this end, we first derive the following theorem.

Theorem 3.4. For a line network (denoted by $\mathcal{F}$) consisting of k domains sequentially labelled as $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_k$, let $\mathcal{F}_R$ denote the RDPN of $\mathcal{F}$. Let ρ be the average path cost of the minimum-cost path between a random node μ ($\mu \in \mathcal{A}_1$) and a random node set M ($\mu \notin M, M \subseteq \mathcal{A}_k$), and ρ_R the average path cost of the minimum-cost path between a random node μ_R ($\mu_R \in \mathcal{F}_R$) and a random node set M_R ($\mu_R \notin M_R, M_R \subseteq \mathcal{F}_R$) such that $|M| = |M_R|$. Then, $\rho_R \leq \rho$ holds.

With Theorem 3.4, the APC lower bound under PS can be obtained by combining the path costs of RDPNs of all associated RCs. Therefore, we only need to focus on the computation of path cost in each RDPN. Viewing each RDPN of RCs as a random graph following a certain degree distribution, we reapply the results in Section 3.6. Specifically, the first step of path cost calculation in a random network is to determine the number of 1-hop and 2-hop vertices from a randomly selected vertex. As such, we present the following lemma.

Lemma 3.5. In the RDPN of a line network $\mathcal{F}$ with k domains and the inter-domain connection parameter β, let ζ_1 and ζ_2 denote the number of vertices that are 1-hop and 2-hop away from a random vertex, respectively. Then, the following holds:$\zeta_1 \approx z_1 + \frac{2\beta(k-1)}{nk}$, $\zeta_2 \approx z_2 + z_1\frac{4\beta(k-1)}{nk}$, if $\beta \ll n$, where z_1 and z_2 are the average number of 1-hop and 2-hop nodes from a randomly chosen node within a domain in $\mathcal{F}$, respectively.

By applying (3.8), which gives an estimation of the path cost between two random nodes within a domain, and substituting relevant parameters of the RDPN, we can express the path cost between two random nodes in an RDPN as $g(k)$, a function of the number of domains (k) in the RDPN :

$$g(k) = \frac{\log(nk/\zeta_1)}{\log(\zeta_2/\zeta_1)} + 1, \tag{3.13}$$

where ζ_1 and ζ_2 are defined in Lemma 3.5. Equation (3.13) estimates the path

cost between two random nodes. However, as discussed in Section 3.6, path construction needs to consider the gateway selection in domains that are not the destination domain. Similarly, in an RC that does not contain the destination node, the constructed path in its RDPN is the minimum-cost path from a random vertex to a random vertex set with the cardinality γ. Therefore, by applying (3.10) and considering the probability of a random vertex not belonging to the random vertex set, the path cost in an RDPN that does not include the destination node is

$$h(k) = \begin{cases} \frac{nk-\gamma}{nk}\left(\frac{\log(\frac{nk+1-\gamma}{\zeta_1\gamma})}{\log(\zeta_2/\zeta_1)}+1\right) & \text{if } \gamma \leq \frac{nk+1}{\zeta_1+1}, \\ \frac{nk-\gamma}{nk} & \text{otherwise.} \end{cases} \tag{3.14}$$

where k is the number of domains in this RDPN.

3.7.3 APC lower bound for PS

For PS of the synchronisation radius τ, again, we use a line network with $\Delta+1$ domains to compute the APC lower bound under PS. Such line network is divided into η_1+1 (when $(\Delta+1) \mod \tau' = 0$) or η_1+2 (when $(\Delta+1) \mod \tau' > 0$) RCs, where $\tau' = \min\{\tau, \Delta+1\}$ and $\eta_1 = \lfloor \frac{\Delta+1}{\tau'} \rfloor - 1$. Moreover, the number of domains in the RC that does not include the destination node is always τ', whereas the number of domains in the RC that includes the destination node is $\eta' = (\Delta+1) \mod \tau'$ when $\eta' \neq 0$, or τ' when $\eta' = 0$.

In a line network, the path cost in all RCs, excluding the one with the destination node, is estimated by (3.14), whereas the path cost in the RC that includes the destination node is estimated by (3.13). Thus, the APC lower bound under PS is

$$L_{\text{PS}}^{\text{lower}} = \begin{cases} \eta_1(h(\tau')+1)+g(\tau') & \text{if } \eta' = 0, \\ (\eta_1+1)(h(\tau')+1)+g(\eta') & \text{if } \eta' > 0. \end{cases} \tag{3.15}$$

Hence, when $\eta' = 0$,

$$L_{\text{PS}}^{\text{lower}} \overset{\eta'=0}{=} \begin{cases} \eta_1\Big(\frac{\xi \log(\frac{n\tau'+1-\gamma}{\zeta_1\gamma})}{\log(\zeta_2/\zeta_1)} + \xi + 1\Big) \\ +\frac{\log(n\tau'/\zeta_1)}{\log(\zeta_2/\zeta_1)} + 1 & \text{if } \gamma \leq \frac{n\tau+1}{\zeta_1+1}, \\ \eta_1(\xi+1) + \frac{\log(n\tau'/\zeta_1)}{\log(\zeta_2/\zeta_1)} + 1 & \text{otherwise}, \end{cases} \tag{3.16}$$

where $\xi = 1 - \frac{\gamma}{n\tau'}$. When $\eta' > 0$, we have $\eta_2 = (\Delta + 1) \mod \tau' = \eta'$; therefore,

$$L_{\text{PS}}^{\text{lower}} \overset{\eta'\geq 0}{=} \begin{cases} (\eta_1+1)\Big(\frac{\xi \log(\frac{n\tau'+1-\gamma}{\zeta_1\gamma})}{\log(\zeta_2/\zeta_1)} + \xi + 1\Big) \\ +\frac{\log(n\eta'/\zeta_1)}{\log(\zeta_2/\zeta_1)} + 1 & \text{if } \gamma \leq \frac{n\tau+1}{\zeta_1+1}, \\ (\eta_1+1)(\xi+1) + \frac{\log(n\eta'/\zeta_1)}{\log(\zeta_2/\zeta_1)} + 1 & \text{otherwise}. \end{cases} \tag{3.17}$$

Clearly, $L_{\text{PS}}^{\text{lower}}$ is linear in the number of RCs, and is logarithmic in network structural parameters such as n and γ. This suggests that enlarging the synchronisation radius to reduce the number of RCs on the domain-wise path results in near linear reduction in APC.

3.8 APC for CS in Type-1 Networks

For complete synchronisation (CS), since all SDN domains are synchronised, controllers can make the global optimal decisions that generate the end-to-end path with minimum path cost. In this regard, we first study whether RCPC can construct such a global optimal path, and then establish the APC expression under CS in Type-1 Networks.

Given two arbitrary nodes v_1 and v_2, suppose the shortest domain-wise path $\mathcal{P}^*$ w.r.t. v_1 and v_2 contains k vertices in the domain-wise topology. If $\mathcal{P}^*$ (selected by RCPC) corresponds to the minimum-cost path between v_1 and v_2, then the APC lower bound under CS can be easily obtained by calculating the APC between two

random nodes in the end-domains of a line network consisting of k domains. However, the global minimum-cost path may visit more than k domains to yield the minimum end-to-end path cost. We, therefore, examine how the domain-wise shortest path $\mathcal{P}^*$ is related to the global minimum-cost path between v_1 and v_2 in the following.

Theorem 3.6. Let $L_k(\beta)$ be the APC between two random nodes in the two end-domains of a line network, which consists of k domains and all inter-domain connections are governed by parameter β. Then, $L_k(\beta) < L_{k+1}(\beta)$ when $k \geq 3$.

Theorem 3.6 reveals an important property of $L_k(\beta)$, i.e., a longer domain-wise path incurs higher end-to-end path cost if the shortest domain-wise path between two nodes contains at least three vertices. See analysis and discussions on the two uncovered cases ($k = 1, 2$) in Appendix A.1.

An implicit assumption for Theorem 3.6 is that the domain-wise path associated with the constructed path is a *simple path*, i.e., a path without repeated vertices. To show that visiting more domains cannot construct a shorter end-to-end path, we still need to prove that visiting one domain more than once is also disadvantageous. To this end, we define $L'_k(\beta)$ which is the same as $L_k(\beta)$ except that the corresponding domain-wise path contains repeated vertices.

Corollary 3.7. For the two-layer network model, $L_k(\beta) < L'_{k'}(\beta)$ for $3 \leq k \leq k'$.

Theorem 3.6 together with Corollary 3.7 suggest the following corollary.

Corollary 3.8. For any source-destination node pair residing in different domains, on average, the optimal path between them traverses the minimum number of domains.

Recall that the average number of domains on the shortest domain-wise path between two random domains is $\Delta + 1 = \frac{\log(m/z'_1)}{\log(z'_2/z'_1)} + 2$. Therefore, we compute the

APC under CS based on a domain-wise path traversing $\Delta + 1$ domains. Under CS, the path construction in each domain is independent of other domains' structures, thus complicating the mathematical analysis. We, therefore, leverage RDPN of a line network with $\Delta + 1$ domains to estimate the APC for CS, which is a lower bound according to Theorem 3.4. Thus, reapplying (3.13) with $\Delta + 1$ as the input, we obtain the APC lower bound for CS, denoted by $L_{\text{CS}}^{\text{lower}}$:

$$L_{\text{CS}}^{\text{lower}} = g(\Delta + 1) = \frac{\log\left(\frac{n\log(m/z_1')}{\zeta_1\log(z_2'/z_1')} + \frac{2n}{\zeta_1}\right)}{\log(\zeta_2/\zeta_1)} + 1. \tag{3.18}$$

The expression of $L_{\text{CS}}^{\text{lower}}$ shows a function that bounds the APC under the best-case scenario, i.e., CS, which therefore is also a lower bound under other synchronisation scenarios. Since (3.18) is a logarithmic function of a logarithmic function, it suggests that the routing efficiency can be significant if CS is achieved in the network. Moreover, under CS, (3.18) is of the form of $\log(n\log(m))$, showing that the number of nodes n has a stronger impact than the number of domains m on the value of $L_{\text{CS}}^{\text{lower}}$, i.e., intra-domain routing is more critical.

3.9 Universal APC Lower Bound

In this section, we present the *Universal APC lower bound*, which provides an estimation of APC under any synchronisation levels for both Type-1 and Type-2 Networks. The phrase *lower bound* carries two separate meanings. First, it summarises the APC obtained for MS and the APC lower bounds obtained for PS and CS in Type-1 Networks. Second, since link preference is at least 1 for Type-2 Networks, this universal lower bound derived for Type-1 Networks also applies to Type-2 Networks.

Theorem 3.9. Universal APC lower bound: Given the synchronisation radius τ, the

lower bound of APC (denoted by L^{lower}) in the two-layer network model is

$$L^{\text{lower}} = \begin{cases} \frac{\eta_1 \xi \log(\frac{n\tau'+1-\gamma}{\zeta_1 \gamma})}{\log(\zeta_2/\zeta_1)} + \frac{\log(n\eta_2/\zeta_1)}{\log(\zeta_2/\zeta_1)} & \\ +\eta_1(\xi+1)+1 & \text{if } \gamma \leq \frac{n\tau'+1}{\zeta_1+1}, \\ \frac{\log(n\eta_2/\zeta_1)}{\log(\zeta_2/\zeta_1)} + \eta_1(\xi+1)+1 & \text{otherwise}, \end{cases} \tag{3.19}$$

where $\tau' = \min\{\tau, \Delta+1\}$, $\eta_1 = \lfloor(\Delta+1)/\tau'\rfloor - 1$, $\eta_2 = (\Delta \mod \tau') + 1$, and $\xi = 1 - \frac{\gamma}{n\tau'}$.

In (3.19), L^{lower}, requiring the network topologies and synchronisation levels as inputs, is a logarithmic function non-increasing with τ. Moreover, when the number of gateways in each domain is sufficiently large (i.e., large γ), the expression of L^{lower} is significantly simplified due to easier inter-domain routing. In addition, the synchronisation radius τ, representing different levels of inter-domain synchronisations, is instrumental in determining the APC lower bound L^{lower}. For example, (3.19) reduces to (3.12) for MS in Type-1 Networks when $\tau = 1$; (3.19) reduces to (3.16) for PS when $\eta' = 0$.

Remark: Since all link preference levels in Type-2 Networks are at least 1, this universal APC lower bound still holds in Type-2 Networks, thus providing insights into the routing performance under any synchronisation and network scenarios. In Sections 3.10–3.13, we derive fine-grained APC expressions under different synchronisation scenarios in Type-2 Networks. More importantly, these fine-grained APC expressions can also be applied to Type-1 Networks by setting all edge weights to 1.

3.10 APC for MS in Type-2 Networks

In this section, we present the APC expression under MS in Type-2 Networks, denoted by $L_{\text{MS}}^{\text{Type-2}}$. Though edges in Type-2 Networks exhibit various edge weights,

such weight information is not available to any controllers under MS. Thus, the path construction from the source to the destination is independent of the edge weight distributions. Recall that in our two-layer network model, all intra-domain link weights are modelled as a given i.i.d. r.v., denoted by W, and all inter-domain edges are of weight 1. Hence,

$$\begin{aligned} L_{\text{MS}}^{\text{Type-2}} &= \Delta \cdot (\mathbb{E}[|\mathcal{P}_1|] \cdot \mathbb{E}[W] + 1) + \mathbb{E}[|\mathcal{P}_{\Delta+1}|] \cdot \mathbb{E}[W] + \Delta \\ &= \left(\frac{(n-\gamma) l \Delta}{n} + \frac{\log(n/z_1)}{\log(z_2/z_1)} + 1 \right) \mathbb{E}[W] + \Delta, \end{aligned} \tag{3.20}$$

where $\mathcal{P}_i$ and l are defined in (3.1) and (3.10), respectively. Substituting the expressions of l and Δ into (3.20), we obtain the full expression of $L_{\text{MS}}^{\text{Type-2}}$ under MS in Type-2 Networks:

$$L_{\text{MS}}^{\text{Type-2}} = \begin{cases} \left(\frac{n-\gamma}{n} \left(\frac{\log(\frac{n+1-\gamma}{z_1 \gamma})}{\log(z_2/z_1)} + 1 \right) \left(\frac{\log(m/z_1')}{\log(z_2'/z_1')} \right. \right. \\ \left. + 1 \right) + \frac{\log(n/z_1)}{\log(z_2/z_1)} + 1 \Bigg) \mathbb{E}[W] \\ + \frac{\log(m/z_1')}{\log(z_2'/z_1')} + 1 & \text{if } \gamma \leq \frac{n+1}{z_1+1}, \\ \left(\frac{n-\gamma}{n} \left(\frac{\log(m/z_1')}{\log(z_2'/z_1')} + 1 \right) + \frac{\log(n/z_1)}{\log(z_2/z_1)} \right. \\ \left. + 1 \right) \mathbb{E}[W] + \frac{\log(m/z_1')}{\log(z_2'/z_1')} + 1 & \text{otherwise.} \end{cases} \tag{3.21}$$

It is verifiable that (3.21) is same as (3.12) when $\mathbb{E}[W] = 1$, i.e., the Type-2 Network is reduced to the Type-1 Network.

3.11 APC for SS in Type-2 Networks

SS is a special synchronisation scenario that only exists in Type-2 Networks. Similar to MS, under SS, no two domains synchronise. To analyse APC under SS, we first introduce a new concept, named *path cost distribution*, as the basis for further

analysis. Here is the sketch of our analytical methodology.

Sketch of Analytical Methodology:

a) We compute the distribution of the path cost (in terms of accumulated link preferences) between two random intra-domain nodes, called *intra-domain path cost distribution*;

b) By (3.1), we need to determine the average cost of $\mathcal{P}_i$ for $i = 1, 2, \ldots, \mu$. Since the total number of RCs is the same as MS, we have that the expected value of μ in (3.1) is $\Delta + 1$;

c) As all controllers involved in the path construction process follow the same procedure, similar to (3.7), it suffices to only quantify the average cost of $\mathcal{P}_1$ and $\mathcal{P}_{\Delta+1}$ using the intra-domain distance distribution derived in *a)*.

3.11.1 Intra-Domain Path Cost Distribution

In one domain, consider a path with λ links. Let $W_1, W_2, \ldots, W_\lambda$ be i.i.d. r.v. of link weights on this path with the probability density functions (pdf) being $f_{W_1}(x) = f_{W_2}(x) = \ldots = f_{W_\lambda}(x)$. Define r.v. $\mathcal{W}_\lambda := \sum_{i=1}^{\lambda} W_i$ as the accumulated weight on this path. Then the pdf of $\mathcal{W}_\lambda$ is the convolution of the pdfs of $W_1, W_2, \ldots, W_\lambda$, i.e., $f_{\mathcal{W}_\lambda}(x) = f_{W_1}(x) * f_{W_2}(x) * \ldots * f_{W_\lambda}(x)$. By the principle in mixture distribution [46], we still need to know the probability $p_{\mathcal{W}_\lambda}$ that the minimum-cost path between two random nodes contains λ links. By the concept of z_i defined in the analysis of MS (Section 3.6), we know that $p_{\mathcal{W}_\lambda}$ is determined by z_λ, i.e., $p_{\mathcal{W}_\lambda} = z_\lambda/n$ (since link weights are i.i.d.). Note that when $\lambda = 0$, $z_0 = 1$ and the cumulative distribution function (cdf) of $\mathcal{W}_0$ is a unit step function. Let r.v. $\mathcal{D}$ be the minimum path cost (in term of accumulated link weights) between two random nodes in one domain, with the pdf being $f_{\mathcal{D}}(x)$, i.e., intra-domain distance distribution. Then by mixture distribution, $f_{\mathcal{D}}(x)$ can be estimated as follows

$$f_{\mathcal{D}}(x) = \sum_{i=0}^{h_{\max}} p_{\mathcal{W}_i} f_{\mathcal{W}_i}(x) = \sum_{i=0}^{h_{\max}} \frac{z_i}{n} \cdot f_{\mathcal{W}_i}(x), \tag{3.22}$$

where $h_{\max} := \arg\max_i z_i$ s.t. $\sum_{i=0}^{h_{\max}} z_i \leq n$. Hence, the APC between two nodes in one domain $\mathbb{E}[\mathcal{D}]$ can be computed using (3.22).

3.11.2 Domain-wise Path

Though SS and MS represent different synchronisation levels, the corresponding domain-wise paths are exactly the same w.r.t. a pair of source and destination nodes in a given network. Thus, by (3.5), again, we have that μ in (3.1) equals $\Delta + 1$. Let $L(\mathcal{P})$ be the end-to-end accumulated link preferential levels (i.e., cost) of path $\mathcal{P}$, which is a random variable. Then the expectation of $L(\mathcal{P})$, i.e., the APC for SS in Type-2 Networks, denoted by $L_{\text{SS}}^{\text{Type-2}}$, is

$$\begin{aligned} L_{\text{SS}}^{\text{Type-2}} &= \mathbb{E}[L(\mathcal{P})] \\ &= \mathbb{E}[L(\mathcal{P}_1) + L(\mathcal{P}_2) + \ldots + L(\mathcal{P}_{\Delta+1})] + \Delta \\ &= \Delta \cdot \mathbb{E}[L(\mathcal{P}_1)] + \mathbb{E}[L(\mathcal{P}_{\Delta+1})] + \Delta \\ &= \Delta \cdot \mathbb{E}[L(\mathcal{P}_1)] + \mathbb{E}[\mathcal{D}] + \Delta. \end{aligned} \tag{3.23}$$

The reason for the last row in (3.23) is that $\mathbb{E}[L(\mathcal{P}_{\Delta+1})]$ essentially is the path cost between two nodes in one domain.

Thus, it suffices to determine $\mathbb{E}[L(\mathcal{P}_1)]$ next, i.e., the minimum path cost between a random node and the closest gateway in one domain connecting to the next domain on the domain-wise path.

3.11.3 Minimum Path Cost Between An Arbitrary Node and Gateways

Section 3.11.1 provides the estimation of path cost between two arbitrary nodes in Type-2 Networks. Based on (3.22), we quantify the path cost between an arbitrary node and the gateway that incurs the minimum path cost, which is formally presented in the following theorem.

Theorem 3.10. Let r.v. $M^{(\beta)}$ denote the path cost between an arbitrary node and the gateway in the candidate gateway set (established with parameter β) that incurs the minimum path cost. Then, the pdf of $M^{(\beta)}$ is:

$$f_{M^{(\beta)}}(x) = \begin{cases} (1 - F_{\mathcal{D}}(x-1))^{\beta} & \\ \quad - (1 - F_{\mathcal{D}}(x))^{\beta} & \text{for } x \geq 1, \\ \\ 1 - (1 - F_{\mathcal{D}}(0))^{\beta} & \text{for } x = 0. \end{cases} \tag{3.24}$$

With Theorem 3.10, we can derive $\mathbb{E}[L(\mathcal{P}_1)] = \mathbb{E}[M^{(\beta)}]$, using the pdf expression in Theorem 3.10.

Then, substituting (3.5), $\mathbb{E}[L(\mathcal{P}_1)]$, and $\mathbb{E}[\mathcal{D}]$ into (3.23), we get the expression of the APC under SS in Type-2 Networks, denoted by $L_{\text{SS}}^{\text{Type-2}}$:

$$L_{\text{SS}}^{\text{Type-2}} = \Delta \cdot \int_{x=0}^{+\infty} x f_{M^{(\beta)}}(x) + 1 + \int_{x=0}^{+\infty} x f_{\mathcal{D}}(x) + \Delta. \tag{3.25}$$

Comparing to $L_{\text{MS}}^{\text{Type-2}}$, the expression of $L_{\text{SS}}^{\text{Type-2}}$ is more complicated as we do not impose any constraint on the distributions of link preference levels. Nevertheless, it is verifiable that $L_{\text{SS}}^{\text{Type-2}}$ is smaller than $L_{\text{MS}}^{\text{Type-2}}$, thus bounded by $L_{\text{MS}}^{\text{Type-2}}$.

3.12 APC for PS in Type-2 Networks

To compute the corresponding APC under PS in Type-2 Networks, denoted by $L_{\text{PS}}^{\text{Type-2}}$, we first present the APC w.r.t. two nodes with their shortest domain-wise path containing exactly q vertices, denoted by L_q, in the following theorem.

Theorem 3.11. Let L_q denote the APC between two arbitrary nodes under PS with synchronisation radius τ in Type-2 Networks where there are q domains on the domain-wise path. Then, we have

$$L_q = \begin{cases} (q/\tau - 1) \cdot (\mathbb{E}[M_\tau^{(\beta)}] + 1) + \mathbb{E}[D_\tau^{(\beta)}] & \text{if } \theta = 0; \\ \\ (\lfloor q/\tau \rfloor - 1) \cdot (\mathbb{E}[M_\tau^{(\beta)}] + 1) + \mathbb{E}[D_\theta^{(\beta)}] & \text{if } \theta > 0, \end{cases} \tag{3.26}$$

where $\theta = q \mod \tau$, $M_i^{(\beta)}$ is the r.v. of the minimum path cost incurred in an RC with i non-destination domains, and $D_i^{(\beta)}$ is the r.v. of the minimum path cost incurred in the RC with $i-1$ non-destination domains and the destination domain.

Recall that the probability that two arbitrary nodes with their domain-wise path containing q domains is $z'_{q-1}/(m-1) \approx z'_{q-1}/m$. Therefore, the APC under PS in Type-2 Networks, $L_{\text{PS}}^{\text{Type-2}}$, is

$$L_{\text{PS}}^{\text{Type-2}} = \sum_{q=2}^{h'_{\max}+1} L_q z'_{q-1}/m, \tag{3.27}$$

where $h'_{\max} := \arg\max_i z'_i$ s.t. $1 + \sum_{i=1}^{h_{\max}} z'_i \leq m$. The accuracy of $L_{\text{PS}}^{\text{Type-2}}$ is evaluated in Section 3.14.

3.13 APC for CS in Type-2 Networks

For complete synchronisation (CS), globally optimal routing decisions are made in Type-2 Networks. Here, let $L_k(\beta) := \mathbb{E}[D_k^{(\beta)}]$, where $D_k^{(\beta)}$ is the r.v. of the minimum path cost incurred in the RC with $k-1$ non-destination domains and the destination domain. By close examination of $L_k(\beta)$, some additional conclusions are made in the following corollaries.

Corollary 3.12. For the two-layer network model, $L_{k+1}(1) - L_k(1) = \mathbb{E}[\mathcal{D}] + 1$.

Corollary 3.13. For the two-layer network model, $\lim_{\beta\to\infty}\big(L_{k+1}(\beta) - L_k(\beta)\big) = 1$.

Note that Theorem 3.6 and Corollary 3.7 remain valid for the Type-2 Network scenario, which suggest that for any source-destination node pair residing in different domains, on average, the optimal path between them traverses the minimum number of domains. Therefore, when the shortest domain-wise path between two nodes contains k vertices, we can use $L_k(\beta)$ to approximate the corresponding optimal APC. Thus, let $L_{\text{CS}}^{\text{Type-2}}$ denote the APC under CS in Type-2 Networks. We have

$$L_{\text{CS}}^{\text{Type-2}} \approx \sum_{k=2}^{h'_{\max}+1} L_k(\beta) z'_{k-1}/m = \sum_{k=2}^{h'_{\max}+1} \mathbb{E}[D_k^{(\beta)}] z'_{k-1}/m, \tag{3.28}$$

where $h'_{\max}$ is defined in (3.27). Though experiencing high complexity due to global cross-domain routing optimality, $L_{\text{CS}}^{\text{Type-2}}$ is shown in Section 3.14 to have high accuracy in estimating APC under CS.

3.14 Evaluations of the Developed Analytical Results

To evaluate our analytical results of distributed SDN for various synchronisation scenarios, we conduct two sets of experiments (called *Evaluation 1* and *Evaluation 2*), with different focus, on network topologies generated from both real and synthetic

datasets. In Evaluation 1, we test the accuracy of the asymptotic analysis presented in Theorem 3.1, which, in its concise form, demonstrates the interplay of different parameters in determining the overall APC. Second, we validate the accuracy of the derived fine-grained expressions for $L_{\text{MS}}^{\text{Type-2}}$, $L_{\text{SS}}^{\text{Type-2}}$, $L_{\text{PS}}^{\text{Type-2}}$, and $L_{\text{CS}}^{\text{Type-2}}$ in Type-2 Networks in Evaluation 2. We compare these theoretical results with the actual APCs collected from the above networks. Based on these evaluation results, we can validate the accuracy of our theoretical results and observe to what extent synchronisation levels and network structural properties affect APCs.

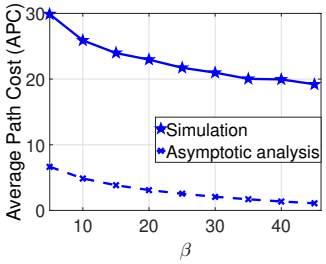


Figure 3.3: APCs for varying β.

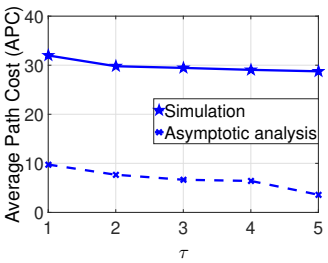


Figure 3.4: APCs for varying τ.

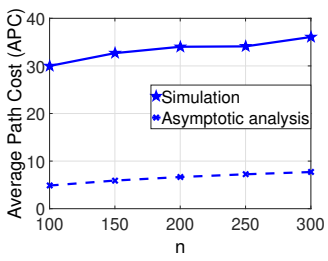


Figure 3.5: APCs for varying n.

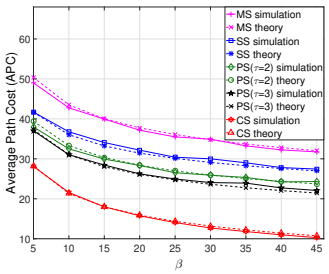


Figure 3.6: Case 1: Intra-domain and inter-domain degree distributions are derived from as20000102, Oregon1010331 datasets, respectively.

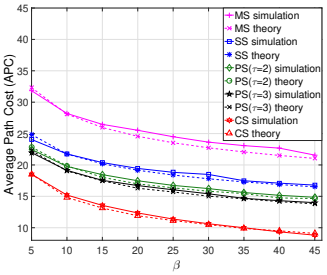


Figure 3.7: Case 2: Intra-domain and inter-domain degree distributions are derived from Rocketfuel AS-1239, CAIDA AS-27524, respectively.

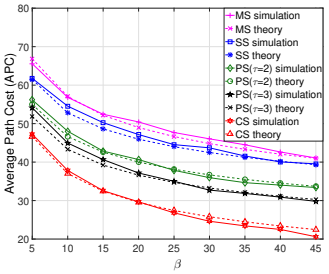


Figure 3.8: Case 3: Intra-domain and inter-domain degree distributions are derived from Barabási-Albert and Erdös-Rényi model, respectively.

3.14.1 Network Realisations

3.14.1.1 Network Topologies Based on Real Datasets

To generate network topologies based on real datasets, we need the degree distributions as the input. Specifically, we use the real datasets collected by the University of Oregon Route Views Project (Routeview project) [47], the Rocketfuel project [48], and the CAIDA project [49] for input degree distributions.

Given a specific degree distribution, one graph realisation is generated in the following way: We assign each vertex (the total number of vertices is given) a target degree according to the degree distribution. We then select two vertices randomly and add an edge between them; the number of edges added w.r.t. each vertex is then recorded. If the degree target w.r.t. a vertex is met, this vertex will not be selected again to connect with other vertices. This process repeats until all vertices reach their degree targets.

3.14.1.2 Network Topologies Based on Synthetic Models

We select Barabási-Albert [43] and Erdös-Rényi [42] models to generate network topologies.

a) Barabási-Albert (BA) model: BA model starts with a small connected graph of a few nodes/edges. Then, we sequentially add new nodes in the following way: For each new node v, we connect v to ϱ existing nodes such that the probability of connecting to node w is proportional to the degree of w. If the number of existing nodes is smaller than ϱ, then v connects to all existing nodes. Vertex degree for the BA model follows a near power-law distribution. BA graphs can be used to model some naturally occurring networks, e.g., social networks.

b) Erdös-Rényi (ER) model: For the ER model, the graph is generated by independently adding an edge between two nodes with a fixed probability p. The result is

a purely random topology where all graphs with an equal number of links are equally likely to be selected. Vertex degree under ER follows a binomial distribution.

Then, intra-/inter-domain topologies are generated based on the above network realisation methods. Next, on top of the generated inter-domain topologies, gateway connections are constructed according to parameter β, and intra-domain links are associated with link preference no less than 1.

Remark: It should be noted that the above network realisations are only for the evaluation purpose. Our developed analytical results are generic and do not require specific topological conditions.

3.14.2 Evaluation Settings

3.14.2.1 Evaluation 1

We conduct three experiments in networks with varying gateway connection parameter β, varying synchronisation radius τ, and varying number of nodes in each domain n, respectively. The APCs collected from these simulated networks are compared with the predictions made by the asymptotic expressions. Intra-domain degree distributions for three experiments are all derived from Rocketfuel “AS 1239”, in which $z_1 = 6.165$, and $z_2 = 41.835$. We configure the domain-wise topologies to have an average domain-wise distance of 10, using statistics collected from CAIDA “AS 27524". Unless otherwise specified, the default parameter settings for three experiments are: $\beta = 5$, $\tau = 2$, and $n = 200$.

3.14.2.2 Evaluation 2

Three evaluation cases are studied to validate the derived fine-grained APC expressions. In particular, Case 1 and 2 use topologies generated based on degree distributions extracted from real network datasets downloaded from Stanford SNAP[50];

their names can be found in captions of Fig. 3.6-3.8. As for Case 3, synthetic data are used where all intra-domain topologies are BA graphs and the inter-domain topology is a ER graph (we pick $p = 0.015$ for ER graphs, $\varrho = 1$ for BA graphs). In all three cases, the distribution of link preference levels (weight) is derived from Rocketfuel topologies, i.e., the intra-domain link preference ranges from 1 to 16 with the expectation and variance being 3.2505 and 4.5779, respectively. For each case, the two-layer network consists of 100 domains, each containing 200 nodes, i.e., $m = 100$ and $n = 200$.

In addition, for PS, two special cases, i.e., $\tau = 2$ and $\tau = 3$, are studied to compare against other synchronisation scenarios. It should be noted that these settings are determined arbitrarily, as our analytical model does not require the input degree distributions to have any patterns/properties. In addition to the evaluation results presented for the three cases above, we conduct extensive evaluations using other randomly chosen datasets from Rocketfuel and CAIDA, for which similar results are generated. Thus, we select the three cases as representatives; others are omitted to save space and to avoid repetitive results.

3.14.3 Evaluation Results

The simulated APCs and the APCs estimated by the asymptotic analysis are presented in Fig. 3.3-3.5 for Evaluation 1. For Evaluation 2, the simulated APC averaged over all network realisations and source-destination node pairs are reported in Fig. 3.6-3.8, for the three simulation cases, respectively. It should be noted that the plotted simulated APCs are the average results of multiple network graph realisations sharing the same given network settings, as the simulation results of a single network instance cannot indicate the characteristics of the network with the given set of settings. Specifically, every curve plotted is the average of results of 30 topology realisations with 50 random source-destination pairs (in different domains) per

topology realisation.

3.14.3.1 Accuracy of the Theoretical Results

The asymptotic analysis is conducted to enable direct and clear observations of the relationships between APC and parameters related to synchronisation levels and network structural properties. The asymptotic analysis' ability to reveal these relationships is confirmed in Evaluation 1. From three figures in Fig. 3.3-3.5, we can see that the trends in APC changes with varying parameters are closely captured by the curves obtained using expressions of the asymptotic analysis, as the simulation and analysis curves have common shapes. The presence of the gap between two curves is due to the fact that the asymptotic analysis is only intended to highlight the relative relationship among different parameters in simple expressions, and thus it is not meant to be employed as an exact estimation. In comparison, the evaluations of various real/synthetic networks in Evaluation 2 demonstrated in Fig. 3.6-3.8 confirm the high accuracy of our fine-grained theoretical results in predicting the performance metric APC in distributed SDN networks. Specifically, the simulation curves can be closely approximated by the theoretical results for all values of β and synchronisation scenarios. Moreover, the theoretical results for PS and CS are obtained by an efficient computation method, which reduces calculation complexity.

3.14.3.2 APC Variations for Different Synchronisation Levels and Structural Parameters

Fig. 3.3-3.8 confirm that the APC in distributed SDN is related to the amount of information available to the controllers, i.e., synchronisation levels. As expected, higher synchronisation levels are superior in reducing APCs. This can be observed in Fig. 3.4, where APC decreases when the synchronisation radius τ gets larger. For Evaluation 2, Fig. 3.6-3.8 show that APC for CS corresponds to the minimum APC

that is achievable in all cases, i.e., a lower bound. By contrast, the results for MS act as an upper bound due to the minimum intra-/inter-domain information availability. Since the APC for MS is expressed as a logarithmic function (3.21), Fig. 3.6-3.8 show that even with the minimum synchronisation level, APC is still relatively small given the network size ($20,000$ nodes in total) when link preference levels are at least 1.

Fig. 3.6-3.8 show that comparing to MS, the APC reduction for CS can be up to 70%. Moreover, comparing to MS, only intra-domain link preference information is available to SS. Nevertheless, such additional information is able to reduce APC by up to 30%. However, when more synchronised information is available, the reduction in APC starts to degrade (i.e., diminishing return). In particular, for PS, comparing against the case of $\tau = 2$, the APC reduction for $\tau = 3$ is rather small, especially when β is small. This observation is also confirmed by Fig. 3.4 where the most significant decrease in APC takes place when τ changes from 1 to 2. Consequently, it is expected that with the increase of τ, the benefit to cost ratio declines sharply.

In addition, we observe that the network performance improves when β increases. This is intuitive as a large β directly renders higher probability of finding a shorter path, as there exist more inter-domain connections. In fact, Fig. 3.3 and Fig. 3.4 show that on average, increasing β is more effective in reducing APC than increasing the synchronisation radius.

Furthermore, Fig. 3.3 and Fig. 3.6-3.8 also demonstrate that APC converges to a certain value when β is large, which can be explained by Corollaries 3.12–3.13.

Finally, Fig. 3.5 reveals that the size of the network does not have a significant impact on APC. Specifically, when the number of nodes triples from 100 to 300 in each domain, APC only marginally increases by 6. Moreover, given that in Evaluation 1 there are on average 10 domains on the domain-wise path, this gives an

average increase of APC by 0.3 in each domain.

In summary, these evaluation results reveal that in distributed SDN, the performance improvement space is only marginal when domains synchronise with other domains in an increasingly larger radius, or when each domain adds more gateways while the number of existing gateways is already large. Such constraints need to be addressed in practical network design and optimisations.

CHAPTER 4

Analysis of Performance Enhancements by Controller Synchronisation - Approach II

4.1 Introduction

In this chapter, we continue with the theoretical quantification of performance enhancement brought by SDN controller synchronisations. Specifically, we introduce here the second family of analytical methods (Approach II), which focuses on developing more interpretable results to provide guidelines for synchronisation protocol designs. Therefore, compared to the fine-grained analytical results developed in the previous chapter, the results here are more coarse-grained but they more explicitly revel the interplay of various factors in determining the efficacy of controller synchronisation. The main differences between Chapter 3 and Chapter 4 are summarised as follows. (i) The analyses conducted in Chapter 4 cannot accommodate arbitrary synchronisation levels, which differs from the four canonical synchronisation levels assumed in Chapter 3. (ii) To make use of synchronised information from arbitrary synchronisation levels, the routing mechanism in Chapter 4 is updated based on that employed in 3; see Section 4.2.3 for details. (iii) The analytical methods used in Chapter 4 render results in the forms of performance bounds and the tightness proofs of such bounds; this is in contrast to the finer-grained but more

complex results obtained in 3. (iv) As the analytical results in Chapter 4 are more interpretable, we have dedicated discussions on the analytical results after all major theorems; see "Insights for protocol design" sections for details.

In addition, here we provide a brief summary to highlight the main results obtained in this chapter. Specifically, our analytical results in this chapter reveal the relative contributions of different parameters to the lower bound of the performance metric. For example, the number of domains on constructed paths contributes linearly to the overall performance metric lower bound; while the number of gateway nodes in domains contribute logarithmically (see Theorem 4.2). Moreover, this lower bound shows that the performance metric is a linear function of the number of routing units (called *routing clusters*; see Definition 8) that are used for path constructions, but is independent of the detailed routing unit structures and distributions. Next, we quantify the tightness of such lower bounds and reveal the interplay among different parameters for various network scenarios. By all these theoretical results, we prove that the contribution of network synchronisation levels depends on specific network parameter settings, which therefore provides insights into real network protocol design. Finally, to validate the accuracy of the derived analytical expressions, they are compared against evaluation results obtained using simulation networks constructed using both real and synthetic network datasets.

As the technical contents in this chapter share the similar contexts as in Chapter 3, refer to Section 3.2 for related work. In addition, the proofs of all theorems and corollaries of this chapter can be found in Appendix A.2.

4.2 Problem Formulation

4.2.1 Link Preference and Path Cost

The analytical methods proposed in this chapter are also based on the analysis of routing performance of paths constructed between arbitrary node pairs in the distributed SDN network. Therefore, we again use the concept of path cost, which is discussed in Section 2.3, to quantify the performance of constructed routing paths. Since the link preference (weight) can be dynamic, we use random variables to capture its dynamicity. Specifically, we assume that intra-domain link weights across all domains are at least 1 and i.i.d. Furthermore, in real distributed SDN environment, unlike the intra-domain links which are potentially wireless, inter-domain gateway-to-gateway links are likely to be wired with high bandwidth, thus more stable. In this regard, we characterize all inter-domain link weights by a non-negative constant C. Furthermore, without loss of generality, we assume $C = 1$; all our theoretical results can be trivially extended to other values of C, if the weights of inter-domain links can be captured by random variables with certain distributions.

4.2.2 Synchronisation Among SDN Controllers

Similar to the previous chapter, the controller synchronisation is defined in Definition 1 in Section 3.3.1. Note that for the analysis in this chapter, we do not have any requirements on how controllers are synchronised with each other. Instead, we assume that the synchronisation status, i.e., which domains are synchronised with which other domains, is known to us for analysis. Moreover, we assume that each controller always has the up-to-date complete view (i.e., any path cost) of its own domain by proactive polling or passive information collection. Moreover, we assume that the domain-wise topology is always known to every controller irrespective of the inter-domain synchronisation status. In real networks, such domain-wise topology

can be identified by techniques such as the BGP protocol.

4.2.3 Routing Mechanisms

With the distributed SDN architecture and the given inter-controller synchronisation status, we then need a routing mechanism to respond to a path construction request between two arbitrary nodes (potentially in two different domains). The aim of the routing mechanism is to minimise the associated path cost under the given network synchronisation status. The general rule governing the routing mechanism is to first determine which domains are involved by finding a *domain-wise path*, defined as follows:

Definition 7.

a) In the domain-wise topology $\mathcal{G}_d$, the top-layer vertex corresponding to domain $\mathcal{A}$ in $\mathcal{G}$ is denoted by $\vartheta(\mathcal{A})$;

b) Given a pair of source and destination nodes v_1 and v_2 with [1] $v_1 \in \mathcal{A}_1$, $v_2 \in \mathcal{A}_2$, and $\mathcal{A}_1 \neq \mathcal{A}_2$, the *domain-wise path* w.r.t. v_1 and v_2 is a path (not necessarily the shortest) in $\mathcal{G}_d$ starting at vertex $\vartheta(\mathcal{A}_1)$ and terminating at vertex $\vartheta(\mathcal{A}_2)$ without containing any repeated vertices.

Note that although the information of the network domain-wise topology is always known, the domain-wise path is selected according to the given network policies (e.g., load balancing, security issues, etc.). The routing mechanism constructs a path segment in each involved domain, and then concatenate all these segments into one end-to-end path; see Section 4.3 for the details of the routing mechanism.

[1] In this thesis, for graph $\mathcal{G} = (V, E)$, by abusing graph theory notations, we use vertex $v \in \mathcal{G}$ to denote $v \in V$ and edge $e \in \mathcal{G}$ to denote $e \in E$.

4.2.4 Problem Statement and Objective

Regarding a path construction request for two nodes, after the (policy-based) selection of a domain-wise path, the path construction quality depends on the synchronisation status of the involved domains. For instance, if every two domains on this domain-wise path are synchronised, then the minimum end-to-end path can be constructed; however, if no domains are synchronised, then the path construction falls back to follow BGP-like protocols. To quantify the performance of the constructed routing path in a selected domain-wise path under any given inter-domain synchronisation status, we employ the APC, discussed and defined in Section 2.3, as the performance metric. Here APC is a natural generalised performance metric, as link weights are dynamically adjusted by controllers based on the current network status to reflect time-varying link preference, and APC is equivalent to the number of hops in unweighted networks. Formally, our research objective is:

Objective: Given the distributed SDN network model, suppose each network instance following the two-layer network model exists with the same probability. For an arbitrary source-destination node pair with their selected domain-wise path containing m domains ($m \geq 2$), our goal is to derive the mathematical expression of APC for these node pairs under any arbitrarily given network synchronisation scenario.

Note that our two-layer network model is a random graph model, i.e., there exists multiple network realisations satisfying the same set of input parameters. Therefore, APC is an expected value over not only random source/destination node pairs (in a domain-wise path containing a particular number of domains) but also random network realisations. All our theoretical results on APC are based on the given network parameters rather than a specific network realisation.

Example: Fig. 4.1 shows a sample domain-wise path consisting of 8 domains ($m = 8$) w.r.t. a path construction request between two random nodes $v_1 \in \mathcal{A}_1$

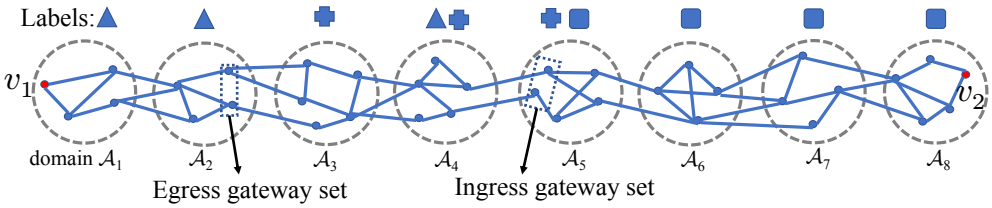


Figure 4.1: Sample domain-wise path between source/destination pair v_1/v_2; domains with the same labels are synchronised.

and $v_2 \in \mathcal{A}_8$, where all intra-/inter-domain connections are captured by the two-layer model. Moreover, Fig. 4.1 also illustrates one sample synchronisation status among these 8 domains. Since the number of synchronisation scenarios among these 8 domains can be up to $2^{\binom{8}{2}}$, it potentially affects the performance of the constructed path. We therefore aim to quantify APC between v_1 and v_2 for any given m and synchronisation scenario.

4.3 Routing Cluster-based Path Construction

The prerequisite for the analysis of APC between two random nodes is a routing mechanism that leverages the given inter-domain synchronised information. To this end, we describe a path construction mechanism called *Routing Cluster-based Path Construction (RCPC)*. It should be noted that our intention here is *not* to design an optimal routing mechanism. Instead, our goal is to quantify APC under a basic and representative routing mechanism. In cases where improved routing mechanisms are applied to distributed SDN, our theoretical results can serve as a performance bound.

4.3.1 Routing Clusters

As the name suggests, RCPC is based on the concept of *routing clusters*, which is defined below.

Definition 8. For a domain-wise path with all involved domains sequentially labeled

as $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_m$, domains $\mathcal{A}_{i+1}, \mathcal{A}_{i+2}, \ldots, \mathcal{A}_{i+c}$ $(0 \leq i \leq m-c)$ that are pairwise synchronised form a *Routing Cluster (RC)* containing c domains.

The concept of RC provides a method to leverage the given inter-domain synchronised information. Specifically, domains belonging to the same RC are all synchronised. Therefore, every controller in this RC is able to determine the optimal intra-RC path w.r.t. any two nodes within this RC, i.e., controllers within the same RC can be regarded as one logical controller. On the other hand, to ensure the optimality of the constructed intra-RC path, we also require the indices of the involved domains be continuous. This can be explained by the example in Fig. 4.1, where although $\mathcal{A}_1$, $\mathcal{A}_2$, and $\mathcal{A}_4$ are all synchronised, it is impossible for controllers in these three domains to compute the minimum cost path for any node pairs within them, because the path cost information in $\mathcal{A}_3$ is unknown.

Next, to efficiently utilise the synchronised information on the domain-wise path, we also need to partition all involved domains into different RCs. Note that two RCs may have overlapped domains. If this happens, then the controller in the overlapped domain may have conflicts in determining path constructions under different RC memberships. Therefore, we only consider non-overlapping RCs as follows.

Definition 9. *RC partition* of domains on a given domain-wise path is a set $\mathcal{R}$ of RCs, such that (i) each domain on this domain-wise path belongs to one and only one RC, and (ii) $|\mathcal{R}|$ is minimised.

In Definition 9, the first condition guarantees that RCs do not overlap, and the second condition ensures there are as many domains as possible in each RC, so that the optimal path traversing multiple domains (within the same RC) can be found. For instance, in Fig. 4.1, one way for RC partition is $\text{RC}_1 = \{\mathcal{A}_1, \mathcal{A}_2\}$, $\text{RC}_2 = \{\mathcal{A}_3, \mathcal{A}_4, \mathcal{A}_5\}$, and $\text{RC}_3 = \{\mathcal{A}_6, \mathcal{A}_7, \mathcal{A}_8\}$. Note that the RC partition is not unique. For example, another way for RC partition in Fig. 4.1 is $\text{RC}_1 = \{\mathcal{A}_1, \mathcal{A}_2\}$, $\text{RC}'_2 =$

$\{\mathcal{A}_3, \mathcal{A}_4\}$, and $\text{RC}'_3 = \{\mathcal{A}_5, \mathcal{A}_6, \mathcal{A}_7, \mathcal{A}_8\}$. In Section 4.4, we will show how different RC partitions (e.g., the number of RCs and the size of each RC) affect the APC. Based on the RC partition, we are now ready to present the routing mechanism - RCPC.

4.3.2 Routing Cluster-based Path Construction (RCPC)

RCPC is described in the following steps:

Step 1) W.r.t. two nodes v_1 and v_2 on a selected domain-wise path containing m domains, identify the domains on this domain-wise path;

Step 2) Based on the given synchronisation status of all involved domains, partition these domains into the minimum number of μ ($1 \leq \mu \leq m$) RCs according to Definition 9;

Step 3) For each RC_i, a path segment starting from the entering node (which is v_1 if $i = 1$, or is specified by RC_{i-1}) and terminating at one of the exiting nodes (which are gateways connecting to RC_{i+1}, or node v_2 if $i = \mu$) with the minimum cost is constructed. Such path segment is denoted by $\mathcal{P}_i$ in RC_i. Also let $e_{i,i+1}$ be the edge leading from $\mathcal{P}_i$ in RC_i to connect to the entering node in RC_{i+1} if $i \leq \mu - 1$;

Step 4) The final v_1-to-v_2 path $\mathcal{P}$ is $\mathcal{P} = \mathcal{P}_1 + e_{1,2} + \mathcal{P}_2 + e_{2,3} + \ldots + \mathcal{P}_{\mu-1} + e_{\mu-1,\mu} + \mathcal{P}_\mu$.

In essence, RCPC intends to minimise the path cost at each RC. As such, ECMP or similar schemes could be applied within RCs for traffic balancing or other control objectives; since equal intra-RC costs would incur as the result, our analytical results still hold. The fact that, instead of each domain making routing decisions based on its view of the network, path construction inside one RC is agreed using commonly synchronised information and carried out consistently in all domains within the RC is to avoid routing loops and other anomalies which are likely to arise because of the mixed centralised-distributed routing. This is analogous to the routing mechanism in ONOS [24] and OpenDaylight [51] in the sense that, for these two controller ar-

chitectures, the domains on a domain-wise path essentially form one RC and they jointly install the same set of forwarding rules. However, the inter-domain synchronised information may be out-of-date, thus inaccurate. By contrast, RCPC partitions involved domains into multiple RCs with the most recent synchronised information among controllers.

4.4 APC for Any Given Synchronisation Status

With RCPC, we are ready to analyse APC of the path constructed between source and destination nodes, and investigate its relationship with various network structural and synchronisation-related parameters. In particular, we first derive the expressions of the lower bound of APC. These results explicitly show how different parameters interact with each other in determining the overall APC. In addition, to guarantee the tightness of our developed APC lower bounds, we also provide the maximum gap between the actual APC and the derived APC lower bounds. The significance of such APC lower bounds is that they provide insights into how the constructed path quality relates to any given network parameters without assuming any inter-domain synchronisation models.

The basic idea of our derivation is to explore how path construction behaviours differ in different types of domains in each RC, and understand how such behaviours are influenced by intrinsic topological properties and the synchronised information. Specifically, we give a sketch of our methodology.

Sketch of Analytical Methodology:

1) Given the domain-wise path with m domains, and the corresponding RC partitions, identify different types of domains (see Definition 10) based on their relative positions in RCs;

2) Derive the APC lower bound for different types of domains and add these

lower bounds together to obtain the overall APC lower bound;

3) Quantify the maximum gap between the actual APC and APC lower bound by comparing the APC lower bound with the APC incurred without synchronised information.

Before discussing technical details, for ease of presentation, we first formally introduce the following definitions.

Definition 10.

a) Source/destination domain: the domain where the source/destination node is located;

b) Type-1 domain: the domain in an RC where the constructed path segment in this RC starts; the source domain is also a Type-1 domain;

c) Type-2 domain: any domain that is not a Type-1 domain or the destination domain;

d) Ingress and egress gateway sets: the sets of gateways in a domain through which data packets can enter or leave this domain w.r.t. the domain-wise path selected between the source and the destination node.

Examples: In Fig. 4.1, based on the RC partition $\text{RC}_1 = \{\mathcal{A}_1, \mathcal{A}_2\}$, $\text{RC}_2 = \{\mathcal{A}_3, \mathcal{A}_4, \mathcal{A}_5\}$, and $\text{RC}_3 = \{\mathcal{A}_6, \mathcal{A}_7, \mathcal{A}_8\}$, $\mathcal{A}_1$, $\mathcal{A}_3$ and $\mathcal{A}_6$ are Type-1 domains; $\mathcal{A}_2$, $\mathcal{A}_4$, $\mathcal{A}_5$ and $\mathcal{A}_7$ are Type-2 domains. The ingress and egress gateway sets are illustrated in $\mathcal{A}_5$ and $\mathcal{A}_2$ in Fig. 4.1, respectively.

Remark: The difference between Type-1 and Type-2 domains is that the starting nodes in Type-1 domains are not decided by controllers in their associated RCs, since the starting node is either the source node or an ingress gateway chosen by the previous RC. In contrast, both ingress and egress gateways in Type-2 domains are jointly decided by controllers in their associated RCs, using the synchronised information they all share. As for the destination domain, if it constitutes an RC itself, then the ingress gateway is decided by the previous RC; otherwise, it is decided

Table 4.1: Main Notations

Notation	Meaning
m	the number of domains on the domain-wise path
n	the number of nodes in each domain
γ	the average number of ingress or egress gateways in a domain w.r.t. a domain-wise path
z_1, z_2	average number of nodes within the same domain that are 1-hop and 2-hop away from a random chosen node
μ	the number of Routing Clusters (see Definition 8) on the domain-wise path
ϕ	$\phi = n/z_1$
ξ	the probability that the ingress node is not in the egress gateway set in a domain
ζ	the probability that the ingress and egress gateway sets in a domain do not have common elements
ω	maximum link preference (weight) in the network

jointly by all controllers within its associated RC. Whether or not the starting node of a domain can be manipulated determines the minimum APC achievable in this domain. Specifically, we derive the minimum APC traversed in different types of domains in Theorem 4.1, which then serves as the lower bound. Before presenting Theorem 4.1, we first define the path cost between two sets of nodes as follows.

Definition 11. The *path cost between node sets* W *and* S is the minimum path cost among all paths in set $\{\mathcal{P}_{ws}$: minimum cost path between node w and node s, $w \in W, s \in S\}$.

Theorem 4.1. Let l_1 denote the APC between the starting node and the egress gateway set for Type-1 domains, and let l_2 denote the APC between the ingress and egress gateway sets for Type-2 domains. Then, when $\omega = 1$,

$$l_1 \geq \begin{cases} \zeta\left(\frac{\log(\phi/\gamma)}{\log(z_2/z_1)} + 1\right) & \text{if } \gamma \leq \phi, \\ \zeta & \text{otherwise,} \end{cases}$$

and

$$l_2 = \begin{cases} \zeta\left(\frac{\log(\phi/\gamma^2)}{\log(z_2/z_1)} + 1\right) & \text{if } \gamma \leq \sqrt{\phi}, \\ \zeta & \text{otherwise,} \end{cases}$$

where ζ is the probability that the ingress and egress gateway sets in a domain do not have common elements, and other notations are defined in Table 4.1.

In Theorem 4.1, according to the definition of ζ, its value is $\zeta = \Pi_{i=1}^{\gamma-1}(1 - \frac{\gamma}{n-i})$ if $\gamma \leq n/2$, or $\zeta = 0$ if $\gamma > n/2$. Note that Theorem 4.1 is derived by setting all link weights to 1 (i.e., equal link preference). In this way, the mathematical expression of APC is simplified. More importantly, since link weights are at least 1 in the distributed SDN model, all derived APC lower bounds are still valid in graphs with dynamic weights. Theorem 4.1 tells us that the minimum APC exhibits different behaviours, depending on the range of the number of ingress/egress gateways. As such, we discuss the APC lower bound in the following three regimes of γ: *sparse inter-domain connections ($\gamma \leq \sqrt{\phi}$), medium inter-domain connections ($\sqrt{\phi} < \gamma \leq \phi$),* and *dense inter-domain connections ($\gamma > \phi$)*, where $\phi = \frac{n}{z_1}$.

4.4.1 Sparse Inter-domain Connections ($\gamma \leq \sqrt{\phi}$)

We present the APC lower bound when the number of gateways is relatively small, i.e., $\gamma \leq \sqrt{\phi}$.

Theorem 4.2. When $\gamma \leq \sqrt{\phi}$, the APC lower bound in a domain-wise path with m domains partitioned into μ RCs, denoted by L^{LB}, is

$$L^{\text{LB}} = \frac{\mu\zeta\log(\gamma) + m\zeta\log(\frac{\phi z_2/z_1}{\gamma^2})}{\log(z_2/z_1)} - 1.$$

Before we draw observations from the expression of L^{LB}, we first show the tightness of such lower bound to justify its close representation of the APC. The basic idea behind the derivation of the gap between APC and APC lower bound is to find

a tractable APC upper bound, and use the difference between this APC upper bound and L^{LB} to capture the maximum gap. Such APC upper bound is estimated by assuming that each domain is an RC, i.e., no inter-domain synchronised information is available. Although this APC upper bound fails to capture controller synchronisations, it helps us quantify the tightness of derived APC lower-bound. The tightness guarantee is provided in the following theorem.

Theorem 4.3. Let L denote the APC in a domain-wise path with m domains partitioned into μ RCs, and ω the maximum link weight in the network. Then, when $\gamma \leq \sqrt{\phi}$, the gap between L and L^{LB} is bounded by

$$L - L^{\text{LB}} \leq \frac{\rho \log(\gamma) + \nu \log(\phi)}{\log(z_2/z_1)},$$

where $\rho = \omega\xi(1-m) + \zeta(2m-\mu)$, $\nu = \omega(\xi(m-1)+\zeta) - \zeta m$.

From the expressions in Theorems 4.2-4.3, they show that neither APC lower bound nor the APC tightness expression are affected by the specific number of domains in each RC. That is to say, the quality of two paths constructed by RCPC are similar, as long as they have the same number of domains and RCs. Furthermore, the APC lower bound grows linearly in both μ and m. Nevertheless, as γ increases, the influence of μ becomes more dominant according to Theorem 4.2. In contrast, m is more important when γ is small. Intuitively, this is because when the number of gateways is small, there are not many gateway options for inter-domain routing, and thus controller synchronisations do not improve the quality of routing that much. In other words, even a random selection of gateways has a relatively high probability of hitting the best choice. This is also echoed by Theorem 4.3 where the upper bound of $L - L^{\text{LB}}$, which represents the gap between the actual APC and the derived APC lower bound, increases when γ gets larger, as the informed routing makes a significant difference when there are more gateways to choose from. Moreover, by

Theorem 4.3, in the extreme case where there is no inter-domain synchronisations (i.e., $m = \mu$) and all link weights are 1 (i.e., $\omega = 1$), with small number of gateways, we have $L - L^{\text{LB}} \leq \log(\gamma)/\log(z_2/z_1) \approx 0$. Therefore, L^{LB} becomes the accurate expression of APC.

Insights for protocol design: When inter-domain connections are sparse, having more inter-domain connections enhances the benefit of improved inter-domain synchronisations. On the other hand, when the number of inter-domain connections is really small, efforts should be diverted to increase domain-wise topological connectivity to reduce the length of domain-wise path m, which will contribute more in reducing APC, due to the effect of the reduction of m is magnified by a relatively large coefficient $\log(\frac{\phi z_2/z_1}{\gamma^2})$ of m.

4.4.2 Medium Inter-domain Connections ($\sqrt{\phi} < \gamma \leq \phi$)

Similar to the previous case, we also provide the APC lower bound and its tightness guarantee in the case of medium inter-domain connections.

Theorem 4.4. When $\sqrt{\phi} < \gamma \leq \phi$, the APC lower-bound in a domain-wise path with m domains partitioned into μ RCs, denoted by L^{LB}, is

$$L^{\text{LB}} = \frac{\mu\zeta \log(\frac{\phi}{\gamma z_2/z_1})}{\log(z_2/z_1)} + m(\zeta + 1) - 1.$$

Theorem 4.5. Let L denote the APC in a domain-wise path with m domains partitioned into μ RCs. Then, when $\sqrt{\phi} < \gamma \leq \phi$, the gap between L and L^{LB} is bounded by

$$L - L^{\text{LB}} < \frac{\tau \log(\phi/\gamma) + \omega \log(\phi)}{\log(z_2/z_1)} - \zeta(m - \mu),$$

where $\tau = \xi\omega(m-1) - \zeta\mu$.

In the medium inter-domain connection regime ($\sqrt{\phi} < \gamma \leq \phi$), although L^{LB}

still grows linearly in μ and m, one noticeable difference comparing to sparse inter-domain connections is that the influence of μ diminishes when γ is large, which is the opposite of the case in Theorem 4.2. This suggests a weakening role of synchronisation (i.e., μ) in reducing APC when γ is large. By closer examinations, this can be explained by the fact that the relative abundance in egress gateways significantly reduces the APC between non-gateway nodes and gateways. One consequence of this is that even most domains operate only on the knowledge of their own domains without any inter-domain synchronised information, i.e., select the gateway that incurs the minimum path cost in each domain, the overall APC of paths constructed in such way is not much worsened. This revelation is also confirmed by the fact that the coefficient of m are only related to the intrinsic property of the network. Another difference comparing to the sparse inter-domain connection case is that the parameter $\phi = \frac{n}{z_1}$, which is an indicator of the intra-domain graph connectivity level, is now part of the coefficient of μ. This suggests that high network connectivity also reduces the impact of a small μ (i.e., richer synchronised information among controllers).

Insights for protocol design: In medium inter-domain connections, improving controller synchronisations is still helpful in reducing APC. However, the effectiveness of it is dwindling. Therefore, the synchronisation policy must consider the connectivity level of domains. Specifically, when the intra-domain connectivity level is relatively low, increasing the inter-controller synchronisation level is more rewarding.

4.4.3 Dense Inter-domain Connections ($\gamma > \phi$)

When the number of inter-domain connections is significantly large, the contribution of the increased inter-controller synchronisation level continues diminishing, as proved below.

Theorem 4.6. When $\gamma > \phi$, the APC lower-bound in a domain-wise path with m

domains partitioned into μ RCs, denoted by L^{LB}, is

$$L^{\text{LB}} = (1+\zeta)m - 1.$$

Theorem 4.7. Let L denote the APC in a domain-wise path with m domains partitioned into μ RCs. Then, when $\gamma > \phi$, the gap between L and L^{LB} is bounded by

$$L - L^{\text{LB}} < \frac{(\omega+\chi)\log(\phi) - \chi\log(\gamma)}{\log(z_2/z_1)} + \zeta\big((m-1)\omega - m\big),$$

where $\chi = \omega(\xi-\zeta)(m-1)$.

When $\gamma > \phi$, μ (which represents the level of controller synchronisations) disappears from the expressions in Theorems 4.6 and 4.7. This suggests that in the dense inter-domain connection regime ($\gamma > \phi$), the role of controller synchronisation matters little. This is the continuation of the trend observed in the medium inter-domain connection regime ($\sqrt{\phi} < \gamma \leq \phi$) where the influence of controller synchronisations is discounted as γ increases. Intuitively, this is because when γ is considerably large, every node inside a domain has a high probability to connect directly to nodes in its neighbouring domains, thus yielding the high probability that the ingress and egress gateways are the same in each domain.

Insights for protocol design: When the inter-domain connections are dense, the synchronisation of controllers achieves little in reducing the APC. Without additional synchronisation cost, a distributed routing algorithm, such as BGP, becomes a fair alternative in this case. Moreover, from Theorem 4.7, it is evident that the performance gap between APC and the APC lower bound is approximately the APC between two arbitrary nodes in the destination domain multiplied by the worst case link weight when γ is large (i.e., $\xi \approx 0$ and $\zeta \approx 0$). This can be explained as follows: For a domain-wise path, the destination domain is the only domain that has a specified ingress gateway and a destination node, i.e., the destination domain does not

have gateway options which may incur lower cost. As such, the destination domain becomes the bottleneck when γ is large. Therefore, the destination domain should be the focus of the synchronisation design when inter-domain connections are dense.

4.5 Evaluations of the Developed Analytical Results

A series of experiments based on network topologies generated from both real and synthetic datasets are conducted in this section. The focus is two-fold. First, we evaluate the developed analytical results' abilities in predicting APC changes for different network parameters. Second, we test the validity of insights our analytical results reveal in simulated networks.

4.5.1 Network Realisations

4.5.1.1 Network Topologies Based on Real Datasets

To generate network topologies based on real datasets, we refer to the Rocketfuel project [48] for the input node degree distribution to generate the network topology for each domain.

Given a specific node degree distribution, one graph realisation is generated in the following way: We assign each vertex a target degree according to the degree distribution. We then select two vertices randomly and add an edge between them; the number of edges added w.r.t. each vertex is then recorded. If the degree target w.r.t. a vertex is met, this vertex will not be selected again to connect with other vertices. Such process repeats until all vertices reach their degree targets.

4.5.1.2 Network Topologies Based on Synthetic Models

In our evaluations, we test the trend of APC changes with varying network parameters and compare them with our derived APC lower bounds. Since we are not

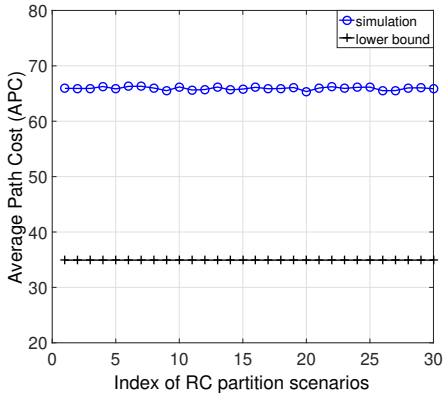


Figure 4.2: Scenario 1: APCs of paths constructed with different RC partitions ($m = 20$, $\mu = 5$).

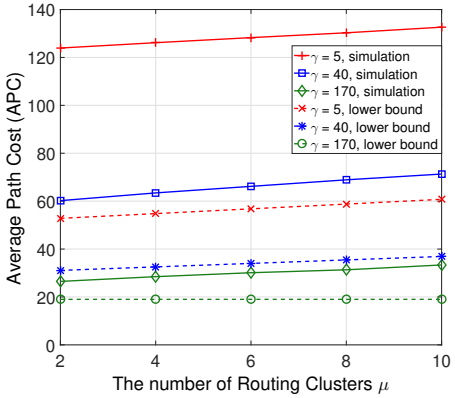


Figure 4.3: Scenario 2: APCs of paths constructed with increasing number of RCs ($m = 20$).

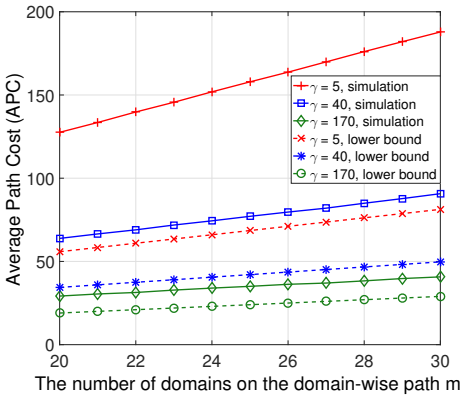


Figure 4.4: Scenario 3: APCs of paths constructed with increasing number of domains ($\mu = 5$).

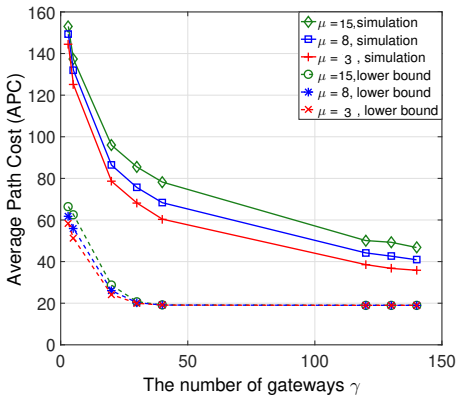


Figure 4.5: Scenario 4: APCs of paths with increasing number of gateways ($m = 20$).

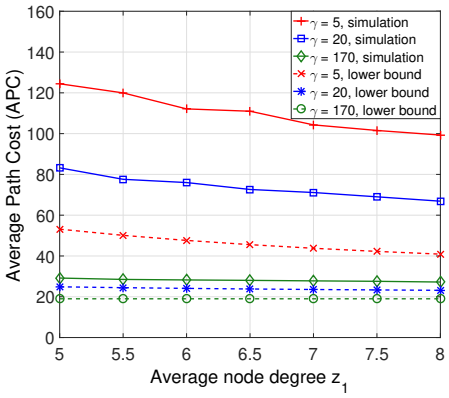


Figure 4.6: Scenario 5: APCs of paths with increasing number of average node degree ($m = 20$, $\mu = 5$).

always able to obtain real network datasets with desired varying network parameters that meet our requirement, we also use network topologies generated by synthetic datasets according to certain network models. Specifically, we select Erdös-Rényi [42] model to generate network topologies for some evaluations.

Erdös-Rényi (ER) model: An ER graph is generated by adding an edge between two nodes with a fixed probability p. The result is a purely random topology where all graphs with an equal number of links are equally likely to be selected. Vertex degree under ER model follows binomial distribution.

4.5.2 Evaluation Scenarios

We conduct our evaluations for the following scenarios, with different objectives: (i) Scenario 1, where we test the APC against different RC partitions that have the same total number of RCs on the domain-wise path; (ii) Scenario 2, where we test the APC for different numbers of RCs on the domain-wise path; (iii) Scenario 3, where we compare the APCs with different numbers of domains but the same number of RCs; (iv) Scenario 4, where we test the influence of the number of gateways on the APC; (v) Scenario 5, where we evaluate the APC for networks with different connectivity levels.

4.5.3 Evaluation Settings

All evaluations are conducted on graphs with 300 nodes in each domain, where the link preference (weight) of intra-domain links are i.i.d. random variable uniformly distributed between 1 and 5. Except for Scenario 3 where the number of domains on the domain-wise path is varied from 20 to 30, there are 20 domains on the domain-wise path from the source to the destination domain in all other scenarios. Network topologies of all domains for evaluation Scenarios 1 - 4 are generated using statistics collected from the RocketFuel project, where we use AS “orange1010331" that has

$10,670$ nodes whose node degrees are distributed between 2 and 49. The average number of 1-hop and 2-hop nodes from a random node in this AS are 3.16 and 13.31, respectively. All datasets used in our evaluations can be downloaded from the Stanford SNAP project website [50].The network topology for evaluation Scenario 5 is generated using the ER network model, because we need a continuous variation of the average node degree for the purpose of this evaluation.

For Scenarios 1, 3, and 5, the domain-wise path is partitioned into 5 RCs. Moreover, some evaluation scenarios require that the constructed path be partitioned into certain number of RCs. Since there are potentially many partitions that result in the same number of RCs, we randomly select 30 such partitions under the constraint of the required number of RCs (for Scenarios 1 - 5) and use results averaged over these partitions as evaluation results (for Scenarios 2 - 5). It should be noted that the plotted simulated APCs are the average results of multiple network graph realisations sharing the same given network settings, as the simulation results of a single network instance cannot indicate the characteristics of the network with the given set of settings. Specifically, for each evaluation scenario, 50 network graphs are realised, and with each realisation, 50 source-destination pairs are randomly picked from the source and destination domains for path constructions.

4.5.4 Evaluation Results

4.5.4.1 Accuracy of the APC prediction

In all evaluation scenarios, the derived APC lower bounds capture the trend of APC changes against variations in different network structural or synchronisation-related parameters. Specifically, Fig. 4.2 confirms our conclusions drawn from the derived APC lower bound that the APC is not affected by how domain-wise paths are organised into RCs, as long as these paths share the same number of domains and the same

number of RCs. Thus, in the following experiments, the lower-bound results stand independent of the specific instantiations of RC partition, as per instance RC partitions yield similar APC when they have the same number of RCs. Fig. 4.3 shows that the APC increases in μ for sparse ($\gamma = 5$) and medium ($\gamma = 40$) inter-domain connections, but stays relatively unchanged under dense inter-domain connections ($\gamma = 170$). This confirms our argument that controller synchronisations only exhibit limited contributions when the number of gateways is large. Fig. 4.4 demonstrates that the APC increases linearly in m under all inter-domain connection ranges. Furthermore, Fig. 4.5 and Fig. 4.6 display the non-linear relationships between the APC and the number of gateways and the average node degree, respectively. These relationships are anticipated by Theorems 4.2, 4.4, and 4.6. Finally, the gaps between evaluation results and APC lower bounds are mainly caused by the various link preference that is used for path constructions.

4.5.4.2 Contributions of different controller synchronisation levels and network structural parameters on the APC

The central question we aim to answer in this thesis is whether higher controller synchronisation levels always deliver performance improvements, and whether a complicated synchronisation protocol is needed. In addition, we also want to understand the influence of other parameters on APC. On top of our analytical results, these evaluation results further offer us answers to these questions. Specifically, the slope of the lines in Fig. 4.3 represents the rate of the APC increase when the number of RCs increases, which signifies a worsening controller synchronisation level. We observe that the slopes decrease when the number of gateways increases. This suggests that the worsening controller synchronisation levels affect networks with less number of gateways, thus showing the importance of having more gateways in situations where controller synchronisations are limited. Despite its role in reducing the APC, Fig. 4.5

shows the limit of having more gateways. This is demonstrated by the diminishing decrease in APC when the number of gateways continues to grow from an already significant amount. Fig. 4.6 reveals that the improved network connectivity level, reflected by the average node degree of intra-domain topologies, also benefits the reduction of the APC. Intuitively, this is because higher network connectivity levels essentially offer more path construction options, from which more cost-saving paths are likely to emerge. However, again such benefit becomes constrained when more gateways are added. In contrast, the results shown in Fig. 4.4 demonstrate that the influence on the APC from the number of domains on the domain-wise path is not strongly coupled with the number of gateways present in domains. This observation suggests that the effort invested in improving the connectivity of the domain-wise topology, which brings down the average number of domains on a domain-wise path, is always beneficial in reducing the APC.

4.6 Limitations of our Analytical Results

The analytical results obtained in Chapter 3 and Chapter 4 shed light on the benefits of controller synchronisation in relationship to various network structural properties. Although our work here is the first to investigate the subject from the graph theoretical perspective, we recognise the limitations of these results, which are discussed in this section.

First, some of the underlying assumptions we make in order to enable mathematically tractable analysis are not realistic in a practical distributed SDN network. For example, in order to have generic and simplified expressions of the APC and the APC bounds, we assume that the degree distribution in all domains are i.i.d. However, real SDN networks are likely to be heterogeneous systems with domains having varying structural properties. Therefore, the analytical results only serve as

guidelines for analysing distributed SDN networks with diverse domain structures.

Second, based on the network model, we use degree distribution of domain graph, which models the intra-domain topology, to characterise structural properties of domain topology. From the perspective of random graph, the degree distribution is the most important characteristic, which determines the statistic properties of the graph. However, communication networks, such as the distributed SDN network, may be described by other statistical models as well. For example, the clustering phenomenon in some real-world networks are attributed to the preferential attachment model [52]. Therefore, our results obtained based on this generic metric, i.e., degree distribution, capture the average behaviours of all graphs sharing the shame degree distribution, but not tailored for certain type of networks.

Third, the way to use synchronised information and the corresponding path construction mechanism employed in Chapter 3 and Chapter 4 are simple and representative mechanisms to assist our analyses. Although these mechanism designs are not our original intention nor focus in this thesis, they do impact the analytical results. Therefore, for real-world scenarios with different inter-domain routing paradigm than ours, the analytical results presented are not directly applicable.

The limitations discussed in this section are the results of a series of necessary trade-offs to balance among factors including the tractability of the problem, interpretability of the results and compatibility with existing SDN controller architectures. We recognise that these limitations constrain the applicability of our results; they nevertheless serve as guidelines, which reveal the relative importance of various factors in affecting the APC.

CHAPTER 5

DQ Scheduler: DRL-based Controller Synchronisation for Inter-domain Routing

---

5.1 Introduction

In this chapter, we explore DRL-based approaches for designing controller synchronisation policies in distributed SDN. In particular, as a starting point, we focus on the inter-domain routing task for which controller synchronisation helps improve the quality of inter-domain routing decisions.

Recall that for distributed SDN, physically distributed controllers synchronise with each other to maintain a logically centralised network view, which is referred to as *controller synchronisation*. Since complete synchronisation among controllers, i.e., all controllers always maintain the same global view, will incur high costs especially in large networks [18, 19], most practical distributed SDN networks can only afford partial inter-controller synchronisations and allow temporary inconsistency in controllers' network view, which is known as the *eventual consistency model* [10].

The eventual consistency model permits temporarily inconsistent network views among physical distributed controllers in the hope that all controllers will *eventually* be mutually updated. In the mean time, higher network availability, i.e., the ability to provide network services, is realised at the cost of temporary inconsistency according

to the CAP theorem [10], which states that it is not possible to provide consistency, availability and partition tolerance at the same time for distributed systems. Despite the fact that existing works recognize the problems caused by inconsistent network views [53], one crucial question that has been largely overlooked is *precisely how* controllers should synchronise with each other, under limited synchronisation budget, to minimise the performance degradation caused by such inconsistency. For example, ONOS [24], which is a state-of-the-art SDN controller, employs the anti-entropy protocol to realise the eventual consistency [54]. The gist of the anti-entropy protocol is that controllers use a simple gossip algorithm to *randomly* synchronise with each other. Although this protocol *can* achieve eventual consistency, is it a *wise* and *efficient* way?

Motivated by this question and inspired by recent success in applying RL techniques to solve complicated problems, we approach this controller synchronisation problem by formulating it as a MDP problem. Then, we design the DRL based *Deep-Q (DQ) Scheduler*, for which the goal is to maximise the long-term benefits of controller synchronisations. Evaluations show that DQ Scheduler outperforms the aforementioned anti-entropy protocol by up to 95.2% for the inter-domain routing task.

5.2 Related Work

5.2.1 Distributed SDN

Many research efforts are directed to the design of distributed SDN controller architecture. Specifically, OpenDaylight [51] and ONOS [24] are two state-of-the-art SDN control platforms proposed to realise logically centralised but physically distributed SDN architecture. In addition, controllers such as Devoflow [33] and Kandoo [34] are designed with their specific aims. However, most of these controller

architectures do not emphasise or justify detailed controller synchronisation protocols they employ.

5.2.2 Controller Synchronisations

Most existing works on controller synchronisation assume either strong or eventual consistency models [55], for which our work uses the latter. The authors in [53] show that certain network applications can rely on the eventual consistency to deliver acceptable performance. This work [56] shows how to avoid network anomalies such as forwarding loops and black holes under the eventual consistency assumption. Similar to our approach, the works [57, 58] propose dynamic adaptation of synchronisation rate among controllers. Compared to these works, DQ Scheduler is more versatile in that there is no assumptions on the network and the policy learning process is automated given any network conditions.

5.2.3 Reinforcement Learning in SDN

Some recent high-profile successes [59, 60] attract enormous interests in applying RL techniques to solve complicated decision making problems. In the context of SDN, our previous work in [61] apply RL-based algorithms to solve service placement problem on SDN switches. This work [62] also discusses the routing problem in SDN using RL techniques. However, the discussion is only limited to intra-domain routing under strong assumptions on the network topology. In addition, tabular settings are used in this work without generalisations. In contrast, as we shall demonstrate in this chapter, DQ-Scheduler does not have any assumptions on network structures or other parameters.

5.3 Problem Formulation

We formulate the controller synchronisation problem with inter-domain routing as an application of interest. We first describe the generalised routing path construction mechanism under distributed SDN with eventual consistency model (Section 5.3.1) and then introduce the performance metric (Section 5.3.2). Next, we discuss the synchronisation of SDN controllers and introduce its formal definition in Section 5.3.3. We then state in Section 5.3.5 the objective of the controller synchronisation problem. Finally, the problem is formulated as an MDP in Section 5.3.6.

5.3.1 Generalised Path Construction Mechanism in SDN

Under distributed SDN paradigm, inter-domain routing, like any other network task, is carried out by matching the packet's header with entries in switches' flow tables that store the forwarding rules installed by the controllers. Due to the flexibility and programmability of the SDN, there are potentially many ways in which routing can be conducted. In this section, we describe a simple routing path construction mechanism which is generalised based on principles of BGP-like protocol [63] in the Internet, and routing mechanisms employed by some state-of-the-art controllers such as the ONOS controller. Note that it is *not* our intention to design any routing mechanisms; we use this simple and representative mechanism as it takes advantage of the synchronised information among controllers for enhancing inter-domain routing. Specifically, the path construction mechanism consists of the following steps.

Step 1: The controller of the domain where the source node sits (*source controller* in the sequel) decides the sequence of domains that the packet will traverse between the source and the destination domains (called the *domain-wise path*), according to certain control objectives of the controller;

Step 2: Based on its view of the topologies of the domains on the domain-wise

path, the source controller constructs the path from the source node to the destination node that optimises the control objective;

Step 3: The source controller communicates the path construction decision to the involving domains' controllers and they install the forwarding rule on switches in the form of ingress and egress gateway IP addresses.

An illustrative example is presented in Fig. 5.1 to demonstrate how the routing mechanism works. The example shows the selected domain-wise path between the source node v_1 and the destination node v_2, in which domains $\mathcal{A}_1$–$\mathcal{A}_4$ are involved. The topology of these domains are the views of the source controller, which therefore constructs the path (red lines) that minimises the hop counts (other performance metrics can also be used; see Section 5.3.2 for details) between v_1 and v_2. Then the source controller instructs the controller of $\mathcal{A}_2$ and $\mathcal{A}_3$ that the packet whose destination is v_2 should egress their domains at node b and c, respectively. Note that such constructed path may be suboptimal, as the view is incomplete or out of date (see Section 5.3.3 for further explanations).

5.3.2 Performance Metric

To quantify the performance of the constructed routing path in a selected domain-wise path under the given inter-domain synchronisation status, we employ the APC, which is discussed and defined in Section 2.3, as the performance metric.

5.3.3 Synchronisation Among SDN Controllers

Under the eventual consistency model in distributed SDN, the quality of constructed routing paths is directly affected by the controller synchronisation levels. We use an example to demonstrate the benefits of controller synchronisation for path construction. In Fig. 5.1 and Fig. 5.2, suppose the source node v_1 in $\mathcal{A}_1$ sends packets to the destination node v_2 in $\mathcal{A}_4$. The topology in Fig. 5.1 represents $\mathcal{A}_1$'s controller's

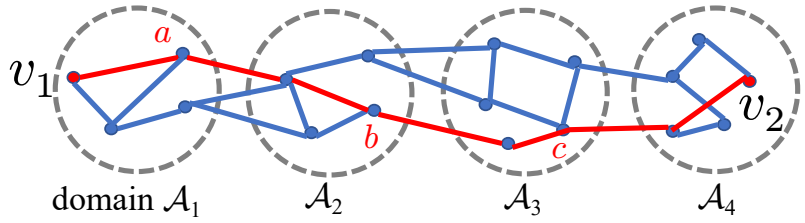


Figure 5.1: Controller $\mathcal{A}_1$'s network view.

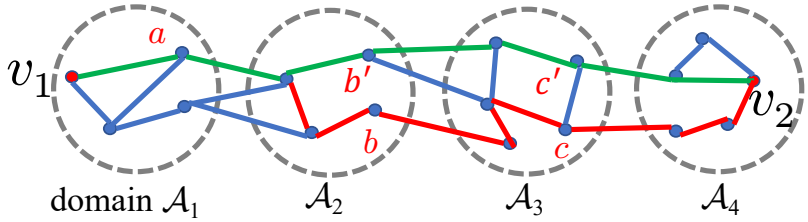


Figure 5.2: Actual network topology.

view of the network, which was obtained during synchronisations between $\mathcal{A}_1$ and $\mathcal{A}_2$–$\mathcal{A}_4$ in the past. However, due to the dynamicity of the networks, the actual topology evolves into the one in Fig. 5.2, which is not promptly synchronised to the source controller. As a result, the source controller, with the outdated view of the network, still uses the old flow table entries which direct packets sent to v_2 to gateways a, b, and c, respectively. In comparison, the source controller would select a shorter route (green lines) that involves b' and c' as egress gateways in domains $\mathcal{A}_2$, and $\mathcal{A}_3$, should it obtain the most up-to-date network topology through synchronisations. This example highlights the important role of *controller synchronisation* in dynamic networks, which is formally defined below.

Definition 12. Domain $\mathcal{A}_i$ is synchronised with domain $\mathcal{A}_j$ if and only if the SDN controller in $\mathcal{A}_i$ knows the minimum path cost between any two nodes in $\mathcal{A}_j$.

Furthermore, we also define the synchronisation budget which limits the amount of controller synchronisations.

Definition 13. The synchronisation budget of an SDN controller is defined as the maximum number of other controllers that it can synchronise with at any time slot.

5.3.4 Scenario

In this chapter, the controller synchronisation scenario we consider is generalised from the example discussed in Fig. 5.1 and Fig. 5.2. Specifically, we consider the inter-domain routing tasks between node pairs in the given source and destination domains. The routing path is established between the source/destination node pairs based on the path construction mechanism described in Section 5.3.1 using the source controller's views of other domains on the domain-wise path. Therefore, the source controller synchronises (Definition 17) with other domain controllers on the domain-wise path, subject to the given synchronisation budget for the source controller (Definition 18).

To model network dynamicity, the scenario considered employs a simple edge rewire model. Specifically, at each time slot, e_i new edges are added randomly between nodes in domain i before e_i existing edges are randomly selected and removed.

Note that we consider a time-slotted system for making the problem tractable. Specifically, at the start of the time slot, the source domain controller synchronises with other domain controllers according the synchronisation policy and budget. The synchronised information remain up-to-date during the current time slot, i.e., it is assumed that network topology changes (by the rewire model above) actuates right before controller synchronisation at each time slot.

5.3.5 Objective

Two questions motivate our definition of problem objective. First, how does the source controller make synchronisation decisions that most efficiently utilise the limited synchronisation budget? Second, how to maximise the benefit of synchronisation over time? With these questions in mind, we formally state the objective of the controller synchronisation problem.

Objective: In dynamic networks whose topologies evolve over time, given the controller synchronisation budget and for a set of source and destination nodes located in different domains that send/receive data packages (i.e., the scenario detailed in Section 5.3.4), how does the source controller synchronise with other controllers on the domain-wise path at each time slot, to maximise the benefit of controller synchronisation (i.e., to minimise the average APC) over a period of time?

Formally, let n and T denote the number of source/destination node pairs, and the time horizon (i.e., the number of time slots) considered for the inter-domain routing problem. Then, the goal of the controller synchronisation problem is stated as follows

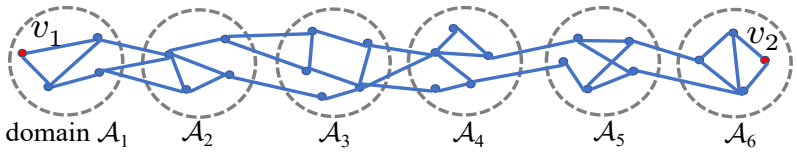
$$\begin{aligned} \min_{v_t} \quad & \frac{1}{nT}\mathbb{E}\Big[\sum_{t=1}^{T}\sum_{i=1}^{n} l_t(p_i)|v_t(b_t)\Big] \\ \text{s.t.} \quad & b_t \leq \beta_t \quad \text{for} \quad t = 1, 2, \cdots, T \end{aligned} \tag{5.1}$$

where $l_t(p_i)$ denotes the the APC for the path between node pair p_i at time slot t, and $v_t(b_t)$ represents the network view of the controller after synchronising with b_t other controllers at time slot t. Note that b_t is bounded by the synchronisation budget at t, which is denoted by β_t.

5.3.6 MDP Formulation

Markov Decision Process (MDP) [64] offers a mathematical framework for modelling serial decision-making problems. Here, we formulate the controller synchronisation problem as an MDP, in which 3-tuple $(\mathcal{S}, \mathcal{A}, R)$ is used to characterise it.

- $\mathcal{S}$ is the state space. In our problem, a state corresponds to the counts of time slots since the source controller was last synchronised with other controllers on the domain-wise path.
- $\mathcal{A}$ is the finite action space. An action w.r.t. a state is defined as the decision to



• State (s): the number of time slots since $\mathcal{A}_1$ last synchronized with the corresponding controllers

$\mathcal{A}_2$	$\mathcal{A}_3$	$\mathcal{A}_4$	$\mathcal{A}_5$	$\mathcal{A}_6$
5	10	15	5	35

• Action (a): the domain to synchronize with

$\mathcal{A}_2$	$\mathcal{A}_3$	$\mathcal{A}_4$	$\mathcal{A}_5$	$\mathcal{A}_6$
0	0	0	1	0

Figure 5.3: A state-action example for the MDP formulation.

synchronise with the selected domain(s), subject to the given synchronisation budget.

- R represents the immediate reward associated with state-action pairs, denoted by $R(s, a)$, where $s \in \mathcal{S}$ and $a \in \mathcal{A}$. $R(s, a)$ is calculated as the average reductions in APC associated with an (s, a) tuple.

The MDP formulation is demonstrated in an example in Fig. 5.3, where there are 6 domains on the domain-wise path between the source and destination nodes. The first entry in the state vector indicates that the last synchronisation between the source controller and the controller of $\mathcal{A}_2$ took place 5 time slots ago. The action vector consists of binary entries where 1 indicates that the source controller will synchronise with the corresponding domain at current time slot and 0 the opposite. The action vector in the above example indicates that under the synchronisation budget of 1, the source controller will synchronise with $\mathcal{A}_5$ only.

The *optimal action* at each state is defined as the action that yields the maximum long-term reward, which is defined as the discounted sum of the expected immediate reward of all future state-action pairs from the current state. The reward for the state-action pair Δt steps ahead of the current state is discounted by $\gamma^{\Delta t}$, where γ is

called the *discount factor* and $0 < \gamma < 1$. Here, γ trades off the importance between the current and the future reward. Therefore, starting from an initial state s_0, the problem is formulated to maximise the long-term accumulated reward expressed in the following Bellman equation by selecting a sequence of actions $\{a_t\}_{t=0}^{T}$:

$$V(s_0) = \mathbb{E}\Big[\sum_{t=0}^{T} \gamma^t R(s_t, a_t)|s_0\Big], \tag{5.2}$$

where s_t and a_t constitute the state-action pair at time t, and T is the time horizon of the synchronisation optimisation problem.

5.4 DQ Scheduler

To solve the formulated MDP, we use RL techniques to find the sequence of actions that maximise the Bellman equation in (5.2). For RL, imagine an agent who jumps from state to state in the formulated MDP by taking some actions associated with certain rewards. The agent's goal is to discover a sequence of state-action pairs, called a *policy*, that maximise the accumulated time-discounted rewards. By interacting with the MDP, the agent's experiences build up which finally lead to the discovery of the *optimal* policy. For this problem, one important aspect is how the agent memorizes its experiences. Traditionally, the storage of experiences in tabular fashion is used. However, this approach is impractical in many RL tasks because of the lack of generalisation for large state-action space. Indeed, the state-action space is enormous in our controller synchronisation problem. Consider the example in Fig. 5.3 assuming the time horizon is 300 time slots and the synchronisation budget is 1, there are as many as 300^5 states and 5 actions associated with each state. In light of this, function approximators [65] have been proposed, among which DNN [66] is a suitable candidate which finds its successes in many recent applications [59, 60]. Motivated by this, we therefore use DNN as the value function approximator in our DQ Sched-

uler, see Section 5.4.2 for details. Finally, we present the training algorithm for DQ Scheduler in Section 5.4.3.

5.4.1 Q-learning with Parameterised Value Function

Q-learning [67] is a classic RL algorithm with performance guarantees under certain conditions [68]. Q- learning uses the *Q-function* to estimate the quality of a state-action pair:

$$\mathcal{S} \times \mathcal{A} \to \mathbb{R}.$$

In particular, the optimal Q-function for a state-action pair in Q-placement is defined as:

$$Q^*(s,a) = \mathbb{E}[R(s,a) + \gamma \max_{a' \in A_{s'}} Q^*(s',a')], \tag{5.3}$$

where s' and a' is the state-action pair at the next time slot, $A_{s'}$ is the set of actions available at the next state s'.

Since we use DNN as the function approximator of the agent's Q-function $Q(s,a)$, it is parameterised by the set of adjustable parameters θ representing the weights of the DNN. The parameterised Q-function and optimal Q-function are denoted by $Q_\theta(s,a)$ and $Q^*_\theta(s,a)$, respectively. The *value iteration update* [67] of the Q-function is based on (5.3), which uses the best estimation of the future reward of the next state to update current Q-function, thus approximating the optimal $Q^*_\theta(s,a)$. During the update, θ is adjusted to reduce the gap between the estimated and the optimal values. In particular, the following loss function using the mean-squared error measurement is defined for adjusting θ:

$$L(\theta) = \mathbb{E}[\big(y - Q_\theta(s,a)\big)^2], \tag{5.4}$$

where

$$y = R(s,a) + \gamma \max_{a' \in A_{s'}} Q_\theta(s', a') \tag{5.5}$$

is the estimation of the maximum accumulated future reward.

Then, by differentiating $L(\theta)$ w.r.t. θ, we have the following gradient:

$$\nabla_\theta L(\theta) \;=\; -2\mathbb{E}[\big(R(s,a) \;+\; \gamma \max_{a' \in A_{s'}} Q_\theta(s', a') \;-\; Q_\theta(s,a)\big)\nabla_\theta Q_\theta(s,a)]. \tag{5.6}$$

Then, weights of the DNN are updated for the next iteration:

$$\theta \leftarrow \theta - \alpha \nabla_\theta L(\theta), \tag{5.7}$$

where α is the step size. Note that the gradient descent update iterations of θ is different from canonical supervised learning because the training target $y = R(s,a) + \gamma \max_{a' \in A_{s'}} Q_\theta(s', a')$ is generated by the same parameterised Q-function $Q_\theta(s,a)$ that is being trained. Therefore, to improve stability and performance of the training process, we improve the training algorithm in the following ways.

1) We maintain a delayed version of the Q-function, $Q_{\theta'}(s,a)$, for the estimation of the maximum next state reward, which was proposed in [59] to improve the stability of their DQN for playing Atari games. As such, the target function in (7.22) is updated to

$$y = R(s,a) + \gamma \max_{a' \in A_{s'}} Q_{\theta'}(s', a'). \tag{5.8}$$

The delayed Q-function is updated with the newest weights every C steps by setting $\theta' = \theta$.

2) To overcome the overestimation of action values, we implement *Double Q-learning* [69] to address the positive bias in estimation introduced when the maximum expected action values are instead approximated by the maximum action values in Q-learning. Specifically, we use the up-to-date Q-function $Q_\theta(s', a')$ to determine

$a'^* = \arg\max_{a'} Q_\theta(s', a')$, and the accumulated reward of the returned action a'^* is estimated by the delayed Q-function using (5.8).

3) We implement the "replay memory" [70] in which some of the agent's past experiences in (s, a, r) tuples are stored and maybe used more than once for training. In particular, a matrix $\mathcal{D}$ is created that can store up to N (s, a, r) tuples. At each training iteration where Q-learning update takes place, samples of experiences are pulled randomly from $\mathcal{D}$ for training.

5.4.2 The Design of the DDN

The parameterised Q-function is implemented by a *Multilayer Perceptron (MLP)* [71] consisting of input/output and three hidden layers. Let m denote the number of domains on the domain-wise path. The input to the MLP is of dimension $2(m-1) \times 1$. The first $m-1$ entries store the state of the MDP, and the rest $m-1$ binary entries store the action. The output of the MLP is the maximum predicted accumulated time-discounted reward given the state-action input. The three hidden layers consist of 128, 64, and 32 hidden neurons, respectively. The MLP is realised using Keras [72] model with TensorFlow [73], in which Adam is chosen as the optimiser and Rectified Linear Unit (ReLU) [74] is employed as activation functions for all neurons except for the output layer.

5.4.3 The Training Algorithm

To train the DNN that represents the parameterised Q-function, we need to first initialize the matrix $\mathcal{D}$, i.e., the agent's "reply memory". This is different from traditional online Q-learning where the update of Q-matrix relies only on the current (s, a, r) tuple. Instead, history data are used for the training of the DNN for stability reasons. In particular, $\mathcal{D}$ is initially filled up with several (s, a, r) tuples for training. As time proceeds, more (s, a, r) tuples are generated and recorded in matrix $\mathcal{D}$ for

training. Since we limit the number of entries of $\mathcal{D}$ to N, old (s, a, r) tuples are gradually replaced by new entries; see Section 5.5 for the simulation settings that generate the training data. There are many ways in which actions can be selected. Traditionally, ε-greedy algorithm [75] is used in Q-learning; there are also new exploration strategies proposed which are tailored for the DNN settings, such as bootstrapped DQN [76] and UCB [77]. According to [78], the exploration strategy that generates all state-action pairs uniformly at random is better for training. Therefore, our training algorithm takes in data generated by random exploration. The training process is summarised in Algorithm 1. After the training algorithm terminates, the returned parameterised Q-function will be able to estimate the best action to take at each state, thus approximating the optimal.

5.5 Evaluation

In this section we present the performance evaluation of the proposed DQ Scheduler comparing to other two default controller synchronisation schemes. Specifically, we first introduce the network and its dynamicity model used for building simulation networks in Section 5.5.1. Then, the settings and datasets used are described in Section 5.5.2. Finally, we present the evaluation results and analysis in Section 5.5.3.

5.5.1 Network and Dynamicty Model

Network Model: The topology of domain i with n_i nodes is modeled as an undirected graph, where n_i nodes are connected following a given *intra-domain degree distribution*, i.e., the distribution of the number of neighbouring nodes of an arbitrary node. Then for any two neighbouring domains $\mathcal{A}_i$ and $\mathcal{A}_j$, we (i) randomly select two nodes w_1 from $\mathcal{A}_i$ and w_2 from $\mathcal{A}_j$ and connect these two nodes if link w_1w_2 does not exist, and (ii) repeat such link construction process between $\mathcal{A}_i$ and $\mathcal{A}_j$ $\beta_{i,j}$

Algorithm 1: Training algorithm for DQ Scheduler

input : DNN model settings; distributed SDN settings; simulation program for generating rewards; delay C of the delayed Q-function.
output: Trained parameterised Q-function.

Initialize the Q-function $Q_\theta(s, a)$ by instantiating the DNN with random initial weights and biases settings;
Initialize matrix $\mathcal{D}$ with history (s, a, r) tuples;
Initialize the delayed Q-function $Q_{\theta'}(s, a) = Q_\theta(s, a)$;
Set initial state s_0; $t = 0$;
while $t \leq T$ **do**
  foreach *time instant* t **do**
    Select an action a_t randomly;
    Pass on the (s_t, a_t) to the simulation program and get return r_t;
    Store (s_t, a_t, r_t) in $\mathcal{D}$;
    Pull random minibatch $\mathcal{D}_t$ of (s_i, a_i, r_i) from $\mathcal{D}$;
    foreach (s, a, r) *in* $\mathcal{D}_t$ **do**
      $a^*_{j+1} = \arg\max_{a_{j+1}} Q_\theta(s_{j+1}, a_{j+1})$;
      $y_j = r_j + \gamma Q_{\theta'}(s_{j+1}, a^*_{j+1})$;
      Calculate the gradient:
      $\nabla_\theta L(\theta) = -2(y_j - Q_\theta(s_j, a_j))\nabla_\theta Q_\theta(s_j, a_j)$;
      Update weights: $\theta \leftarrow \theta - \alpha\nabla_\theta L(\theta)$;
    end
    if $t \mod C = 0$ **then**
      $Q_{\theta'}(s, a) = Q_\theta(s, a)$.
    end
  end
end

times. By this link construction process, network topology $\mathcal{G} = (V, E)$ is therefore formed (V/E: set of nodes/links in $\mathcal{G}$, $|V| = \sum_{i=1}^{q} n_i$, where q is the number of domains).

Dynamicity of the network: The dynamic pattern of intra-domain topologies is modelled by the rewiring model described in Section 5.3.4. If the random edge rewire procedure results in a fragmented network topology, we randomly add the minimum number of edges to connect all components to make it a connected graph again. The edge weights of all newly added edges are generated by the given edge weight distribution. It should be noted that our DQ Scheduler does not have any requirement on the dynamicity model of the network, the rewire model we propose

only serves as a simple and representative example.

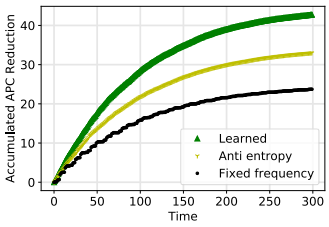


Figure 5.4: Accumulated APC reduction, $m = 6$.

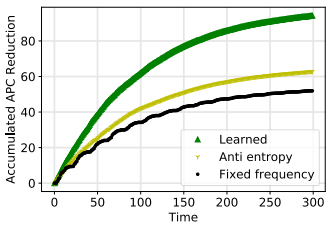


Figure 5.5: Accumulated APC reduction, $m = 10$.

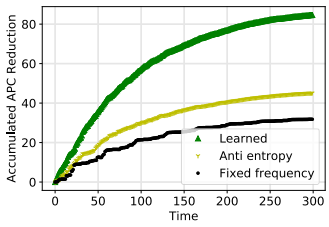


Figure 5.6: Accumulated APC reduction, $m = 12$.

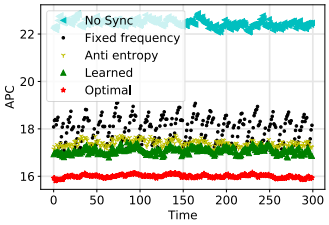


Figure 5.7: Immediate APC, $m = 6$.

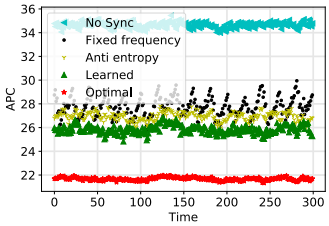


Figure 5.8: Immediate APC, $m = 10$.

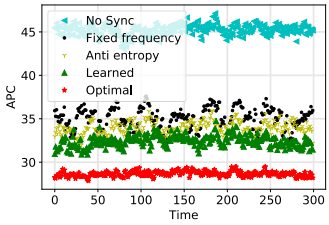


Figure 5.9: Immediate APC, $m = 12$.

5.5.2 Network Settings

According to the path construction mechanism described in Section 5.3.1, the domain-wise paths extracted w.r.t. source/destination node pairs are always in a linear fashion such as the example in Fig. 5.3. Let m be the number of domains on the domain-wise path with the indices of domains sequentially labelled from 1 (source domain) to m (destination domain). Our evaluation considers three scenarios where $m = 6$, $m = 10$, and $m = 12$. The degree distributions used is extracted from the RocketFuel Project [48], where we use data from"AS1239". The weights of edges are randomly drawn from the set $\{1, 2, 5, 8, 10, 12\}$ with the corresponding probability being $10\%, 10\%, 10\%, 30\%, 20\%,$ and 20%, respectively. In addition, the synchronisation budget is set to 1 for all evaluations. This limitation is in place to allow simple policy derivation from the learned valued function. In particular, DQ Scheduler derives synchronisation policy by calculating the returns of all possible actions with the given state, and it selects the action with the highest return estima-

Table 5.1: Evaluation Parameters

m	$\mathcal{N}_m$	$\mathcal{E}_m$	$\mathcal{B}_m$
6	$[100, 300, 550, 150, 210, 420]$	$[5, 1, 40, 60, 2, 120]$	$[5, 20, 8, 20, 12]$
10	$[100, 300, 550, 150, 210, 420, 380, 520, 120, 340]$	$[5, 1, 160, 1, 180, 150, 40, 150, 1, 40]$	$[5, 30, 50, 20, 15, 20, 20, 30, 10]$
12	$[130, 50, 550, 150, 80, 420, 380, 330, 250, 150, 80, 100]$	$[10, 1, 100, 1, 1, 120, 1, 180, 130, 1, 1, 1]$	$[12, 30, 25, 5, 15, 10, 20, 5, 8, 15, 20]$

tion. Clearly , the increase of synchronisation budget results in the combinatorial increase of possible state action combinations. In Chapter 6, we address this limitation by customising the design of the DNN function approximator. In addtion, let $\mathcal{N}_m = [n_1, n_2, \ldots, n_m]$, and $\mathcal{E}_m = [e_1, e_2, \ldots, e_m]$ be the vectors of the number of nodes, and the number of edge rewires at each time slot for the m domains on the domain-wise path, respectively. In addition, $\mathcal{B}_m = [\beta_{1,2}, \beta_{2,3}, \ldots, \beta_{m-1,m}]$ is the vector of the gateway connection parameters between all pairs of directly connected domains on the domain-wise path with m domains. The parameters used for three evaluation scenarios are listed in Table 5.1. Note that DQ Scheduler *does not* have any requirements for aforementioned parameters.

The anti-entropy [54] and fixed-frequency synchronisation [57] schemes are employed as benchmarks for evaluating the performance of DQ Scheduler. The core of anti-entropy protocol, which is implemented in the ONOS controller, is based on a simple gossip algorithm that each controller randomly chooses another controller to synchronise. In comparison, the fixed-frequency algorithms synchronise controller pairs at constant rates, which may vary for different controller pairs subject to various objectives. For this evaluation, all controllers synchronise at the same rate.

5.5.3 Evaluation Results

The evaluation results of the DQ Scheduler for three scenarios are presented in Fig. 5.4-5.9, where Fig. 5.4–Fig. 5.6 show the performance in terms of the objective stated in Section 5.3.5. Fig. 5.7–Fig. 5.9 show the APC of packets delivered under three controller synchronisation schemes.

1) Superiority of DQ Scheduler for long-term routing quality: Recall that our objective is aimed at overall routing quality over a period of time, i.e., to maximise $V = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t)\right]$. The evaluation results in Fig. 5.4–Fig. 5.6 confirm the superiority of DQ Scheduler in achieving this goal. In particular, during the testing period of 300 time slots, DQ scheduler outperforms the anti-entropy algorithm by 31.2%, 58.3%, and 95.2%; the algorithm with constant synchronisation rate by 90.9%, 90%, and 173.3%, for three scenarios, respectively.

2) Superiority of DQ Scheduler for immediate routing quality: Although DQ Scheduler is trained to maximise the accumulated APC reduction over time, surprisingly, its synchronisation decisions also lead to the lowest APC in real-time among three algorithms tested, as shown in Fig. 5.7–Fig. 5.9. This means that the DQ scheduler optimises the accumulated APC reduction in a way that the immediate and long-term performance are balanced, since there is not a period in which the immediate performance is worsened for better future performance according to these results. In these evaluations, the APCs are also compared to the “optimal" case where the source domain is always synchronised with all other domains on the domain-wise path,s and to the “worst" case when there is no synchronisation (“no sync") between any controllers.

3) Other findings: Compared to the other benchmark algorithms, DQ Scheduler’s performance is more stable when the domain-wise path involves more domains. In contrast, two benchmark algorithms’ performance first improves and then becomes worse when the number of domains on the domain-wise path increases from 6 to 10,

and then to 12. In addition, we realise that the performance degradation of no controller synchronisation is more concerning when there are more domains involved on the domain-wise path, as “no sync" performance are worsened by 37.5%, 59.1%, and 61% in three scenarios, comparing to the optimal cases. This highlights the important role of controller synchronisations.

5.6 Limitations of the DQ Scheduler

The main limitations of the DQ Scheduler stem from the simple implementation of the DNN approximator for the value function. Recall that since the inputs to the value function are state-action pairs, the size of input vector space grows combinatorially in both the state and action space sizes. In particular, let n and t denote the number of domains on the domain-wise path and the time step of the problem. Then, the number of state-action pairs can be up to $(2t)^n$ at time t. For DRL problems with relatively small action spaces, there exists value function-based DRL solutions, which render reasonable performance. For example, Google's DQN [59] can handle 8 canonical actions. The DNN-based value function approximator employed by DQ Scheduler is realised by ordinary MLP without customised design. Therefore, several compromises are made to reduce the state-action space size, in order to ensure the value function can be properly approximated. These limitations are discussed as follows.

DQ Scheduler maintains a coarse-grained description of network status information. Specifically, recall that the basic piece of network status information defined in DQ Scheduler is the time of last synchronisation between two SDN controllers, which does not account for the status information of specific networking elements in SDN domains. The lack of the ability to support fine-grained synchronisation of status information prevents its application to a wider range of control tasks. Therefore,

DQ Scheduler is designed to only assist inter-domain routing tasks in this Chapter.

In summary, the DL architecture employed by the DQ Scheduler is simplistic for faster prototyping, which is also the main source of its limitations.

CHAPTER 6

MACS: DRL-based Controller Synchronisation for Fine-grained Control

6.1 Introduction

In the previous chapter, we present DQ Scheduler, a simple controller synchronisation scheduler that generates synchronisation schedules for SDN controllers, which lead to enhanced inter-domain routing performance. However, there are several limitations associated with DQ Scheduler, which are discussed in Section 5.6. To further support SDN applications that require the capability for finer-grained controller synchronisation policy, we introduce MACS, which is a much improved DRL-based controller synchronisation policy generation framework compared to DQ Scheduler. The main advantage of MACS lies in its ability to produce finer-grained controller synchronisation policy, which is enabled by the customised DRL-network designed to handle larger state-action space for the controller synchronisation-induced RL problem.

To this end, this chapter is organised as follows. Section 6.2 describes the SDN application we consider in this chapter, which requires finer-grained controller synchronisation compared to the inter-domain routing application with DQ Scheduler. Section 7.3 formally formulates the controller synchronisation problem and states

the objective. Then, MACS and its design and training algorithm are detailed in Section 6.4. Finally, we conduct evaluations on MACS and present evaluation results in Section 6.5.

6.2 System Description

6.2.1 Controller Synchronisation Application

To materialise potential performance gains controller synchronisation can bring under distributed SDN, we focus on an application where find-grained information about communication and computation resources are essential for its operation. Therefore, we choose *service path construction* as the application of interest. In the Network-as-a-Service (NaaS) SDN environment, QoS-aware service path construction is a crucial problem in the context where network services are virtualised in servers [79]. Specifically, we investigate the problem where several network services, e.g., wireless access admission, firewall, etc., are installed on servers across all SDN domains. Requests for services are submitted by users to domain controllers, who construct service paths for requests submitted, based on their network views. Note that the process of finding a service path is an anycast problem [80], as a service can have multiple installations in different domains. In order to calculate the best service path w.r.t. the given performance metric, domain controllers rely on up-to-date information about other domains (e.g., traffic levels, network delay, available computation and services, etc.) gained through controller synchronisation.

6.2.2 Network Model

We formulate the distributed SDN network as a directed graph, where m vertices are connected via directed links. Let the graph representing the SDN network be

denoted by $\mathcal{G} = (V, E)$ (V/E: set of vertices/edges in $\mathcal{G}$, $|V| = m$), which is referred to as the *domain-wise topology*. The existence of two edges in $e_{1,2}, e_{2,1} \in E$ connecting two vertices $v_1, v_2 \in V$ in the domain-wise topology implies that the two network domains corresponding to v_1 and v_2 are connected. Then, we further associate weights for the directed inter-domain links, which represent *gateway delays* (see Section 6.3.1). We use $\mathcal{L}_i$ to denote the set of outgoing edge weights from the vertex of domain $\mathcal{A}_i$ in the network. For instance, let $\mathcal{A}_1$ and $\mathcal{A}_2$ denote the network domains corresponding to vertices v_1 and v_2 in the graph; then the weight of edge $e_{1,2}$, denoted by $l_{1,2} \in \mathcal{L}_1$, represents the estimation of latency a packet going thorough domain $\mathcal{A}_1$ and entering domain $\mathcal{A}_2$ should expect to experience in $\mathcal{A}_1$ (i.e., the *gateway delay*); similarly, $l_{2,1} \in \mathcal{L}_2$ is the latency a packet going thorough domain $\mathcal{A}_2$ and entering domain $\mathcal{A}_1$ would experience in $\mathcal{A}_2$.

Let $\mathcal{C}$ be the set of all installed services on servers located across different domains in the SDN network. Let $c_j^{(i)}$ denote service i installed in domain $\mathcal{A}_j$. Note that we do not differentiate two identical services installed in the same domain. Then, $\mathcal{D}_j$ is the set of server delays of all service installations in domain $\mathcal{A}_j$, and $d_j^{(i)} \in \mathcal{D}_j$ denotes the waiting time before a request for service i starts being processed in domain j (i.e., the *server delay*, see Section 6.3.1).

Remark: Note that due to network dynamicity, e.g., traffic levels, user demand for service patterns, etc., the values of $l_{i,j}$ and $d_j^{(i)}$ are time-varying. With the SDN settings described in this section, domain controllers always have the up-to-date status views of network elements residing in their domains; in other words, the controller of domain $\mathcal{A}_i$ always knows the newest $\mathcal{L}_i$ and $\mathcal{D}_i$. The controller of domain $\mathcal{A}_i$ relies on synchronisations with other domain controllers (see Section 6.3.2) to learn up-to-date $\mathcal{L}_j (j \neq i)$ and $\mathcal{D}_j (j \neq i)$.

6.3 Problem Formulation

In this section, we first define the performance metric, controller synchronisations and the synchronisation budget. Then, we discuss the service path construction mechanism employed, which is followed by the MDP formulation of our controller synchronisation problem.

6.3.1 Performance Metric

Once the user submits a service request, the sooner the request gets served, the better. Therefore, the gap in time between the submission of a service request and the server starting processing the request is a natural performance metric of qualities of constructed service paths, which we call the *request latency*.

Request latencies consist of two parts: transit latency and the waiting time at the server. Many factors, such as the link congestion levels, the number of hops, may contribute to the overall transit latency for the inter-domain routing of a service request from the user domain to the domain of the chosen server. For easier modelling, we use the delays incurred at egress gateway routers of SDN domains (referred to as *gateway delays*), which are usually the bottlenecks [81], to abstract the transit latency incurred traversing through SDN domains. If a service request is submitted to a server located in the same domain as the user, transit latency incurred is assumed to be negligible.

On the other hand, delays incurred at the server are the waiting times in the server before available computation resources can be assigned for processing the submitted requests, which we refer to as the *server delay*. Based on the network model described in Section 6.2.2, $l_{i,j}$ and $d_j^{(i)}$ correspond to gateway and server delays, respectively.

Therefore, we define the performance metric w.r.t. a constructed service path

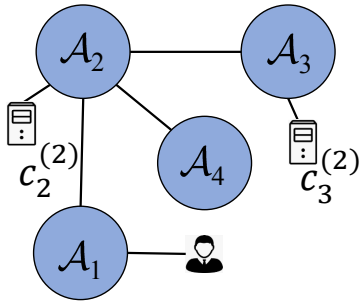


Figure 6.1: A service path construction example.

as the accumulated gateway delays en route the service path and server delay at the chosen server, which is referred to as *request latency* in the sequence.

6.3.2 Synchronisation Among SDN Controllers

W.r.t. the application of interest introduced in Section 6.2.1, controller synchronisation levels directly affect the quality of constructed service paths. We use an example to demonstrate this. In Fig.6.1, suppose the user located in domain $\mathcal{A}_1$ submits a service request for service 2. The domain controller for $\mathcal{A}_1$ is aware that the request can be forwarded to the server hosting $c_2^{(2)}$ in domain $\mathcal{A}_2$ or the server hosting $c_3^{(2)}$ in domain $\mathcal{A}_3$. The domain controller uses its view of the network conditions in domains $\mathcal{A}_2$ and $\mathcal{A}_3$ to decide the service path which it thinks would minimise the request latency before the request can be served. However, the domain controller's view could be stale. Specifically, domain $\mathcal{A}_1$ controller's view of the gateway delay between $\mathcal{A}_1 - \mathcal{A}_2$, $\mathcal{A}_2 - \mathcal{A}_3$ are 2 and 1, respectively. $\mathcal{A}_1$'s view of the gateway delay between $\mathcal{A}_1 - \mathcal{A}_2$ is always accurate and up-to-date, since the gateway router is within domain $\mathcal{A}_1$; while the up-to-date gateway delay between $\mathcal{A}_2 - \mathcal{A}_3$ is actually 3. On the other hand, $\mathcal{A}_1$'s view and the up-to-date server delays of the servers hosting $c_2^{(2)}$ and $c_3^{(2)}$ are 4 and 2, 1 and 3, respectively. Therefore, the domain controller of $\mathcal{A}_1$ would forward the service request to domain $\mathcal{A}_3$ rather than $\mathcal{A}_2$ since the request latencies are estimated to be $2 + 1 + 2 = 5$ in the former and $2 + 4 = 6$ in the latter based on

its view of the network. However, if $\mathcal{A}_1$'s domain controller holds up-to-date view of the network condition in domains $\mathcal{A}_2$ and $\mathcal{A}_3$, it would send the request to domain $\mathcal{A}_2$ instead.

This example highlights the important role of controller synchronisation that distributes up-to-date network information among domain controllers. Here, we formally define controller synchronisation w.r.t. our service path construction problem in this subsection. First, we define the unit that quantises the synchronisable domain information.

Definition 14. The gateway delay between a pair of domains, or the server delay of a service is referred to as a Basic Information of synchronisation (BIS).

A BIS corresponds to the most fundamental piece of information that can be synchronised to domain controllers. Note that in existing distributed controller implementations such as ONOS, when two controllers synchronise, they exchange their entire state information. In this thesis, we propose a more fine-grained synchronisation policy where only selected state information is exchanged. According to the above definition, an SDN network $\mathcal{G}$, with the set of installed service $\mathcal{C}$, has $N = |E| + |\mathcal{C}|$ number of BISes in total (see Section 6.2.2). With the concept of BIS, we then formally define controller synchronisation in the following definition.

Definition 15. Controller synchronisation is the process of domain controllers broadcasting/receiving up-to-date BISes originated in their domains/received from other domain controllers.

Definition 17 implies that controllers synchronise with each other in the way that selected up-to-date BIS(es) are broadcasted to all domain controllers via control plane messages. Then, all domain controllers update their network views by incorporating the received up-to-date BIS(es) from other controllers and BIS changes reported in their own domains.

As discussed in previous chapters, frequent dissemination of inter-controller synchronisation messages introduce prohibitively large overheads [18, 19]. Moreover, excessive status updates can potentially lead to network performance degradation caused by instability [82]. Indeed, we show in Section 6.5.3 that excessive synchronisations cause performance deterioration for MACS. Therefore, the number of synchronisation messages that can be exchanged at a time is limited, for which we introduce the *synchronisation budget*.

Definition 16. The synchronisation budget at a time is the maximum number of BISes that can be broadcasted simultaneously.

Remark: Note that in addition to the synchronisable BIS, all domain controllers always have knowledge of the correct domain-wise topology (without edge weights) and the available services in every other domains.

6.3.3 Service Path Construction using BIS Information

Due to the flexibility and programmability of the SDN, there are potentially many ways in which domain controllers calculate service paths. In this section, we describe a simple service path construction mechanism that uses BISes and aims at minimising the overall request latency. Note that in this thesis, it is *not* our intention to design any new such mechanisms; we use this simple and representative mechanism for the sake of problem formulation. Specifically, the service path construction mechanism consists of the following steps.

Step 1: The domain controller that receives a service request (*source controller* in the sequence) calculates the minimum accumulated gateway delay(s) of service path(s) leading to all possible server(s) that host the requested service (recall that the process of finding a server is an anycast problem).

Step 2: The domain controller calculates the request latency (latencies) for the

requested service for all potential service path(s) in ***Step 1*** by combining their accumulated gateway delays and the corresponding server delays. The source controller chooses the service path that incurs the lowest request latency based on its calculation results.

Step 3: The forwarding rule of the service request is installed on involved SDN switches in forms of flow table entries.

Applying the above service path construction mechanism, only the source controller decides the domain-wise service path based on its view of the network status. Other domain controllers are deliberately left out in calculating service paths to avoid forwarding anomalies, e.g., routing loops and black holes, which could arise if different domain controllers with heterogeneous network views attempt to independently calculate service path for a packet transiting through their domains.

6.3.4 The Objective of Controller Synchronisation Policy

Two questions motivate our definition of the objective of controller synchronisation. First, how does the central controller develop the synchronisation policy that maximises the performance metric, given the limited synchronisation budget? Second, since a synchronisation decision has lasting effects, how does the synchronisation policy maximise the performance enhancement of controller synchronisation over time? Here, a *synchronisation policy* refers to a series of synchronisation decisions (i.e., which up-to-date BIS(es) to broadcast, subject to the available synchronisation budget) over a period of time. With this in mind, we state the objective below.

Objective: In dynamic networks where gateway and server delays are time-varying, given the controller synchronisation budget, how do domain controllers synchronise with each other by broadcasting up-to-date BISes, to maximise the performance improvements brought by controller synchronisations (i.e., reductions in average request latency due to the availability of accurate BISes via synchronisa-

tions) over a period of time?

6.3.5 Markov Decision Process (MDP) Formulation

Our controller synchronisation policy can be modelled as an MDP with the 3-tuple $(\mathcal{S}, \mathcal{A}, R)$ as follows.

- $\mathcal{S}$ is the finite state space. In particular, a state corresponds to the collection of the respective counts of time slots elapsed since the last broadcasts of up-to-date values of each BIS. As such, a state of the MDP represents the *staleness* of status information of network components. The size of a state is N, which is the total number of BISes.
- $\mathcal{A}$ is the finite action space. An action w.r.t. a state is defined as whether each up-to-date BIS should be broadcasted (indicated by 1) or not (indicated by 0), i.e., $\mathcal{A} \in \{1, 0\}^N$. As such, the size of an action is also N.
- R represents the immediate reward associated with state-action pairs, denoted by $R(\mathbf{s}, \mathbf{a})$, where $\mathbf{s} \in \mathcal{S}$ and $\mathbf{a} \in \mathcal{A}$ are the state and action vectors. $R(\mathbf{s}, \mathbf{a})$ is calculated as the reductions in average request latency of all service requests in the network after taking action $\mathbf{a}$ in state $\mathbf{s}$.

With this MDP formulation, $\mathcal{S}$ and R are collected by domain controllers from data planes through SDN's northbound interface [83] and are supplied to the central controller.

Fig. 6.2 demonstrates the $(\mathcal{S}, \mathcal{A})$ pair in the formulated MDP in a simple example. The first entry in the state vector indicates that the last synchronisation broadcast of the up-to-date value of gateway delay from $\mathcal{A}_1$ to $\mathcal{A}_2$ took place 5 time slots ago. An action consisting of binary entries indicates that the up-to-date value of gateway delay from $\mathcal{A}_2$ to $\mathcal{A}_1$ and the server delay of service 1 installed in domain $\mathcal{A}_2$ are to be broadcasted to all domain controllers at the current time slot.

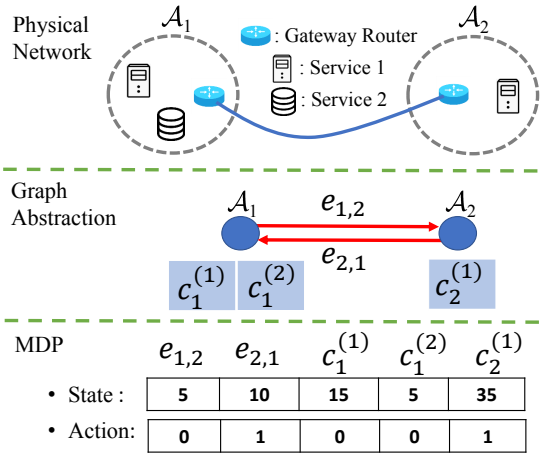


Figure 6.2: An SDN network with 2 domains $\mathcal{A}_1$ and $\mathcal{A}_2$, where service 1 is available in both domains and service 2 is only available in $\mathcal{A}_1$. Two domains are connected by a pair of gateway routers. There are in total 5 BISes in the network.

The *optimal action* at each state is defined as the action that yields the maximum long-term reward, which reflects the fact that the synchronisation decision at a time slot has lasting effects. In particular, the long-term reward is defined as the discounted sum of the expected immediate reward of all future state-action pairs from the current state. The reward for the state-action pair Δt steps ahead of the current state is discounted by $\gamma^{\Delta t}$, where γ is called the *discount factor* and $0 < \gamma < 1$. Here, γ trades off the importance between the current and the future reward. Therefore, starting from an initial state s_0, the problem is formulated as finding a policy π (i.e., the selection of a sequence of actions $\{\mathbf{a}_t\}_{t=0}^{T}$) such that the long-term accumulated reward expressed in the Bellman equation below is maximised

$$V^{\pi}(\mathbf{s}_0) = \mathbb{E}_{\pi}\left[\sum_{t=1}^{T} \gamma^t R(\mathbf{s}_t, \mathbf{a}_t) | \mathbf{s}_0\right], \tag{6.1}$$

where $\mathbf{s}_t$ and $\mathbf{a}_t$ are the state-action pair at time t, and T is the total time horizon of the problem.

Time Scale of the MDP: The basic unit of time in the defined MDP include a series of events, which are jointly referred to as a *synchronisation time slot* (*time slot*

for short) in the sequence. Specifically, synchronisation broadcastings take place at the start of a time slot. After all domain controllers finish broadcasting and receiving up-to-date BIS, they recalculate service paths for requests originated from their respective domains, based on the updated network views. Then, actual service latencies en route for new service paths are recorded for the calculation of R. For modelling tractability, we assume that newly broadcasted BIS values remain accurate until actual service latencies are recorded. This is true in practical SDN networks where periodical synchronisation means that the collection of latest network information takes place at certain time. The collected information is considered to be “up-to-date" for a short while, as “real time" is a relative concept. It should be stressed that the concept of “times slot” in the formulated MDP is very different to the performance metric, i.e., service latency. The former is defined as a full cycle of controller synchronisation after which the MDP shifts from one state to another state. In contrast, the latter, which is coupled with time, is the actual length of a request waiting time. Since domain controllers need to record the actual service latency before the end of an MDP time slot, the length of a time slot is determined by the latency incurred en route the most delayed service path constructed at the beginning of the time slot.

Remark: Note that our MDP formulation is not only specific to the service path construction application defined in Section 6.2.1. In essence, the definition of the state space here represents a staleness measure of status information about various network components in the distributed SDN. In our problem, these components are gateway routers and servers. For other controller synchronisation problems, as long as networking elements can be itemised to fit in the state definition, such problems can be modelled by the formulated MDP.

6.4 The MACS

To solve the formulated MDP, we use RL techniques to find the sequence of actions that maximises the Bellman equation in (6.1). For RL, imagine an agent who jumps from state to state in the formulated MDP by taking some actions associated with certain rewards. The agent's goal is to discover a sequence of state-action pairs, i.e., a *policy*, that maximises the accumulated time-discounted rewards. By interacting with the environment modelled by the MDP, the agent's experiences build up through "trial-and-error" where good decisions are positively enforced by positive rewards, and bad ones the opposite.

During training, the most important aspect is how the agent generalises and memorises what it has learned. Traditionally, the agent's estimations of future reward following state-action pairs are kept in tabular fashion. However, this approach soon becomes impractical in most RL tasks because of large state-action space sizes. Indeed, the state-action space is enormous in our controller synchronisation problem. Consider a scenario with N number of BIS and a time horizon of T time slots, then there are as many as T^N states and 2^N actions associated with each state. In light of this, function approximators [65] have been proposed to approximate the Q-function, which represents the agent past experiences as it estimates the potential value of a given state-action pair (see Section 6.4.3). Among these approximators, DNN [66] stands out due to its exceptional ability in capturing latent and complicated relationships from input data. Therefore, we also employ DNN in MACS to help the agent make sense of and generalise past experiences.

In this section, we first discuss design challenges and the MACS architecture in Section 6.4.1 and Section 6.4.2, respectively. Then, mathematical details of how policies are estimated and the weights update process for the DNN in MACS are discussed in Section 6.4.3. Finally, we present the training algorithm for MACS in Section 6.4.5.

6.4.1 Challenges

From our experiences leveraging several RL techniques to solve the formulated MDP, we identify some non-trivial challenges arising mainly from the two following aspects. First, the problem is a discrete control problem. This prevents the application of a number of well-established and relatively mature actor-critic approaches [84] based on the policy gradient theorem [85]. For example, DeepMind's recent work on the deep deterministic policy gradient (DDPG) agent [86] is the state-of-the-art for continuous control problems. Second, our formulated MDP has a high-dimensional state-action space. In particular, there are up to 2^N number of possible actions for any state. Thus, the size of the action space increases exponentially with the number of BISes in the network. It has been shown that such large action space is very difficult to explore and generalise from [86]. Indeed, classic RL techniques [67] and their variations [87], which work well in scenarios with relatively small discrete action spaces, are not suitable solutions for our problem.

6.4.2 The MACS Architecture

Considering the challenges discussed in the previous section, we need a DRL solution that is designed for discrete problems and can perform well in the presence of enormous state-action spaces. To this end, we build our learning architecture based on proposals in [88, 89], which have design features suitable for the nature of our formulated MDP.

DeepMind's dueling network architecture [88] explicitly separates the training for estimations of state-values and the advantages for individual actions, i.e., these values can be obtained separately. This is in contrast to most existing DRL architectures where the output of the DNN is conventionally a single value, i.e., the estimated Q-value for the input state-action pair. The dueling network architecture is particu-

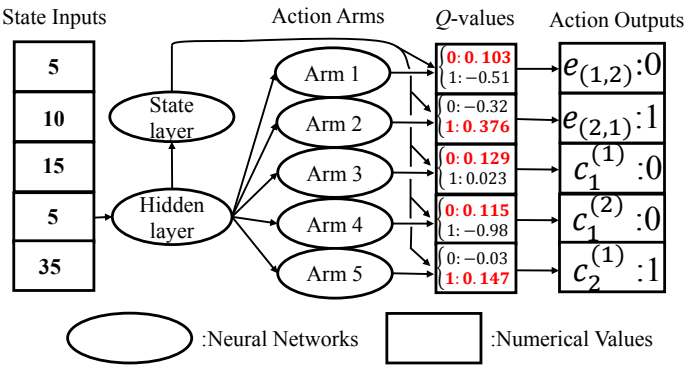


Figure 6.3: The MACS network based on the example in Fig. 6.2. State inputs are first fed to the hidden layer, which is shared by the state layer and all action arms. The state layer is responsible for estimating the shared state value, whereas all action arms are responsible for estimating advantages of their 2 sub-actions.

larly helpful in situations where there are many similar-valued actions.

The action branching architecture (ABA) [89] takes the dueling network a step further by categorising all actions as belonging to an *action dimension*. Moreover, a separate action advantage estimator is assigned to each action dimension (referred to as an *action arm* in the neural network) to estimate the advantage of all actions belonging to the action dimension (referred to as *sub-actions*). Under such arrangements, the Q-value estimation for each action is obtained by combining (i) the state value estimation, which is shared by all possible actions given the state; and (ii) the sub-action advantage estimated by the assigned action arm. A key characteristic of the action branching architecture is that a degree of freedom is given to each action dimension by dedicating a separate arm in the network for advantage estimations of sub-actions belonging to that action dimension. This design greatly improves learning efficiency and reduces complexity which arises from the combinatorial increase of the total number of action dimensions. Therefore, the design principles in the ABA are suitable for approximating the Q-function in our formulated MDP, as they address the challenges discussed in Section 6.4.1. In the following, we use the example in Fig. 6.2 to demonstrate how these design principles work in MACS.

The diagram of the DNN constructed for MACS shown in Fig. 6.3 is based on the example scenario in Fig. 6.2, where there are 5 BISes. In Fig. 6.3, there are 5 action arms, each corresponds to a BIS. Moreover, there are two sub-actions in every action dimension, i.e., 0: not to broadcast the corresponding up-to-date BIS value; and 1: to broadcast the corresponding up-to-date BIS value. An action arm outputs the estimation of action advantages of the two sub-actions under that action dimension. Furthermore, there is a separate state layer which outputs the estimation of the state value given the state inputs. Both the state layer and all semi-independent action arms are preceded by the input layer, which is designed for coordination. Based on the estimated Q-values, which are obtained by combining the estimated state value (output of the state layer) and action advantages (outputs of action arms), the sub-action with the highest Q-value is selected for each action arm. Finally, the outputs of all action arms concatenate and form the chosen action w.r.t. the state input.

In the following section, we give further details on how state values, action advantages, and Q-values are calculated and their relationships in MACS.

6.4.3 Design Details of MACS

Q-function, which originates from the classic *Q-learning* algorithm [67], is commonly used to estimate the quality (i.e. potential value) of a state-action pair, i.e., $\mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. In particular, the Q-function for a state-action pair following a policy π is defined as

$$Q^{\pi}(\mathbf{s}, \mathbf{a}) = \mathbb{E}_{\mathbf{s}'}\big[R(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{\mathbf{a}' \sim \pi(\mathbf{s}')}[Q^{\pi}(\mathbf{s}', \mathbf{a}')]\big], \tag{6.2}$$

where $(\mathbf{s}', \mathbf{a}')$ is the state-action pair at the next step. $Q^{\pi}(\mathbf{s}, \mathbf{a})$ estimates the long-term value of a particular state-action pair following policy π; whereas the *state*

value, denoted by $V^{\pi}(\mathbf{s})$, estimates the expected long-term value of the state $\mathbf{s}$

$$V^{\pi}(\mathbf{s}) = \mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{s})}[Q^{\pi}(\mathbf{s},\mathbf{a})]. \tag{6.3}$$

Furthermore, to better distinguish the relative qualities of all possible actions under a given state $\mathbf{s}$, we define the *advantage* of an action $\mathbf{a}$ following policy π, denoted by $A^{\pi}(\mathbf{s},\mathbf{a})$, as

$$A^{\pi}(\mathbf{s},\mathbf{a}) = Q^{\pi}(\mathbf{s},\mathbf{a}) - V^{\pi}(\mathbf{s}). \tag{6.4}$$

W.r.t. the action branching architecture we employ, let $a_i \in \mathbf{\Omega}_i, i \in \{1,\dots,N\}$ denote a sub-action belonging to action arm i, where $\mathbf{\Omega}_i$ is the set of all sub-actions of action arm i. Then, the Q-value and the action advantage of a_i are denoted by $Q_i^{\pi}(\mathbf{s},a_i)$ and $A_i^{\pi}(\mathbf{s},a_i)$, respectively. A straightforward way to combine the state value and action advantages for Q-values is to follow (6.4), i.e., $Q_i^{\pi}(\mathbf{s},a_i) = V^{\pi}(\mathbf{s}) + A_i^{\pi}(\mathbf{s},a_i)$. However, as suggested by authors in [89], normalising the action advantage by the mean of action advantages before combining it with the shared state value yields better performance. Thus, the following aggregation method is used instead

$$Q_i^{\pi}(\mathbf{s},a_i) = V^{\pi}(\mathbf{s}) + \Big(A_i^{\pi}(\mathbf{s},a_i) - \frac{1}{|\mathbf{\Omega}_i|}\sum_{a_i\in\mathbf{\Omega}_i} A_i^{\pi}(\mathbf{s},a_i)\Big). \tag{6.5}$$

For action arm i, since we use DNN as the function approximator of its sub-action's Q-function $Q_i^{\pi}(\mathbf{s},a_i)$, it is parametrised by the set of adjustable parameters $\boldsymbol{\theta}_i$ which are weights of the DNN. Then, the parametrised Q-function is denoted by $Q_i^{\pi}(\mathbf{s},a_i;\boldsymbol{\theta}_i)$. The *value iteration update* [67] of the Q-function uses the estimation of future rewards at the next state to update current Q-function, with the reasoning that estimations at the next state are more accurate, hence increasingly-

more-accurate $Q_i(\mathbf{s}, a_i; \boldsymbol{\theta}_i)$ is eventually able to converge to the optimal policy π^* after enough rounds of iterations. During weight update, $\boldsymbol{\theta}_i$ is adjusted to reduce the gap between current prediction (i.e., current $Q_i(\mathbf{s}, a_i; \boldsymbol{\theta}_i)$) and the next state estimate. Specifically, we define the target for arm i as

$$y_i = R(\mathbf{s}, \mathbf{a}) + \gamma \max_{a_i' \in \boldsymbol{\Omega}_i} Q_i(\mathbf{s}', a_i'; \boldsymbol{\theta}_i). \tag{6.6}$$

Then, the following loss function using the mean-squared error measurement is defined for adjusting $\boldsymbol{\theta}_i$

$$L(\boldsymbol{\theta_i}) = \mathbb{E}[\big(y_i - Q_i(\mathbf{s}, a_i; \boldsymbol{\theta}_i)\big)^2]. \tag{6.7}$$

Before the weight update process takes place, the total loss is calculated as the the mean across all arms

$$L(\boldsymbol{\theta}) = \mathbb{E}\bigg[\frac{1}{N}\sum_{i=1}^{N} L(\boldsymbol{\theta_i})\bigg]. \tag{6.8}$$

Then, by differentiating $L(\boldsymbol{\theta})$ w.r.t. $\boldsymbol{\theta}$, weights of the DNN are updated for the next iteration, where α is the earning rate.

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}). \tag{6.9}$$

Since the gradient descent updates of $\boldsymbol{\theta}$ is different from canonical supervised learnings because the training target $y_i = R(\mathbf{s}, \mathbf{a}) + \gamma \max_{a_i' \in \boldsymbol{\Omega}_i} Q_i(\mathbf{s}', a_i'; \boldsymbol{\theta}_i)$ is generated by the same Q-function $Q_i(s, a_i; \boldsymbol{\theta}_i)$ that is being trained. Therefore, to improve stability and performance of the training process, we improve the training algorithm in the following three ways.

1). We maintain a delayed version of the Q-function, $Q_i(\mathbf{s}', a_i'; \boldsymbol{\theta}_i')$, for the esti-

mation of the maximum next state reward, which was proposed [59] to improve the stability of their DQN. As such, the target function in (6.6) is updated to

$$y_i = R(\mathbf{s}, \mathbf{a}) + \gamma \max_{a_i' \in \mathbf{\Omega}_i} Q_i(\mathbf{s}', a_i'; \boldsymbol{\theta}_i'). \tag{6.10}$$

The delayed Q-function is updated with the newest weights every C ("target sync gaps") steps by setting $\boldsymbol{\theta}_i' \leftarrow \boldsymbol{\theta}_i$.

2). To overcome the overestimation of action values, we implement *Double Q-learning* [69] to address the positive bias in estimation introduced when the maximum expected action values are instead approximated by the maximum action values in Q-learning. Specifically, we use the up-to-date Q-function $Q_i(\mathbf{s}, a_i; \boldsymbol{\theta}_i)$ to determine the optimal sub-actions, i.e.,

$$a_i'^* = \arg\max_{a_i' \in \mathbf{\Omega}_i} Q_i(\mathbf{s}', a_i'; \boldsymbol{\theta}_i). \tag{6.11}$$

The accumulated reward of the returned action $a_i'^*$ is estimated by the delayed Q-function using (6.10). Therefore, the target in (6.6) is further improved to be

$$y_i = R(\mathbf{s}, \mathbf{a}) + \gamma Q_i(\mathbf{s}', \arg\max_{a_i' \in \mathbf{\Omega}_i} Q_i(\mathbf{s}', a_i'; \boldsymbol{\theta}_i); \boldsymbol{\theta}_i'). \tag{6.12}$$

3). We implement the "replay memory" [70] where the agent's past experiences are stored (*matrix* $\mathcal{D}$) and might be used more than once for training. See Section 6.4.5 for details.

6.4.4 Details of the DNN

The structure of MACS is demonstrated in the example in Fig. 6.3, which can be categorised as a *Multilayer Perceptron (MLP)* [71]. The input to the MLP is of dimension N, which corresponds to a state of the formulated MDP (see Section 6.3.5).

The input layer, which precedes and is fully connected to all action arms and the state layer, consists of 2 hidden fully connected layers with 512 and 256 neurons, respectively. Every action arm contains a hidden layer of 128 neurons which is followed by an output layer with 2 neurons that output the advantage estimations for 2 sub-actions, respectively. The state layer has a hidden layer of 128 neurons followed by a single neuron in its output layer, which gives the estimation of state value. Overall, the output of the MLP is a vector of Q-values whose dimension is $2N$. Note that the state value (output of the state layer) and the advantages of all sub-actions (outputs of the all action arms) are combined according to (6.5). When the trained MLP is used for making action predictions, the chosen sub-action for each action arm is decided by comparing Q-values of its two sub-actions, whichever is greater gets picked, as demonstrated in Fig. 6.3. In cases where the number of arms giving "1" output is larger than the given synchronisation budget, those sub-actions with the greater Q-values get picked first until the budget is reached.

The MLP is realised using Keras [72] model with TensorFlow [73], in which Adam is chosen as the optimiser with initial learning rate of 0.0001 and Rectified Linear Unit (ReLU) [74] is employed as activation functions for all neurons except for the outputs. The discount factor is set to $\gamma = 0.99$; while the target network is updated every 20 steps (i.e., $C = 20$).

6.4.5 The Training Algorithm

So far, the design of immediate reward only takes into account the average reduction in service latency after synchronisation broadcastings. Since most broadcastings of up-to-date BIS bring positive rewards, this makes the agent think that it is always better to broadcast as many as possible. However, as we have the synchronisation budget constraint, we need to make this realisable to the agent during training. To this end, we offset the immediate reward defined in the MDP by a small value for the

sub-actions where up-to-date BISes are broadcasted (i.e., sub-actions indicated by 1, referred to as *positive sub-actions*). In particular, let ι denote the unit offset for each positive sub-action, and ρ be the number of positive sub-actions for the state-action pair $(\mathbf{s}, \mathbf{a})$; then, the 4-tuple stored is $(\mathbf{s}, \mathbf{a}, r, \mathbf{s}')$, where $r = R(\mathbf{s}, \mathbf{a}) - \rho\iota$.

MACS is first pre-trained on past $(\mathbf{s}, \mathbf{a}, r, \mathbf{s}')$ tuples already stored in fixed-size matrix $\mathcal{D}$ (i.e., the agent's "reply memory"). Then, MACS starts making synchronisation decisions, while keeps being trained in a semi-online fashion in the sense that new $(\mathbf{s}, \mathbf{a}, r, \mathbf{s}')$ tuples gradually replace old entries in matrix $\mathcal{D}$ on a first-in-first-out basis. At each training iteration where Q-learning update takes place, minibatch samples of stored experience tuples are pulled randomly from $\mathcal{D}$ for training.

The training process is summarised in Algorithm 3.

6.5 Evaluation

This section starts by introducing the evaluation scenarios and benchmarks in Section 6.5.1. Then, network settings for evaluation scenarios are described in Section 6.5.2. Finally, we present evaluation results and analysis in Section 6.5.3.

6.5.1 Evaluation Scenarios and Performance Benchmarks

Three scenarios are considered for evaluations, where *Scenario 1* serves as the baseline where the overall performance of MACS is compared against other benchmarks. In addition, under the settings of Scenario 1, we also evaluate the performance of MACS in a fully online manner, for which no pre-training on history data is performed. Then, Scenario 2 and Scenario 3 evaluate the sensitivity of MACS performance under different network settings. Specifically, we compare the performance of MACS alongside other benchmarks under varying BIS value distributions in Scenario 2 and varying synchronisation budget in Scenario 3. In the following, we

briefly discuss the benchmarks used for comparing the performance of MACS.

The Full/Worst Controller Synchronisation Levels correspond to the case where all up-to-date BISes are broadcast to domain controllers at every time slot; and the case where there is no broadcasting of any up-to-date BIS at any time slot, respectively. We can see that the full synchronisation level is identical to having one logical central controller and that it incurs the maximum synchronisation overheads. Note that these two cases serve as the lower and upper bounds of our performance metric and they are not subject to the given synchronisation budgets.

The Greedy (MinMax) Algorithm is a simple controller synchronisation scheme that aims at reducing the staleness of controller-perceived BIS values by minimising the maximum state value of the defined MDP. Specifically, with the given synchronisation budget at a time slot, the up-to-date values of those BISes that have not been synchronised to all domain controllers the longest get broadcasted first.

The Anti-entropy [54] Algorithm, which is implemented in the ONOS controller [24], is based on a simple gossip algorithm that controllers randomly synchronise with each other [54]. W.r.t. the definition of controller synchronisation in our problem (Definition 17) and the given synchronisation budget, controller synchronisation with the anti-entropy algorithm is carried out such that up-to-date values of BIS are randomly selected for synchronisation broadcastings.

6.5.2 Network Settings

6.5.2.1 Network topology of the simulated SDN network

For our evaluations, all domain-wise topologies, i.e., how domains are connected to each other, are constructed according to a real dataset. Specifically, there are 8 domains connected according to the *degree distribution* extracted from the “CAIDA AS-27524" dataset [50], which refers to the distribution of the number of neighbour-

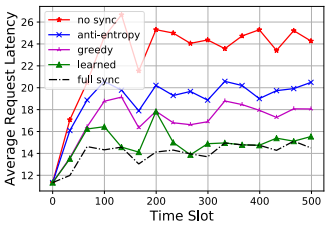


Figure 6.4: Scenario 1: Average request latency of all constructed paths at all time slots.

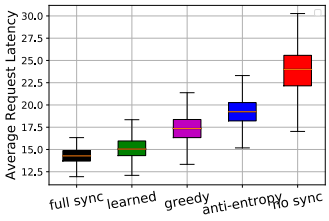


Figure 6.5: Scenario 1: Box plot of average request latency over all times slots.

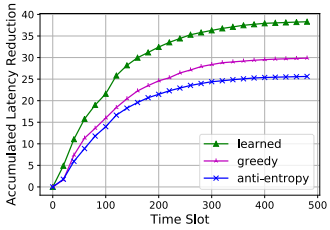


Figure 6.6: Scenario 1: Accumulated average request latency reductions over time slots.

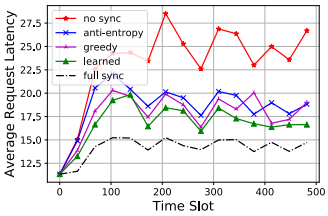


Figure 6.7: Scenario 1: Average request latency at all time slots with *online* MACS.

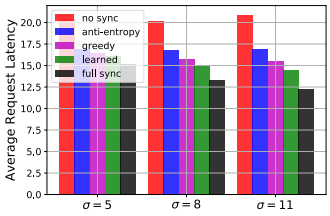


Figure 6.8: Scenario 2: Average request latency with different BIS value distributions.

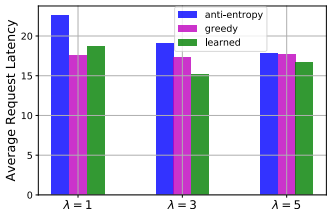


Figure 6.9: Scenario 3: Average request latency with different sync' budget distributions.

ing domains of an arbitrary domain.

6.5.2.2 Server distribution in domains and user request pattern

10 unique services are considered in all scenarios, with two installations in two different domains for each service. Moreover, domains are divided randomly and equally into two groups, service installations are more likely to be inside the first group (with 70% probability) than the second one (with 30% probability). In addition, service request patterns follow Zipf-Mandelbrot distribution [90], which is widely used to model the content popularity in content delivery networks (CDN). Specifically, the popularity of i-th most popular service is proportional to $(q+i)^{-\beta}$, where $q = 5$ and $\beta = 0.8$.

6.5.2.3 Network dynamicity pattern

The distribution of available synchronisation budget for different time slots is modelled by Poisson process with mean $\lambda = 3$ in Scenario 1 and Scenario 2; whereas in Scenario 3, three experiments are conducted with $\lambda = 1$, $\lambda = 3$, and $\lambda = 5$ to evaluate the impacts of the level of available budget. As for BIS values, they are uniformed distributed in Scenario 1 and Scenario 3.[1] In order to evaluate the impact of BIS value distribution on the performance of MACS, we conduct three experiments where the BIS values are Gaussian distributed with mean $\mu = 10$ and standard deviation (STD) being $\sigma = 5$, $\sigma = 8$, and $\sigma = 11$, respectively. All evaluation settings are summarised in Table 6.1.

Remark: It should be noted that the network settings described are only for the sake of evaluations, there are no assumptions on any of the parameters used above.

6.5.3 Evaluation Results

All evaluation results are presented in Fig. 6.4-Fig. 6.9. In particular, Fig. 6.4 shows plots of average request latencies of all constructed service paths in each time slot. Fig. 6.5 shows the box plots of average request latencies resulted from different synchronisation algorithms. Fig. 6.6 contains plots of accumulated latency reduction defined in (6.1), which is the maximisation objective of the MDP. Fig. 6.7 is similar to Fig. 6.4, except that here MACS is trained in the fully online manner as synchronisation decisions are made and new $(\mathbf{s}, \mathbf{a}, r, \mathbf{s}')$ tuples become available for training. Fig. 6.4-Fig. 6.6 all correspond to Scenario 1. The bar plot in Fig. 6.8 are the evaluation results of Scenario 2 where we vary the STD of BIS value distributions. Fig. 6.9 presents the evaluation results of Scenario 3, which illustrate the impacts of varying available synchronisation budget distributions. Note that the bar

[1] In particular, BISes are randomly drawn from the set $\{1, 2, 4, 6, 8, 13, 17, 20, 25, 30\}$. Note that there is no assumption on the relationship between new and old BIS values (e.g., whether or not they are correlated).

plots in Fig. 6.8-Fig. 6.9 are the averaged results over all time slots. In these figures, the legend “learned” refers to the synchronisation policy learned via MACS; “full sync", “greedy", “anti-entropy", and “no sync" respectively refer to the synchronisation cases discussed in Section 6.5.1. Moreover, it should be stressed that in these plots, the unit of the performance metric, i.e., the request latency recorded in the simulated network, and the time scale of the MDP, i.e., the time slot, are different as discussed in Section 6.3.5.

1) Superiority of MACS: From Fig. 6.4, we can see that the gap between “full sync" and “no sync" curves clearly demonstrates the important role controller synchronisation plays in improving performance. Among the three synchronisation algorithms implemented, MACS consistently performs the best in both scenarios. Surprisingly, after 300 time slots, MACS, which runs on a limited synchronisation budget, can almost achieve full sync performance, as can be seen in Fig. 6.4 where two curves overlap. The superiority of MACS over the other two schemes is also evident in Fig. 6.5, which compares the statistic properties of the request latency results of different synchronisation regimes. In addition, these results reveal that while the goal is to minimise long-term average request latency, this objective is achieved by consistently minimising average request latency at each time slot.

2) Superiority of MACS for maximising the Bellman equation: Recall that our objective is to maximise the accumulated reductions in request latency over a period of time, i.e., to maximise $V^{\pi}(\mathbf{s}_0) = \mathbb{E}_{\pi}\left[\sum_{t=1}^{T}\gamma^t R(\mathbf{s}_t, \mathbf{a}_t)|\mathbf{s}_0\right]$. The evaluation results in and Fig. 6.6 confirm the superiority of MACS in achieving this goal. In particular, during the testing period of 500 time slots, MACS outperforms the greedy algorithm by approximately 30%; the anti-entropy by 56%, respectively.

3) Online performance of MACS: In Fig. 6.7, it can be seen that although the synchronisation policy by MACS outperforms that of other algorithms, the performance margin is significantly smaller than in Fig. 6.4 where pre-training is conducted. The

performance differences here indicate the value of pre-training with history data before using MACS to generate synchronisation policies. From Fig. 6.7 we can also see that as the training continues and more training samples become available, synchronisation decisions by MACS keep improving.

4) The impact of BIS value distribution: From the results in Fig. 6.8 we can see that when the STDs of the BIS value distributions increase from 5 to 11, only the "full sync" and "learned" latency results delivered by MACS show improvements. This suggests that the policies developed by MACS are more valuable when the network conditions are highly volatile, manifested by a wider range where BIS values can vary. In comparison, the anti-entropy and greedy algorithms are non-adaptive to varying network conditions, as expected.

5) The impact of available synchronisation budget: Since the "full sync" and "no sync" cases are not subject to the synchronisation budget constraints, we do not plot their evaluation results in Fig. 6.9. We can see that when the average synchronisation budget (λ in the Poisson process) at a time slot increases from 1 to 5, the performance of MACS initially improves and then deteriorates. Recall that MACS makes synchronisation decisions by selecting the sub-actions with the highest Q-values first until the synchronisation budget is exhausted. Therefore, when a large synchronisation budget is allowed, MACS may select some actions with low or even negative Q-values, which explains the performance deterioration. On the other hand, when the synchronisation budget is too constraints (i.e., $\lambda = 1$), the greedy algorithm outperforms MACS. This shows the important role of the budget in regulating the controller synchronisation process. Moreover, the greedy algorithm appears to be insensitive to budget levels, i.e., having greater synchronisation budget does not help its performance.

6.6 Limitations of the MACS

Recall that MACS operates on the assumption that a central node in the SDN network is responsible for learning the synchronisation policy based on necessary information gathered from other controllers in the network. It is also assumed that the central node can always obtain needed information for RL-based policy generation. Such a central node can be a separate entity who can communicate with all domain controllers, or it can be one of the domain controllers. Either way, such a centralised learning scheme is vulnerable since it creates a single point of failure. On the other hand, although the central node is only responsible for developing controller synchronisation policy, and other control operations are reserved for domain controllers, the presence of the central node nonetheless leads to poor scalability as network grows. Moreover, it is not consistent with the design principles of distributed SDN.

Moreover, MACS depends on accurate information from domain controllers for its DRL-based policy generation. For example, the reward values are calculated using the changes in request latency, which are provided by domain controllers. However, in real distributed SDN environment, it is possible that the central node may not be able to receive all information required from other controllers, due to network constraints or anomalies. This creates another vulnerability for MACS, that it may not always be able to obtain all required information for learning.

In the next chapter, further work are conducted to improve MACS's scalability and reliability.

Algorithm 2: Training algorithm for MACS

input : MLP model settings; distributed SDN settings; simulation program for generating rewards.

output: Trained parameterised Q-functions for all arms.

Initialize Q-functions $Q_i(\mathbf{s}, a_i; \boldsymbol{\theta}_i), i \in \{1, \ldots, N\}$ by instantiating the MLP; Set initial state s_0; $t = 0$;
Initialize matrix $\mathcal{D}$ with past $(\mathbf{s}, \mathbf{a}, R(\mathbf{s}, \mathbf{a}) - \rho\iota, \mathbf{s}')$ tuples;
Initialize the delayed Q-functions $\boldsymbol{\theta}'_i \leftarrow \boldsymbol{\theta}_i, i \in \{1, \ldots, N\}$.
while $t \leq T$ **do**
    foreach *time instant* t **do**
        $indicator = 0, 1$ with probabilities $1 - \varepsilon, \varepsilon$, respectively;
        if $indicator = 0$ **then**
            Select an action a_t randomly;
        else
            Select an action a_t according to Section 6.4.4;
        end
        Pass on the $(\mathbf{s}_t, \mathbf{a}_t)$ to the simulation program to get return r_t and $\mathbf{s}_{t+1}$;
        Store $(\mathbf{s}_t, \mathbf{a}_t, r_t - \rho\iota, \mathbf{s}_{t+1})$ in $\mathcal{D}$;
        Pull minibatch $\mathcal{D}_t$ of $(\mathbf{s}_i, \mathbf{a}_i, r_i - \rho\iota, \mathbf{s}_{i+1})$ from $\mathcal{D}$;
        foreach $(\mathbf{s}_i, \mathbf{a}_i, r_i - \rho\iota, \mathbf{s}_{i+1})$ *in* $\mathcal{D}_t$ **do**
            foreach $g \in \{1, \ldots, N\}$ **do**
                Calculate $L(\boldsymbol{\theta}_g)$ from a_g and y_g using (6.7), (6.11), and (6.12), respectively;
            end
            Calculate aggregrated loss $L(\boldsymbol{\theta})$ according to (6.8);
            Update weights: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$;
        end
        if $t \mod C = 0$ *(`C: target sync gap`)* **then**
            foreach $i \in \{1, \ldots, N\}$ **do**
                Update weights: $\boldsymbol{\theta}'_i \leftarrow \boldsymbol{\theta}_i$.
            end
        end
    end
end

Table 6.1: Evaluation Parameters

Parameter	Scenario 1	Scenario 2	Scenario 3
synchronisation budget distribution	Poisson distributed with $\lambda = 3$	Poisson distributed with $\lambda = 3$	Poisson distributed with $\lambda = 1, 3, 5$
BIS value distribution	Uniformly distributed	Gaussian distributed with $\mu = 10$, $\sigma = 5, 8, 11$	Uniformly distributed
Probabilities that BISes change value	The probability of BIS i changing value is proportional to $\frac{1}{\sigma\sqrt{2\pi}} e^{-(i-\mu)^2/2\sigma^2}$, $(\mu = 30, \sigma = 10)$		
Probabilities that service installed in domains	30% probability in any 4 domains and 70% probability in the other 4 domains		
Service request pattern	The probability that service i is requested is proportional to $(q + i)^{-\beta} (q = 5, \beta = 0.8)$		
The number of domains	8 domains		
The number of services	10 unique services, each installed twice in 2 different domains		

CHAPTER 7

SMART: Scalable and Robust SDN Controller Synchronisation Designs by DL and RL Techniques

---

7.1 Introduction

In the previous chapter, we introduce MACS, which delivers fine-grained controller synchronisation policy designs via customised DRL techniques. We note in Section 6.6 that the two main limitations of MACS are the poor scalability and reliability related to the centralised learning paradigm. Therefore, in this chapter, our main aim is to address these two vulnerabilities by introducing a decentralised policy derivation framework based on MACS, which is called SMART.

Similar to the previous chapter, we start by introducing the SDN application which relies on controller synchronisation for improved performance. Recall that we consider two types of network resources for the SDN application in Chapter 6, i.e., network elements for computation and communication. In the discussion of this chapter, we further unify different network resources as services provided by the SDN when consider SDN applications, which reflects the commonly used network function virtualisation (NFV) approaches. Furthermore, instead of only considering service request delay as performance metric as in Chapter 6, in this chapter we employ a more realistic performance metric which incorporates load balancing con-

siderations. To account for more realistic scenarios where RL algorithms do not have full view of the environment due to communication constraints, we formulate the problem of developing the controller synchronisation policy that maximises the defined performance metric as a *partially observable Markov decision process (POMDP)*, which is used to model many serial decision-making problems in real life. For the POMDP, we examine potential decentralised solutions for deriving such policies and then conduct theoretical analysis to identify the conditions upon which a decentralised solution can perform comparably as the centralised counterpart does. This is to ensure that any scalable solution developed does not come at prohibiting costs to performance. With these theoretical results in mind, we then employ a combination of DL and DRL techniques for designing SMART. Specifically, SMART consists of Long-Short Term Memory (LSTM) cells and MACS-based DRL networks, to handle network uncertainties/dynamicity, and to learn policies, respectively. In the high level, SMART's DRL network is trained to generate policies based on temporal data that are collected from the SDN network and enhanced by the LSTM network.

Overall, SMART conducts decentralised RL and employ DL techniques to improve MACS's scalability and robustness.

7.2 Problem Statement

7.2.1 Controller Synchronisation for Service Request Scheduling

In the Network-as-a-Service (NaaS) SDN environment, QoS-aware service provision is a crucial problem in the context where network services are virtualised in servers distributed in different domains. To materialise potential performance gains controller synchronisation can bring, we focus on an application scenario where SDN controller synchronisation helps determine where requests for network services

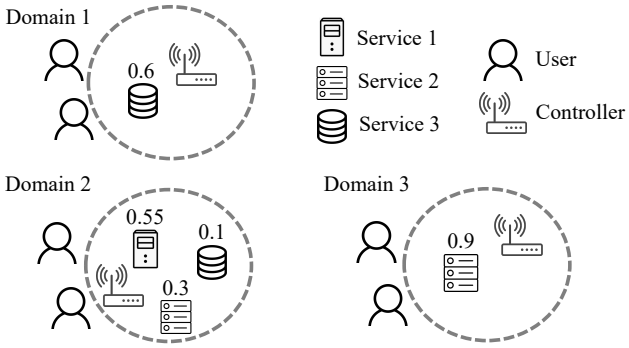


Figure 7.1: An example of a distributed SDN network.

should be submitted, to ensure that work loads across servers in different domains are as evenly distributed as possible, and to minimise request loss due to the lack of available server processing power. Therefore, we choose *service request scheduling* as the application of interest. Specifically, we investigate the problem where several network services, e.g., wireless access admission, firewall, etc., are installed on servers across all SDN domains. Requests for services are submitted by users to domain controllers, which determine where to submit these requests, based on their network views and communications with other controllers. Note that the process of finding a destination server is an anycast problem [80], as a service can have multiple installations in different domains. In order to decide request forwarding rules that balance the server load levels on servers offering the service, domain controllers can give instructions to each other regarding request submissions. In the following, we use a simple example to demonstrate how controller synchronisations assist the service-request-scheduling application.

In Fig. 7.1, servers providing various services are distributed in the SDN domains. In particular, there are 3 SDN domains with their respective domain controllers. Three network services are distributed in three domains as shown in the figure; the numbers above server icons indicate their current load levels. When the load level of a server reaches 1, any additional submitted requests will be rejected due to server overload. Users in all domains can submit requests for services. There

are several users belonging to these domains who request different network services from time to time. Assume that at a time slot all requests for service 2 from users in domain 1 correspond to 60% of the maximum load level. Domain 1 controller is aware that service 2 is available in both domain 2 and 3, and it has to decide where to submit these requests. On the other hand, the controllers of domain 2 and 3 are aware that there are service requests for service 2 coming from domain 1. Without prior knowledge of server status in domain 2 and 3, domain 1 controller may choose to evenly divide the amount of requests received and send them to domain 2 and 3. In this case, 50% of requests will be rejected in domain 3 due to server overflow. However, if domain 3 controller sends a message to domain 1 controller warning it of the critical server load level for service 2, domain 1 controller would then establish a new rule that forwards all requests for service 2 to domain 2. With this new rule, not only all requests can be served in due course, load levels are balanced among servers offering service 2, suppose all servers have the same processing power.

This simple example highlights the importance of controller synchronisation for service request scheduling in distributed SDN. Digging deeper into it, a couple of questions naturally arise. First, what kind of information needs to be synchronised for guaranteeing network performance? Second, is it possible to design a synchronisation schedule which only involves a sub-group of controllers at a time, in order to reduce the overall control plane overhead? In this regard, we formally introduce the concepts of controller synchronisation and the synchronisation budget.

Definition 17. Controller synchronisation w.r.t. the service request scheduling problem is the process of domain controllers informing other domain controllers, via control plane messaging, about the servers in their domains they do not wish to receive service requests for (the messages sent are referred to as request injunctions for servers).

Note that since domain controllers always have up-to-date view of the status of

servers residing in its own domain, they can decide internally the amount of requests to be satisfied by servers offering the requested services that are located within their own domains without receiving instructions from other controllers. In other words, we only consider the portion of service requests to be satisfied by external servers for the controller synchronisation problem. Note that a controller synchronisation message is only valid for the current time slot (see Section 7.3.1 for discussions on time scales). Upon receiving request injunction(s), domain controllers update their request forwarding rules to send requests evenly to remaining accessible servers. If a domain controller receives request injunctions for all servers of a service, it sends requests evenly to all servers offering that service, as a simple heuristic strategy.

Then, we define synchronisation budget which limits the amount of message that a controller can send during a time slot.

Definition 18. The synchronisation budget of a domain controller for a time slot is the maximum number of servers in its domain for which it can issue request injunctions to other domain controllers.

The concept of synchronisation budget quantifies the amount of control messages that are allowed to be exchanged in SDN control plane. The synchronisation budget reflects the status of the network where a higher budget signals a relative good condition that tolerates more communication overheads.

7.2.2 Performance Metric

Discussions in the previous section show that how controllers synchronise with each other has major impacts on the outcomes of service request scheduling. In this section, we define the performance metrics for evaluating the quality of requests scheduling decisions.

Specifically, we define *server makespan* as the total amount of time a server

takes to completely serve all of its waiting jobs. Let $l_{ij}(t)$ denote the makespan of the server offering service i in domain j at time slot t. Clearly, $l_{ij}(t)$ corresponds to the total waiting time for a newly arrived request for service i at the server in domain j at time t. Since new jobs arriving at the server may be rejected and lost due to buffer overflow, we use $w_{ij}(t)$ to denote the amount of work corresponding to the rejected jobs by the server offering service i located in domain j at time slot t. Note that we assume that a domain does not have multiple servers for one service, and one server only hosts one service.

Then, the objective of the request scheduling w.r.t. service i is to (i) minimise the variation of makespan of all servers offering the same service; (ii) minimise the request losses due to overflow caused by uneven distribution of requests. Hence, we define the performance metric w.r.t. service i at time t, $\mathcal{M}_i(t)$, as

$$\mathcal{M}_i(t) = \zeta \sqrt{\frac{\sum_{j=1}^{m} \left(l_{ij}(t) - \sum_{z=1}^{m} l_{iz}(t)/k_i \right)^2}{k_i - 1}} + \eta \sum_{j=1}^{m} w_{ij}(t), \tag{7.1}$$

where k_i is the number of servers providing service i, and m is the total number of domains. It can be seen that the performance metric is defined as the weighted sum of makespan standard deviation for all servers and the amount of lost request work (ζ, η are weighting factors). Clearly, the smaller the metric, the better the performance. We refer to the two terms after ζ and η as *load balance level (LBL)* and *request loss level (RLL)*, respectively. Finally, the overall performance metric $\mathcal{M}$ is obtained by taking the average of the metrics for all services, i.e., $\mathcal{M} = \sum_{i=1}^{n} \mathcal{M}^{(i)}/n$, where n is the total number of services.

7.2.3 Objectives of Controller Synchronisation Policies

Here, a controller synchronisation policy refers to a series of synchronisation decisions defined by Definition 17 over time. With the definitions of controller synchro-

nisation, synchronisation budget, and performance metric, the objective of controller synchronisation policy design is stated as follows

$$\begin{aligned} \min_{d_t} \quad & \frac{1}{T}\mathbb{E}\Big[\sum_{t=1}^{T}\sum_{i=1}^{n}\mathcal{M}_i(t)|d_t\Big] \\ \text{s.t.} \quad & |d_t| \leq \beta_t \quad \text{for} \quad t = 1, 2, \cdots, T \end{aligned} \tag{7.2}$$

where d_t and β_t are the synchronisation decision and synchronisation budget at time slot t, respectively. The constraint $|d_t| \leq \beta_t$ abstractly indicates that the number of synchronisation message should not exceed the given synchronisation budget. See more discussions on this in Section 7.3.4.

Objective: In dynamic SDN networks where user's demands for virtualised network services are time-varying and service-specific, given the controller synchronisation budget, how do domain controllers synchronise with each other according to Definition 17, in order to minimise the defined performance metric in (7.1) over a period of time, subject to the given synchronisation budget?

Remark: It is an extremely challenging task to achieve the stated objective for the following reasons. First, distributed SDN networks are large, complex, and mostly heterogeneous systems exhibiting complicated dynamicity and hebaviors. Traditional model-based control optimisation techniques, which normally come with specific assumptions on network dynamicity or structures, are difficult to be applied for modelling here. Second, this decision-making problem belongs to the class of general job scheduling problems widely known to be NP-complete [91]. Thus, any potential solutions are required to have the ability to generalise for handling the combinatorial complexity. Third, decision-making algorithms need to be able to use inadequate network status data as inputs for making synchronisation decisions, since they are not always able to collect all data needed to avoid excessive overheads the process may incur. In this thesis, we aim to address all these challenges and develop controller synchronisation policies by leveraging a combination of DL and

DRL techniques.

7.3 Problem Formulation and Analysis

In this section, we first formulate the problem of finding controller synchronisation policies for service request scheduling as a POMDP (Section 7.3.1). Then, we analyse the potential RL-based solutions to the formulated POMDP. In particular, we consider two scenarios where the synchronisation policy is developed in centralised and decentralised manners (Section 7.3.2). We conclude that the centralised policy-learning is not sustainable due to the lack of scalability, and thus a decentralised policy is preferred. Naturally, a centralised solution is inherently better than the decentralised counterpart due to the availability of more information. Therefore, we conduct theoretical analysis to identify the conditions under which the decentralised policy can achieve the performance of that of a centralised one (Section 7.3.3).

7.3.1 POMDP Formulation

POMDP [92] offers a mathematical framework for modelling sequential decision-making problems where the operating environment is not fully observable. We will demonstrate in this section why the environment is not always fully observable, and thus why POMDP is a suitable candidate for modelling our controller synchronisation policy. In general, a POMDP can be characterized by an 8-tuple $(\mathcal{S}, \mathbf{b}, \mathcal{A}, T, r, \Omega, O, \gamma)$. For our problem, we define the 8-tuple in the following.

- $\mathcal{S}$ is the finite state space. In particular, a state corresponds to the collection of makespan of all servers in the distributed SDN network. Therefore, the dimension of a state vector is $u = \sum_{i=1}^{n} k_i$, i.e., the total number of servers.
- $\mathbf{b}$ is the belief vector containing the probabilities that the environment is in each state, e.g., $b(\mathbf{s})$ is the probability that the environment is in state $\mathbf{s}$. Note

that the presence of the belief vector suggests that there are times when the exact state of the POMDP is not known. For our problem, since the latest makespan of all servers may not always be obtainable due to potentially poor network conditions, the belief vector estimates the state of POMDP, when some state elements are missing.

- $\mathcal{A}$ is the finite action space. The action vector w.r.t. a state is defined as whether domain controllers issue request injunction(s) for server(s) via controller synchronisation, as defined in Definition 17. Specifically, a binary action vector $\mathbf{a}$ consists of 0's, which indicates no request injunction is sent for the corresponding server; and 1's, the opposite. As such, the dimension of $\mathbf{a}$ is also u.

- T is the set of conditional transition probabilities between states after certain actions are taken. For example, $T(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ ($\mathbf{s} \in \mathcal{S}, \mathbf{a} \in \mathcal{A}$) is the transition probability from state $\mathbf{s}$ to $\mathbf{s}'$ after action $\mathbf{a}$ is taken.

- r is the immediate reward associated with a state-action pair. In particular, $r(\mathbf{s}, \mathbf{a})$ ($\mathbf{s} \in \mathcal{S}, \mathbf{a} \in \mathcal{A}$) is the reward of the state-action pair. Since RL approaches are commonly used to solve POMDPs, $r(\mathbf{s}, \mathbf{a})$ is the stimulus for any RL agent to learn in RL algorithms. Specifically, $r(\mathbf{s}, \mathbf{a})$ signals to the RL agent "good actions" to be reinforced and the bad ones to avoid. For our problem, the designed reward signal consists of 4 components, each conveys a unique message to the RL agent. When calculating rewards, it is assumed that state is known to the RL agent; the state could be estimated by the belief vector, or by other methods which we will discuss later. Formally, the reward is defined as follows

$$r(\mathbf{s}, \mathbf{a}) = \sigma(\mathbf{s}') - \sigma(\mathbf{s}) + (\iota\mathbf{s} \cdot \mathbf{a} - \psi \sum_{i=1}^{u} a_i - \sum_{i=1}^{n} \sum_{j=1}^{m} w_{ij})/u. \tag{7.3}$$

The first component is the difference of LBL between the next and current states seen by the RL agent, i.e., $\sigma(\mathbf{s}') - \sigma(\mathbf{s})$, where $\sigma(\cdot)$ is the standard deviation operator of a vector, and $\mathbf{s}, \mathbf{s}' \in \mathcal{S}$ are the RL agent perceived current and next states. The perceived states can be different from the actual state, see Section 7.4.2 for details. This term indicates to the agent the effectiveness of current action in reducing the makespan variation. The second term is the dot product of the state and action vector, i.e., $\iota\mathbf{s} \cdot \mathbf{a}$, where ι is the scaling factor that decides the relative contribution of this term. This term serves two purposes: it encourages the agent to take 1 action for servers where the current makespan is already high, while also discourages agent from taking 1 action for servers with a low makespan. The third negative term is simply the sum of elements in the action vector times the scaling factor ψ, i.e. $-\psi \sum_{i=1}^{u} a_i$. This term discourages the agent from taking excessive 1 actions, which increases overheads. The last term accounts for the total lost makespan due to overflow after the $(\mathbf{s}, \mathbf{a})$ pair, which is calculated as $-\sum_{i=1}^{n}\sum_{j=1}^{m} w_{ij}$. Note that the last three terms are normalised by the size of the state vector.

- Ω is the observation space, which contains observations of RL algorithms. In our problem, an observation $\boldsymbol{\omega}$ ($\boldsymbol{\omega} \in \Omega$) is the vector of observed server makespan. Note that the newest server makespan may not always be available, therefore, last known server makespan will stay in observations until new values are recorded.

- O is the set of conditional observation probabilities. Specifically, receiving certain observation is conditioned on the next state and the action taken, i.e., $O(\boldsymbol{\omega}'|\mathbf{s}', \mathbf{a})$.

- γ is called the *discount factor* ($0 < \gamma < 1$), which trades off the importance between current and future rewards.

Starting from an initial observation $\boldsymbol{\omega}_0$, the problem is formulated as finding a policy π (i.e., the selection of a sequence of actions $\{\mathbf{a}_t\}_{t=0}^{\Upsilon}$) such that the long-term accumulated reward expressed in the Bellman equation below is maximised

$$V^{\pi}(\boldsymbol{\omega}_0) = \mathbb{E}_{\pi}\Bigg[\sum_{t=1}^{\Upsilon}\sum_{\mathbf{s}_t\in\mathcal{S}} b(\mathbf{s}_t)\gamma^t r(\mathbf{s}_t,\mathbf{a}_t)|\boldsymbol{\omega}_0\Bigg], \tag{7.4}$$

where Υ is the total time horizon of the problem.

Time Scale of the POMDP: The basic unit of time in the defined POMDP includes a series of events, which are jointly referred to as a *synchronisation time slot* (*time slot* for short) in the sequence. Specifically, synchronisation messages exchanges take place at the start of a time slot. Then, based on the request injunctions controllers received, they update rules for forwarding service requests as described in Section 7.2.1. For modelling tractability, we assume that the lifespan of a synchronised request injunction is only the current time slot. This is true in practical SDN networks where periodical synchronisation means that the collection of latest network information takes place at certain time.

With the formulated POMDP, RL algorithms are commonly used to find π that maximise (7.4). For RL, imagine an agent who moves from state to state in the formulated POMDP by taking some actions associated with certain rewards. The agent's goal is to discover a sequence of state-action pairs, i.e., the *policy* π, that maximises the accumulated time-discounted rewards. Although the agent's observability of states are incomplete in the formulated POMDP. By interacting with the environment, the agent's internal belief vector are becoming increasingly accurate as its experiences build up through "trial-and-error" where good decisions are positively enforced by positive rewards, and bad ones the opposite.

When designing RL algorithms, a central question is whether the learning process takes place in a centralised or a relatively decentralised manner. In the next

section, we analyse these design principles in the context of our synchronisation problem.

7.3.2 Centralised Vs. Decentralised Policy Learning

There are three important issues of concern in order for any RL algorithms to generate an appropriate synchronisation policy: (i) Whether to train the state-action to value function(s) using the single or multi-agent setting? (ii) Since states are partially observable to RL agent(s), how to help agent(s) resolve uncertain states from observations and update their belief vector(s)? (iii) How to map trained value functions to policies? In this subsection, we consider the first question, and defer the other two to the next section.

Centralised Learning Approach: For the formulated POMDP, the most straightforward way to learn value function is to employ the classic single agent Q-learning algorithm[67]. Here, we use Q-learning only for the analysis of centralised and decentralised policy learning processes.

In general, Q-learning algorithm uses the *Q-function* to estimate the quality of a state-action combination

$$\mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}.$$

In particular, the optimal Q-function for our POMDP, denoted by $Q^*(\boldsymbol{\omega}, \mathbf{a})$, can be written as

$$Q^*(\boldsymbol{\omega}, \mathbf{a}) = \sum_{\mathbf{s} \in \mathcal{S}} b(\mathbf{s})[r(\mathbf{s}, \mathbf{a}) + \gamma \sum_{\mathbf{s}', \boldsymbol{\omega}'} p \max_{\mathbf{a}'} Q^*(\boldsymbol{\omega}', \mathbf{a}')], \tag{7.5}$$

where $\mathbf{s}', \boldsymbol{\omega}', \mathbf{a}'$ are the next step state, action, and observation, respectively; $p = T(\mathbf{s}'|\mathbf{s}, \mathbf{a})O(\boldsymbol{\omega}'|\mathbf{s}', \mathbf{a})$. The *value iteration update* process, which is detailed in (7.6) (α is the learning rate ($0 < \alpha < 1$)), makes $Q(\boldsymbol{\omega}, \mathbf{a}) \rightarrow Q^*(\boldsymbol{\omega}, \mathbf{a})$ when $t \rightarrow \infty$. In particular, as the learning process proceeds, the Q-function is updated using the

following rule:

$$Q(\boldsymbol{\omega}, \mathbf{a}) = (1-\alpha)Q(\boldsymbol{\omega}, \mathbf{a}) + \alpha[r(\mathbf{s}, \mathbf{a}) + \gamma \sum_{\mathbf{s}', \boldsymbol{\omega}'} p \max_{\mathbf{a}'} Q(\boldsymbol{\omega}', \mathbf{a}')]. \tag{7.6}$$

where α ($0 < \alpha < 1$) is the learning rate.

Remark: Although the centralised learning approach based on the Q-learning can eventually make the value function converge to the exact one in the limit of time, it is not difficult to see the intractability of the centralised learning approach as follows. First, the centralised approach is not scalable. With only one RL agent gathering all required information for training, not only a single point of failure is possible, the single agent needs to have a tremendous amount of processing power to handle large-scale information gatherings and RL trainings. Second, the size of the state-action space in the POMDP increases exponentially in the number of servers in the network. According to [67], one prerequisite of the convergence guarantee of Q-learning is that all state-action pairs are visited infinitely often, which is a condition almost impossible to meet for large networks. Based on our experiences, the agent almost would not be able to learn anything at all, if only applying (7.6) without generalisation.

Decentralised Learning Approach: The discussion above implicitly advocates for decentralised learning approaches in practice, which potentially involves multiple RL agents. When pursuing a decentralised RL approach for learning policies based on the formulated POMDP, a natural way to proceed is to create multiple RL agents, each for a type of service. In other words, all servers providing the same service form a *Learning Group (LG)*, and one RL agent is assigned for this LG, referred to as the *LG agent*. Note that the LG agent performs a set of learning functionalities, which can physically reside on a domain controller, or it can be implemented as an independent network unit. We use an example in Fig. 7.2 to illustrate the concept

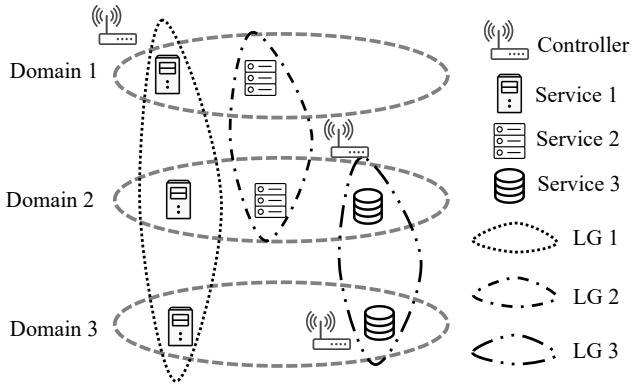


Figure 7.2: An example of 3 Learning Groups (LGs).

of LGs. There are 3 services supported in 3 domains in Fig. 7.2, and thus 3 LGs are formed as shown. Each LG agent is responsible for training the LG-wise value function based on observations associated with the LG. These observations are past makespan values and actions taken for the servers within the LG. Again, the quality of the observations depends on the ability of the agent to gather information from LG members, which is ultimately dictated by network conditions. In addition, the synchronisation budget defined in Definition 18 is the bond that link together all LGs together, as the decisions made by some LGs may compete for the limited synchronisation budget of some domain controllers.

Essentially, each LG agent manages a subset of the 8-tuple $(\mathcal{S}, \mathbf{b}, \mathcal{A}, T, r, \Omega, O, \gamma)$ of the formulated POMDP. Therefore, we use $(\mathcal{S}_i, \mathbf{b}_i, \mathcal{A}_i, T_i, r, \Omega_i, O_i, \gamma), i \in \{1, 2, \cdots, n\}$ to denote the section of the POMDP accessed by LG agent i. Let the value function of LG agent i be denoted by $Q_i(\boldsymbol{\omega}_i, \mathbf{a}_i)$. Comparing to the centralised single agent scenario, where the RL agent has access to all aspects of the POMDP, one naturally wonders how the value functions developed by the decentralised LG agents compare to the centralised one? In other words, we need to analyse the relationship between $Q(\boldsymbol{\omega}, \mathbf{a})$ and $\hat{Q}(\boldsymbol{\omega}, \mathbf{a})$, which is defined as

$$\hat{Q}(\boldsymbol{\omega}, \mathbf{a}) = \sum_{i=1}^{n} Q_i(\boldsymbol{\omega}_i, \mathbf{a}_i). \tag{7.7}$$

If it can be proved that $\hat{Q}(\boldsymbol{\omega}, \mathbf{a}) = Q(\boldsymbol{\omega}, \mathbf{a})$, this implies that while a decentralised policy improves the scalability and robustness of the learning process, it also retains the same performance as achieved by a centralised policy. This is the issue to be examined as follows. In particular, we first investigate the relationship between $\hat{Q}(\boldsymbol{\omega}, \mathbf{a})$ and $Q(\boldsymbol{\omega}, \mathbf{a})$ without considering the constraints of synchronisation budget. Then, by taking into account the synchronisation budget that all agents share, we identify the protocols by which the agents work cooperatively to achieve comparable performance of the centralised learning approach.

7.3.3 Performance Analysis of the Decentralised Learning Approach Without Considering Synchronisation Budget

First, we define $Q(\mathbf{s}, \boldsymbol{\omega}, \mathbf{a})$ and $Q_i(\mathbf{s}_i, \boldsymbol{\omega}_i, \mathbf{a}_i)$ at time slot t, which are shortened to Q^t and Q_i^t, as

$$Q^t = r(\mathbf{s}^t, \mathbf{a}^t) + \gamma \sum_{\mathbf{s}^{t+1}, \boldsymbol{\omega}^{t+1}} p \max_{\mathbf{a}^{t+1}} Q^{t+1}, \tag{7.8}$$

and

$$Q_i^t = r(\mathbf{s}_i^t, \mathbf{a}_i^t) + \gamma \sum_{\mathbf{s}_i^{t+1}, \boldsymbol{\omega}_i^{t+1}} p_i \max_{\mathbf{a}_i^{t+1}} Q_i^{t+1}, \tag{7.9}$$

respectively, where $p_i = T(\mathbf{s}_i^{t+1}|\mathbf{s}_i^t, \mathbf{a}_i^t) O(\boldsymbol{\omega}_i^{t+1}|\mathbf{s}_i^{t+1}, \mathbf{a}_i^t)$. We start our analysis by comparing Q^t and Q_i^t.

Let Υ be the total number of time slots in the POMDP, then, (7.8) and (7.9) reduce to $r(\mathbf{s}^t, \mathbf{a}^t)$ and $r(\mathbf{s}_i^t, \mathbf{a}_i^t)$ when $t = \Upsilon - 1$, respectively, because there are no future rewards. Then, $Q^t = \sum_{i=1}^n Q_i^t$ holds for $t = \Upsilon - 1$, since by definition there is $r(\mathbf{s}, \mathbf{a}) = \sum_{i=1}^n r(\mathbf{s}_i, \mathbf{a}_i)$. Then, by mathematical induction, we change the time slot to $t-1$ and apply the result from t to identify the conditions for which the same holds for $t-1$. For easier expression, we use Q^t and Q_i^t to represent $Q(\mathbf{s}^t, \boldsymbol{\omega}^t, \mathbf{a}^t)$

and $Q_i(\mathbf{s}_i^t, \boldsymbol{\omega}_i^t, \mathbf{a}_i^t)$ in the sequence. Below we apply the results proved at $t = \Upsilon - 1$ to time slot $t-1$, and add necessary conditions for the induction proof. See the texts later for the reasoning for each step.

$$Q^{t-1} = r(\mathbf{s}^{t-1}, \mathbf{a}^{t-1}) + \gamma \sum_{\mathbf{s}^t, \boldsymbol{\omega}^t} p \max_{\mathbf{a}^t} Q^t \tag{7.10}$$

$$= \sum_{i=1}^{n} r(\mathbf{s}_i^{t-1}, \mathbf{a}_i^{t-1}) + \gamma \sum_{\mathbf{s}^t, \boldsymbol{\omega}^t} p \sum_{\mathbf{s}_i^t, \boldsymbol{\omega}_i^t} \max_{\mathbf{a}_i^t} Q_i^t \tag{7.11}$$

$$= \sum_{i=1}^{n} r(\mathbf{s}_i^{t-1}, \mathbf{a}_i^{t-1}) + \gamma \sum_{i=1}^{n} \sum_{\mathbf{s}_i^t, \boldsymbol{\omega}_i^t} p_i \max_{\mathbf{a}_i^t} Q_i^t \tag{7.12}$$

$$= \sum_{i=1}^{n} Q_i^{t-1}. \tag{7.13}$$

From (7.10) to (7.11), we simply apply the results obtained for $t = \Upsilon - 1$. The key condition we used from (7.11) to (7.12) is $\sum_{\mathbf{s}^t, \boldsymbol{\omega}^t} p = \sum_{i=1}^{n} \sum_{\mathbf{s}_i^t, \boldsymbol{\omega}_i^t} p_i$. What this condition means is that the state and observation probabilities of an LG are fully determined by the members inside the LG. This is the first condition we identified.

Based on (7.5) and (7.10) - (7.13), we use the similar approach to prove $\hat{Q}(\boldsymbol{\omega}, \mathbf{a}) = Q(\boldsymbol{\omega}, \mathbf{a})$ as follows

$$Q(\boldsymbol{\omega}, \mathbf{a}) = \sum_{\mathbf{s} \in \mathcal{S}} b(\mathbf{s}) Q(\mathbf{s}, \boldsymbol{\omega}, \mathbf{a}) \tag{7.14}$$

$$= \sum_{\mathbf{s} \in \mathcal{S}} b(\mathbf{s}) \sum_{i=1}^{n} Q_i(\mathbf{s}_i, \boldsymbol{\omega}_i, \mathbf{a}_i) \tag{7.15}$$

$$= \sum_{i=1}^{n} \sum_{\mathbf{s}_i} b_i(\mathbf{s}_i) \sum_{i=1}^{n} Q_i(\mathbf{s}_i, \boldsymbol{\omega}_i, \mathbf{a}_i) \tag{7.16}$$

$$= \sum_{i=1}^{n} Q_i(\boldsymbol{\omega}_i, \mathbf{a}_i). \tag{7.17}$$

From (7.14) to (7.15), we apply the result in (7.10) - (7.13). The key condition in deriving (7.16) from (7.15) is $\sum_{\mathbf{s} \in \mathcal{S}} b(\mathbf{s}) = \sum_{i=1}^{n} \sum_{\mathbf{s}_i} b_i(\mathbf{s}_i)$. Similar to the previous

finding, this condition suggests that a LG agent's belief vector is fully determined and updated by members of the LG. By closer examination, this condition can actually be seen as a corollary of the previous one, as belief vector is updated based upon observations. We summarise these identified conditions as follows.

Without considering the synchronisation budget, the collection of all value functions gained via LG-based decentralised policy learning approach is guaranteed to be identical to the single value function learned in the centralised manner, if the observation probabilities, the belief vectors and their updates of all LG agents are fully determined by members of the respective LG.

Note that we have not yet considered the synchronisation budget for domain controllers, which impose constraints on the actions derived by LG agents. In the next subsection, we discuss the impacts of synchronisation budget on the performance of our LG-based decentralised approach.

7.3.4 Performance Analysis of the Decentralised Learning Approach Considering Synchronisation Budget

Due to the limited synchronisation budget assigned to domain controllers, it is possible that servers belonging to different LGs are competing for synchronisation budget in various domains. Specifically, let β_j denote the synchronisation budget, defined according to Definition 18, for domain controller j at time t. Then, let $a_{ij} \in \mathbf{a}_i$ (a_{ij} is either 0 or 1 per definition of the action space in Section 7.3.1) be the action element for the server offering service i in domain j at time t. Then, for the domain controller j, there is no competition for synchronisation budget if $\sum_{i=1}^{n} a_{ij} \leq \beta_j$ ($a_{ij} = 0$ if service i is not available in domain j). On the other hand, the domain controller j needs to decide how to assign the synchronisation budget if $\sum_{i=1}^{n} a_{ij} > \beta_j$. The constraint of the synchronisation budget essentially restricts the values of some action elements in the affected LGs. In particular, let Z and H denote the set containing

the indices of the LG(s) that have server(s) competing for synchronisation budget, and the set containing the indices of the domains that have servers competing for synchronisation budget, respectively. Then, based on results from (7.10) - (7.13), the collection of value functions learned for LGs competing for the synchronisation budget work cooperatively to solve the following optimisation problem.

$$\begin{aligned} \max_{\mathbf{a}_i} \quad & \sum_{i \in Z} Q_i(\mathbf{s}_i, \mathbf{a}_i) \\ \text{s.t.} \quad & \sum_{i=1}^{n} a_{ij} \leq \beta_j \quad \text{for all} \quad j \in H. \end{aligned} \tag{7.18}$$

Therefore, the performance of the decentralised learning approach is ultimately decided by how close the solution of (7.18) is, when compared to the optimal solution. As such, in theory, our LG-based decentralised learning approach can achieve the performance of the centralised learning approach if (7.18) is solved exactly.

As we shall show in Section 7.4, the policy learning part of the SMART, which is employed to learn synchronisation policies for LGs, is based on the MACS designed in Chapter 6. Therefore, in Section 7.4.4, we develop a heuristic algorithm, which take advantage of the MACS design, to solve the optimisation problem defined in (7.18).

7.4 The SMART

7.4.1 Overview of the SMART Design

Firstly, recall that the state in our POMDP is defined as the makespan of all servers in the SDN networks. Correspondingly, with the multi-agent learning approach we use, each LG agent collects current server makespan from its LG members. However, LG agents are not always able to collect all up-to-date server makespan, due to network constraints that prevent timely communications between network components, thus

resulting in the partial observability in the problem formulations. Temporal data enhancement techniques are therefore needed to resolve such partial observability. Traditionally, methods used to update the belief vectors based on observations are studied and applied to resolve the partial observability. Here, we propose to use LSTM[93] to help LG agents resolve unobservable states. LSTM is a type of recurrent neural network (RNN) that has been used successfully in the wider area of temporal data forecasting (see Section 7.4.2). For our LSTM, the input data is the LG agent's observations and actions in the past several time slots. The LSTM outputs the prediction of current states if such information cannot be obtained by the agent. In other words, if the LSTM is able to make 100% accurate predictions about unobservable states, this is equivalent to having a belief vector with all 0 but only one entry equals to 1 for the POMDP. We show in Section 7.5 that LSTM works especially well when a large portion of state elements are unobservable.

Secondly, we describe details of MACS-based DRL network designed for LG agents to output selected actions based on past observations. Essentially, the DRL network is a value function that evaluates the long-term quality of an observation-action combination. Given the enormity of the state-action space (even for the sub-state-action space that an LG agent controls), we use deep neural network (DNN) [66] in the DRL network to generalise the value function due to its exceptional ability in capturing latent and complicated relationships from input data. As such, the input to the DRL network is the concatenation of observed state elements and the predictions made by LSTM for those unobserved states elements. The design for the DRL networks used by all LG agents is similar to MACS introduced in the previous chapter, refer to Section 6.4.2 for details.

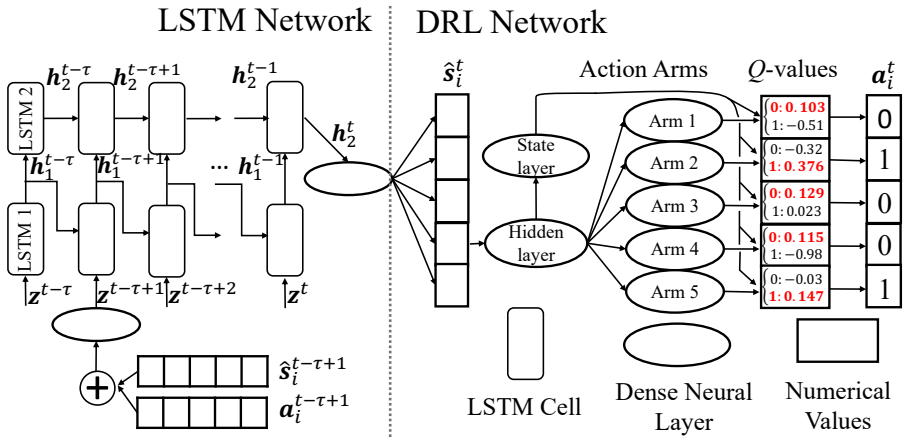


Figure 7.3: The structure of SMART.

7.4.2 LSTM for Resolving Unobservable States

LSTM is the state-of-the-art of RNNs, which solves the vanishing gradient problem that previously plagued the training process of long RNNs. Here, we employ LSTM to handle unobservable states for LG agents, which can be caused by hostile network environment or communication resource constraints. See [93] for more details about LSTM and its structure.

For our task, the input to an agent's LSTM network is its past observations up to τ time slots ago from the current time slot. Note that an observation includes the LG agent's perceived makespan and the action taken by the agent at that time slot. The left part of Fig. 7.3 shows the structure of our designed LSTM network. Here, we stack 2 LSTM cells to enhance the capability of the LSTM network. At each time slot, the observations up to τ time slots ago are fed into the LSTM cells sequentially in the order of time. For the observation input at time t, the agent perceived state ($\hat{\mathbf{s}}_i^t$) and actions taken ($\mathbf{a}_i^t$) are concatenated to one vector and are fed into a two-layer Multilayer Perceptron (MLP) before entering the fist LSTM cell. The input to the first LSTM cell at time slot t is $\mathbf{z}^t = \text{MLP}[2u \times 128](\hat{\mathbf{s}}_i^t \oplus \mathbf{a}_i^t)$ ($\oplus$ indicates the concatenation of vectors), where the numbers inside the square brackets indicate the number of neurons in the input/output layers of the MLP. Then, the outputs of the

first LSTM cells, $\mathbf{h}_1^t = \text{LSTM}[64](\mathbf{z}^t, \mathbf{h}_1^{t-1})$, are passed on to the second LSTM cell, and to itself as the input in the next time step. The number in the square brackets is the number of memory units we use in the LSTM. The second LSTM cell takes as input its own output from the previous time slot $\mathbf{h}_2^{t-1} = \text{LSTM}[64](\mathbf{h}_1^{t-1}, \mathbf{h}_2^{t-2})$, and the input from the first LSTM cell $\mathbf{h}_1^t$. Finally, the outputs of the second LSTM cell at the last time step enter an output MLP with the number of input/output neurons being $128 \times k_i$. The output of this MLP is the final output of the LSTM network, which will be fed into the DRL network for generating synchronisation decisions.

7.4.3 MACS-based DRL Network for Learning Policies

For the MACS-based DRL network used by all LG agents, let the value function of LG agent i be denoted by $Q_i(\hat{\mathbf{s}}_i, \mathbf{a}_i)$, the *state value* of the agent's resolved state $\hat{\mathbf{s}}_i$ is defined as

$$V_i(\hat{\mathbf{s}}_i) = \mathbb{E}_{\mathbf{a}_i(\mathbf{s})}[Q_i(\hat{\mathbf{s}}_i, \mathbf{a}_i)]. \tag{7.19}$$

Note that with the help of the LSTM network, the observation of LG agent i, $\boldsymbol{\omega}_i$, turns into resolved state $\hat{\mathbf{s}}_i$.

Furthermore, to better distinguish the relative qualities of all possible actions under a given state $\mathbf{s}_i$, we define the *advantage* of an action $\mathbf{a}_i$, denoted by $A_i(\hat{\mathbf{s}}_i, \mathbf{a}_i)$, as

$$A_i(\hat{\mathbf{s}}_i, \mathbf{a}_i) = Q_i(\hat{\mathbf{s}}_i, \mathbf{a}_i) - V_i(\hat{\mathbf{s}}_i). \tag{7.20}$$

Following [89], we define the value of a sub-action belonging to action arm j for agent i, denoted by $a_{ij} (a_{ij} \in \{0, 1\})$, as

$$Q_{ij}(\hat{\mathbf{s}}_i, a_{ij}) = V_i(\hat{\mathbf{s}}_i) + \big(A_i(\hat{\mathbf{s}}_i, a_{ij}) - \sum_{j \in \{0,1\}} A_i(\hat{\mathbf{s}}_i, a_{ij})/2\big). \tag{7.21}$$

To help agent i generalise over the state-action pairs it has seen, we parame-

terise its value function by DNNs, which is denoted by $Q_{ij}(\hat{\mathbf{s}}_i, a_{ij}; \boldsymbol{\theta}_{ij})$. The set of adjustable parameters $\boldsymbol{\theta}_{ij}$ are weights of the DNN.

Training the DRL Network: The *value iteration update* [67] of the Q-function uses the estimation of future rewards at the next state to update current Q-function, with the reasoning that estimations at the next state are more accurate, hence $Q_{ij}(\hat{\mathbf{s}}_i, a_{ij}; \boldsymbol{\theta}_{ij})$ is eventually accurate enough to estimate the values of both "0" and "1" actions of each action arm. Then, the synchronisation policy can be easily derived by the LG agent by comparing the value estimations of two sub-actions of each action arm. (See ***Phase 4*** in Section 7.4.5 for details.)

During each weight update iteration, $\boldsymbol{\theta}_{ij}$ is adjusted to reduce the gap between current prediction (i.e., current $Q_{ij}(\hat{\mathbf{s}}_i, a_{ij}; \boldsymbol{\theta}_{ij})$) and the next state estimate. Specifically, we define the target for action arm j of agent i as

$$y_{ij} = r(\hat{\mathbf{s}}_i, a^{ij}) + \gamma \max_{a'_{ij}} Q_{ij}(\hat{\mathbf{s}}'_i, a'_i; \boldsymbol{\theta}_{ij}). \tag{7.22}$$

Then, the following loss function using the mean-squared error measurement is defined for adjusting $\boldsymbol{\theta}_{ij}$

$$L(\boldsymbol{\theta}_{ij}) = \mathbb{E}[\big(y_{ij} - Q_{ij}(\hat{\mathbf{s}}_i, a_{ij}; \boldsymbol{\theta}_{ij})\big)^2]. \tag{7.23}$$

Before the weight-update process takes place, the total loss is calculated as the mean loss across all arms

$$L(\boldsymbol{\theta}_i) = \mathbb{E}\bigg[\frac{1}{k_i}\sum_{j=1}^{k_i} L(\boldsymbol{\theta}_{ij})\bigg]. \tag{7.24}$$

Then, by differentiating $L(\boldsymbol{\theta}_i)$ w.r.t. $\boldsymbol{\theta}_i$, the weights of the DNN are updated for

the next iteration (α is the learning rate)

$$\boldsymbol{\theta}_i \leftarrow \boldsymbol{\theta}_i - \alpha \nabla_{\boldsymbol{\theta}_i} L(\boldsymbol{\theta}_i). \tag{7.25}$$

Other details of the DRL network are described as follow. As shown in the right part of Fig. 7.3, The input layer, which is fully connected to all action arms and the state layer, consists of 2 hidden fully connected layers with 512 and 256 neurons, respectively. Every action arm contains a hidden layer of 128 neurons which is followed by an output layer with 2 neurons that output the advantage estimations for 2 sub-actions, respectively. The state layer has a hidden layer of 128 neurons followed by a single neuron in its output layer.

7.4.4 Synchronisation Policy Generation With Given Synchronisation Budgets

Note that the core idea behind the MACS-based DRL network in SMART is to enable the estimations of relative contributions of each action element. For example, "0" action is selected for the server providing service i in domain j if $Q_{ij}(\mathbf{s}_i, a_{ij})|_{a_{ij}=0} > Q_{ij}(\mathbf{s}_i, a_{ij})|_{a_{ij}=1}$. Therefore, the structure of the value function for SMART offers a natural heuristic to solve the optimisation problem defined in (7.18) in a decentralised way. In particular, for the domain controller j with the synchronisation budget β_j, let $\mathbf{d}_j$ denote the initial action vector containing the chosen action elements for all servers in its domain. Then, $\mathbf{d}_j$ is determined by

$$\mathbf{d}_j = \arg\max_{\{a_{ij}\}} \sum_{i \in I_j} Q_{ij}(\mathbf{s}_i, a_{ij}), \tag{7.26}$$

where set I_j contains the indices of all services domain j provides. For the server offering service i in domain j, its corresponding action element is obtained by sim-

ply choosing the sub-action with higher value, i.e., $a_{ij} = 0$ if $Q_{ij}(\mathbf{s}_i, a_{ij})|_{a_{ij}=0} \geq Q_{ij}(\mathbf{s}_i, a_{ij})|_{a_{ij}=1}$, or $a_{ij} = 1$ otherwise. Therefore, $\mathbf{d}_j$ is obtained by carrying out such comparisons for all servers in I_j using the learned $Q_{ij}(\mathbf{s}_i, a_{ij})$. The initial action vector $\mathbf{d}_j$ is then adjusted to the final action $\mathbf{d}'_j$ as follows. If there is no competition for the synchronisation budget in domain j, i.e., $\sum\{\mathbf{d}_j\} \leq \beta_j$, the initial action vector $\mathbf{d}_j$ stands unchanged, i.e., $\mathbf{d}_j = \mathbf{d}'_j$. Otherwise, the values associated with all "1" actions are ranked in descending order. The "1" actions whose corresponding values are in top β_j in the rank remain unchanged, and those "1" actions whose corresponding values are not in top β_j in the rank are changed to "0" actions, thus rendering a new $\mathbf{d}'_j$. Finally, the concatenate of all $\mathbf{d}'_j$ action vector provide the heuristic solution to the optimisation problem in (7.18).

7.4.5 From Data to Policy

In this section, we describe the whole process of how LG agents turn their available network data to a executable policy. To reduce the complexity of the training process and to avoid the potential vanishing gradient problem resulted from very deep neural networks, we train the LSTM and the DRL networks separately. The whole process can be summarised in the following phases.

Phase 1: Data Collection. At each time slot t, LG agent i collects up-to-date makespan from all servers within its LG to the best of its ability. If the agent does not have access to the newest makespan information of some servers in this time slot due to network constraints, it uses the last known makespan values. The agent then store its observation of time slot t as ω_i^t.

Phase 2: Collected Data for LSTM Network Training. To train the LSTM network, each agent divides its received observations into chunks of consecutive observations whose lengths are $\tau + 1$. The first τ observations in a chunk are sequentially

fed to the LSTM cells and the observed state in the last observation in the chuck is the training target.

Phase 3: Collected Data from trained LSTM for DRL Network Training. The training data for the DRL network are the $(\hat{\mathbf{s}}_i, \mathbf{a}_i, r(\hat{\mathbf{s}}_i, \mathbf{a}_i), \hat{\mathbf{s}}'_i)$ tuples. Here, $\hat{\mathbf{s}}_i$ and $\hat{\mathbf{s}}'_i$ are the resolved current and next states of agent i, with unobservable state elements predicted by the trained LSTM network.

Phase 4: Trained DRL Network for synchronisation Policy. Preliminary controller synchronisation decision at time slot t is derived by the agent's trained DRL with its perceived state $\hat{\mathbf{s}}_i^t$ as input. Specifically, as shown in the right side of Fig. 7.3, at any time slot, each agent's DRL network estimates Q-values for two sub-actions of all action arms (each action arm corresponds to a server in the LG) from the $\hat{\mathbf{s}}_i^t$ inputs. Finally, each domain controller, with the given given synchronisation budget determined by the current network conditions, generates the final action vector using the heuristic algorithm introduced in Section 7.4.4.

7.5 Evaluation

This section presents performance evaluations of SMART, in which we compare its performance to other algorithms in different scenarios. We also evaluate the performance of SMART in hostile network environments where significant amount of data used for deriving the policy are unobtainable. Finally, to test how well SMART handles growing number of services, we fix the synchronisation budget while increase the number of services across SDN domains in a series of experiments, and compare the performance of SMART to all benchmarks. Here, we first introduce the evaluation scenarios and the performance benchmarks used in Section 7.5.1. Then, the evaluating settings studied are described in Section 7.5.2. Finally, we present the evaluation results and analysis in Section 7.5.3.

Table 7.1: Evaluation Parameters

Parameter	Scenario 1	Scenario 2	Scenario 3	Scenario 4
The amount of available state status data	100%	100%	**50%, 70%, 100% in three cases**	100%
Maximum sync' budget at a time slot	**Gaussian distributed with $\mu = 1$**	**1,2, and 3 in three cases**	Gaussian distributed with $\mu = 1$	**2 for all cases with different # services**
# domains and services	6 domains and 4 unique services; each service has 3,4,5,6 installations (servers), respectively			**6 domains and 3-10 unique services in 8 cases**
Server processing power	service 1: $[0.15, 0.2, 0.25]$, service 2: $[0.1, 0.15, 0.2, 0.25]$, service 3: $[0.1, 0.15, 0.2, 0.25, 0.3]$, service 4: $[0.1, 0.15, 0.2, 0.25, 0.2, 0.15]$			3 installations per service, all server processing power uniformly distributed between 0.1 and 0.3
Max server makespan	1 for all servers			
# requests per time slot	Poisson distributed with $\lambda = 1$			
Unit request length	0.04			
Other parameters	$\zeta = 1$, $\eta = 1$, $\iota = 0.5$, $\phi = 0.25$, $\Upsilon = 300$, $\tau = 10$			

7.5.1 Evaluation Scenarios and Performance Benchmarks

Four scenarios are considered for evaluations. In Scenario 1, we compare the performance of SMART over three other algorithms in an ideal environment assuming all data are available for training SMART. As for Scenario 2, we vary the synchronisation budget and compare the performance of SMART with other algorithms. Then, in Scenario 3, we evaluate the performance of SMART in networks with increasing difficulty in obtaining input data for training. Finally, in Scenario 4, we test SMART's performance when an increasing number of services are present in the network. Before discussing detailed settings for 4 scenarios, we briefly introduce the benchmarks employed as follows.

The Local Greedy Algorithm is a controller synchronisation scheme in which each domain controller aims at minimising the makespan of the most loaded servers in its domain. Therefore, the domain controller issues request injunctions to other controllers for the servers that are most heavily loaded first. In this way, domain controllers attempt to minimise makespan lost due to requests overflow.

The Anti-entropy [54] Algorithm implemented in the ONOS controller [24], is based on a gossip algorithm that controllers randomly synchronise with each other [54]. For our synchronisation problem, a domain controller running anti-entropy algorithm informs and prevents other controllers from submitting requests to a number of randomly selected servers.

The Static/Pre-calculated Policy are controller synchronisation decisions made based on statistical data about network dynamics. In particular, domain controllers are aware of the average number of request arrivals for each server and the servers' processing power. They plan the issuance of service injunctions in advance to minimise request lost due to overflows.

Remark: The three benchmarks we use are representations of some existing synchronisation strategies. The local greedy algorithm can be seen as a fully decentralised policy in which decisions are made based only on local information. The Anti-entropy algorithm is used by default in some existing SDN controller designs for its stochastic nature and relatively simple implementations. The static or pre-calculated policies represent some research efforts in academia, which aim at taking advantage of traditional optimisation techniques to determine the best controller synchronisation rate or policy. An example of this can be found in [94]. However, a considerable disadvantage comes with employing traditional optimisation techniques is the presence of many ideal and unrealistic assumptions that cannot be met in reality.

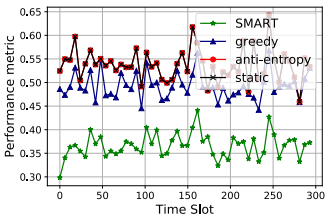


Figure 7.4: Performance at every time slot.

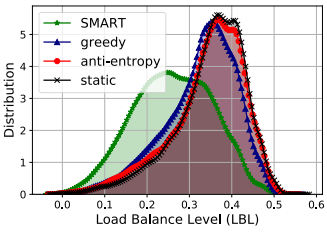


Figure 7.5: Servers' LBL distribution comparison.

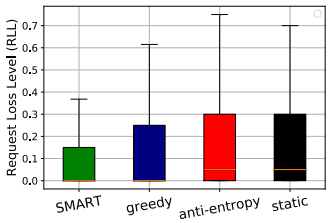


Figure 7.6: RLL comparison (box plot).

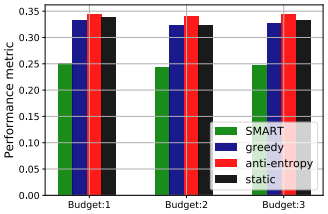


Figure 7.7: Performance comparison with different max. budgets.

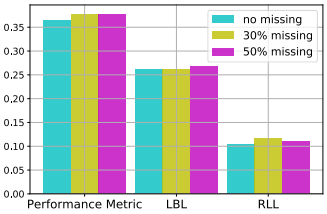


Figure 7.8: SMART performance with missing data.

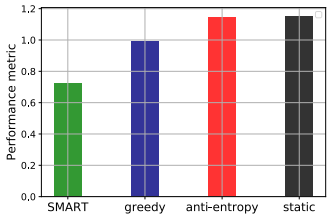


Figure 7.9: Performance comparison over all servers.

7.5.2 Evaluation Settings

The parameters for the simulated networks in our evaluations are summarised in Table 7.1. The independent variables for different scenarios are highlighted in bold fonts to distinguish them from default (control) variables. Since state data are not always available to LG agents, we use the amount of available state data to model the level of LG agents' ability to collect up-to-date makespan information from servers in its LG. For example, if only 50% of state information is available, this means that at each time slot, each server only reports its newest makespan to the LG agent with 50% probability. If the agent does not receive the newest makespan, it uses the last known data (perceived makespan) as an estimation. The default maximum synchronisation budgets at different time slots are drawn from the Gaussian distribution with $\mu = 1$. In Scenario 2, we use constant maximum synchronisation budgets of $1, 2$ and 3 to investigate its impact. In Scenario 4, we keep a constant budget of 2 and evaluate how well various algorithms scale up when the number of services increases from 3 to 10. Moreover, since SMART does not necessarily use all available synchronisa-

tion budget at a time slot, for fair comparisons, we make sure that all algorithms use the same amount of synchronisation budget at each time slot. We assume that the capacity of server processing power is largely determined by its hardware configuration. Therefore, we use the server processing power in Table 7.1 consistently in all experiments. In addition, the requests pattern for different services in all domains are generated by Poisson distribution with $\lambda = 1$. All evaluations are repeated for 100 simulated trajectories using the same parameter settings for accurate results.

As for the settings of DNNs used for designing SMART, the configuration for the LSTM network can be found in Section 7.4.2. The DRL network is implemented as a Multilayer Perceptron [71]. The specifications of the MLP used can be found at the end of Section 7.4.3. The dimension of the input layer of the MLP is of the same size of the state space of the formulated POMDP (see Section 7.3.1). We choose Adam [95] as the optimiser for optimising DNN weights. In addition, we use Rectified Linear Unit (ReLU) [74] as activation functions for all neurons, except for output layers, for which no activation is employed. All DNNs used in SMART are realised using Keras framework [72] built upon Tensorflow [73] DL library.

Remark: The settings described are for evaluation purpose only and there are no assumptions made on any parameters herein.

7.5.3 Evaluation Results

All evaluation results are presented in Fig. 7.4 -Fig. 7.9. Specifically, Fig. 7.4 shows plots of the performance metric $\mathcal{M}$ ($\mathcal{M} = \sum_{i=1}^{n} \mathcal{M}_i/n$, where $\mathcal{M}_i$ is defined in (7.1)) over all experiment time slots. Fig. 7.5 shows the distributions of the LBL of servers' makespans. Fig. 7.6 are box plots of average RLL at every time slot. See Section 7.2.2 for the expressions of LBL and RLL. Fig. 7.4 - Fig. 7.6 correspond to evaluation Scenario 1. For Scenario 2's results in Fig. 7.7, we conduct experiments on increasing maximum allowed synchronisation budgets and compare the perfor-

mance of 4 algorithms under these budget settings. In Fig. 7.8, we show evaluation results of SMART's performance when different amounts of network data are available for training. The plots in Fig. 7.4 - Fig. 7.8 are service-wise results, which are first calculated among servers offering the same service and then averaged over all services. While they are defined as our optimisation goals, it is still interesting to see the overall performance calculated among all servers in the network, which represent the cross-service, overall server performance. Therefore, in Fig. 7.9, we also provide bar plots of the this overall performance metric defined as

$$\mathcal{M}' = \sum_{t=1}^{\Upsilon} \left(\sigma(\mathbf{s}_t) + \sum_{i=1}^{n} \sum_{j=1}^{m} w_{ij}(t) \right) / \Upsilon. \tag{7.27}$$

For Scenario 4, the aim is to test how well SMART performs when the network scales up with more services while the synchronisation budget remains unchanged. Fig. 7.10-7.11 show the average performances metric and rewards as the number of services in the network increase from 3 to 10, respectively. The evaluation results can be summarised as follows.

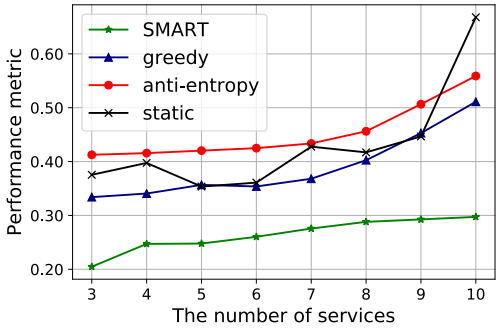


Figure 7.10: Average performance with different # services.

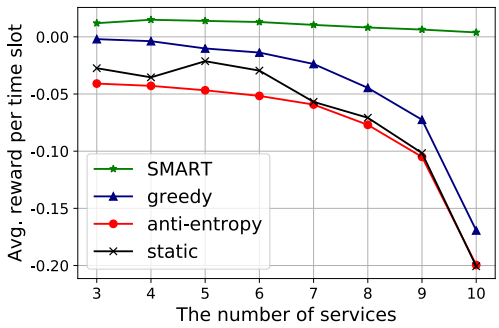


Figure 7.11: Average reward with different # services.

1) Superior performance of SMART: All evaluation results confirm the superiority of SMART. In particular, in Fig. 7.4, we can see that SMART outperforms all other algorithm by sizeable margins consistently. Fig. 7.5 shows the LBL distributions of servers, which represents the load balancing ability in the algorithms'

performance. While the average LBL for SMART's is lower than other algorithms, the distribution curve for SMART is flatter too. Since the metric consists of the LBL and RLL terms, this shows SMART's flexibility to balance different aspects for the better overall performance. Fig. 7.6 reveals SMART's ability to satisfy more requests, as manifested by having the lowest RLL when compared to other algorithms. Interestingly, although SMART is not designed to work in a cross-service way with the metric $\mathcal{M}'$, in which the objective is to minimise the sum of lost makespan and STD of all servers (regardless of the service they are running), it's performance still tops other approaches in this setting, as shown in Fig. 7.9.

2) Robustness of SMART: One main goal of SMART is to make sure that it survives the training data scarcity. This is why we build the LSTM network for handling such potential issues in noisy or hostile environments. Evaluation results show that this design pays off even when significant amount of data are unobtainable. Specifically, Fig. 7.8 shows that SMART's performance only deteriorates by about 3.5% when 50% of the state information w.r.t. the POMDP are unobservable. Moreover, there is only negligible deterioration from 30% to 50% loss of training data. This result not only confirms SMART's survivability when it is hard to obtain data, it also motivates the voluntary reduction in data collections by LG agents to reduce the overheads incurred during data collections.

3) Greater synchronisation budget may not be helpful: When domain controllers have higher synchronisation budget, one would intuitively expect better synchronisation performance. However, this is not evident in our results. In particular, from Fig. 7.7 we can see that when the maximum allowed budget change from 1 to 3, the performance of SMART barely changes. This suggests that SMART does not need a large synchronisation budget to operate.

4) The static algorithm performs almost identically to the anti-entropy algorithm: This suggests that the pre-planned policy based on statistical patterns in reality be-

haves similar to a random policy. This further implies that controller synchronisation policies designed based statistical data without adaptability will perform poorly in dynamic environments.

5) SMART's performance edge increases as network services scale up: Recall that the performance metric employed and the reward defined in the POMDP formulation can be found in Section 7.2.2 and Section 7.3.1, respectively. Note that smaller is better for the former and greater is better for the latter. Since synchronisation budget is kept constant while more services are added, performance deterioration is to be expected. Clearly, as shown in Fig.7.10 and Fig.7.11, it can be seen that SMART results in the most modest performance deterioration compared to the three benchmarks by both measures.

CHAPTER 8

sasRL: An Efficient RL Architecture with Implicit Action Space

---

8.1 Introduction

Chapters 5-7 present our work on control and management of distributed SDN by employing DRL approaches. As our experiences with various mainstream DRL algorithms build up, we realise that although they are powerful tools for solving complicated serial decision-making problems, there are remaining issues with these existing RL algorithms, which require further research efforts. In light of this, in this chapter, we propose a new RL architecture, which is designed to address one of the most cited issues with existing RL algorithms, i.e., the issues of inefficiency in learning. The RL architecture we propose is very generic in the sense that it is not only applicable to network control problems, but also general RL problems. Therefore, the technical contents of this chapter contribute to RL and DRL research communities.

Classic RL [15] methods, which were developed to solve serial decision-making and control problems, have been investigated for decades. For instance, Q-learning algorithm [67], which first appeared in the late 1980s and had since been thoroughly studied and analysed, inspires many successful algorithms and applications. However, due to the lack of general means for function approximation, value functions in Q-learning were estimated in tabular settings or by using simple linear parameterisations. As such, their applicabilities are limitted to some simple problems with

relatively small state-action spaces. In recent years, the advancements in deep learning [96] extend RL to Deep Reinforcement Learning (DRL) [97], for which Deep Neural Networks (DNN) [98] are employed as value function approximators. Due to the DNN's exceptional capability in capturing complex and high-dimensional data structures, DRL methods are empowered to handle many real-world RL problems which involve enormous state-action spaces.

Conventionally, MDP is used to model RL problems. For an MDP, the RL agent jumps between states by taking actions and it collects a reward after transitioning from one state to the next state. The agent maintains a state-action-value (SAV) function (e.g., the Q-value [67]) to estimate the long-term returns of state action pairs. This SAV function is iteratively updated using rewards associated with state transitions. For DRL methods based on MDPs, both model-based and model-free [15] approaches use DNNs as approximators for SAV functions. For example, Google's DQN [59] and a family of algorithms using the actor-critic framework [84] all share this common feature, despite some variations in design details. Then, the agent's policy is developed, either directly or indirectly, according to the learned SAV function.

The SAV function based on the MDP formulation is a convenient choice for developing policies, since actions directly define the behaviours of RL agents. By coupling its behaviours with potential returns, one implicit assumption is that the long term return of an agent is a function of its current state and available actions. Although this is generally true, for many RL problems, the return of a state transition is directly determined by the next state after the state transition, and the action is only indirectly related to the return as it causes such a state transition. Furthermore, for some RL problems, there are potentially several actions that can cause the same state transition. Then, all these actions have the same effect as far as rewards are concerned. As a result, the excessive actions induce extra burdens for training the

DNN function approximators. In this case, having more training data actually exerts adverse impacts on RL. Another source of training inefficiency for the DNN function approximator is the enormity of the state-action space over which the DNN have to generalise for approximating the value function, as the size of the state-action space is combinatorial of already large state and action spaces. All these factors considered, we argue that although SAV function based on MDP can be intuitive and convenient, the aforementioned issues can cause inefficiencies in utilising training data for learning the SAV function in RL.

Aimed to address these issues for RL tasks where rewards are tightly associated with state transitions, we propose an alternative RL paradigm, called the State Action Separable Reinforcement Learning (sasRL), by formulating the RL problem as a modified Markov Reward Process (mMRP, defined in Section 8.3.1). Specifically, we employ a new value function, the *state-transition-value (STV)* function, to estimate returns of state transitions. The STV function takes the current and the next states as input and estimates the return of such a state transiton pair. While the STV function is targeted at addressing the issues discussed above, another added benefit is that in this way, the input to the DNN function approximator only spans the state space. This is in contrast to the case of MDP-based SAV function, whose DNN approximator takes inputs that span both state and action spaces. Our intuition is that the input dimensionality reduction speeds up the training procedure for the DNN approximator, as the agent's actions are not explicitly modelled in mMRP. As such, sasRL develops raw policies in the form of desired next state given the current state. Given this form of the policy, it should be note that the agent still needs to know what action to take to behave in the environment. This is not an issue for RL tasks for which the agent can determine the corresponding action given the desired next state. For tasks where such mappings are not obvious, we use a light-weight deterministic transition model to help the agent determine the action that causes the de-

sired state transition. This transition model is trained on the same data collected for the sasRL training via standard supervised learning procedures. Compared to other model-based RL approaches [15], most of which require full environment model for planning and experience simulations, our light-weight transition model is only required to make predictions of actions, but no other more complex environment dynamics. Therefore, it is much easier to train. In sum, given the same amount of training data, sasRL separates the original RL problem into a less complicated model-free RL problem and a simple supervised learning problem. Our view is that such decoupling procedure is the key factor that leads to better utilisation of the training data and higher RL learning efficiency.

8.2 Related Work

The core idea of sasRL is closely related to the work in [99], where the authors use $Q(s, s')$ (QSS), instead of the commonly used $Q(s, a)$ (QSA) to estimate the value of a state transition. However, this work does not provide any systematic reformulations for RL problems, as such QSS still depends on the traditional MDP formulation. In comparison, our mMRP formulation lays the foundation for a new paradigm to frame RL problems, which enables the analysis of convergence properties and the theoretical discussions on the efficacy of sasRL based on such mMRP formulations. In addition, there are noticeable differences in implementations between sasRL and QSS. For instance, sasRL uses the combination of the nearest heighbour-based regularisation loss and a light-weight transition model to ensure the feasibilities of state-state and state-action-next state transitions, whereas QSS employs a more complex setting where 2 environment dynamic models are trained.

Moreover, sasRL is inspired by the idea of combining the strengths of both model-based and model-free RL techniques to improve training efficiency. In this

regard, our work is closely related to [100] in that [100] trains a low-dimensional encoding of the environment, and such a encoding module is used for planning. However, the model employed in [100] is very heavy, which requires the modelling of all elements (transition, reward dynamics, etc.) of the environment, in addition to the parameterised value function. In practice, we find that the low-dimensional encoding of the state space is rather difficult to train. Results from exisiting literatures are not conclusive on if using human insights to aid such encoding design would help DNNs better capture the state structures, as echoed by [101]. Similarly, other works on separating the model-free and model-based learning in RL focus on learning state, action, and/or reward representations/dynamics, separately. The central idea of these approaches is that modularizations of RL tasks have the benefit of potential transfer learning and improved learning efficiency. For example, [102] decouples the RL problem into a state dynamic learning component and a reward function learning component. The learned state dynamic model is shown to be transferable to new scenarios. The method proposed in [103] offers a simple yet effective way to obtain a sparse DNN representation of the training data to assist the DRL agent in better understanding useful and pertinent dynamics in RL tasks. On the other hand, the works in [104, 105] investigate the embeddings of action space from theoretical and practical perspectives. Moreover, the Value Prediction Network (VPN) [106] avoids the challenging task of modelling the full environment by only focusing on predicting value/reward of future states. The VPN's model-based part learns the dynamics of the abstract state transitions, while its model-free part predicts rewards and values from the abstract state space. In addition, [107] associates RL tasks with a "pseudo-reward" which encourages the agent to learn features/representations from the input data. The authors argue that the auxiliary task of learning these features aligns with the agent's long-term goal, which is to maximise the accumulated extrinsic rewards. Therefore, these auxiliary tasks incentivized by the pseudo-reward help the agent

develop more useful representations of RL tasks.

Although these works use a combination of model-based and model-free RL techniques like ours, we explore the problem from different perspectives. The most distinctive difference here is that they rely on embedding and/or representation learning techniques. In contrast, we do not use any dimension reduction techniques. Instead, we decouple the action space from the model-free RL procedure and build a separate light-weight transition model which is trained via supervised learning.

8.3 Problem Formulation

In this section, we introduce the problem formulation under the sasRL framework, which uses a modified Markov Reward Process (mMRP) for modelling the RL problem. Then, Section 8.3.2 defines the value function, i.e., the STV function, for policies based on the mMRP formulation. To update the STV function and the policy, we discuss the policy-gradient based method we employ in Section 8.3.3. Then, Section 8.3.4 discusses how the light-weight transition model in sasRL is trained to predict actions based on the policy derived from the trained STV function. Finally, Section 8.3.5 discusses the limitations of the mMRP-based problem formulation under sasRL.

8.3.1 The Modified Markov Reward Process (mMRP)

In the high level, we aim to formulate the RL problem in a way that the action space is decoupled from the value function. Then, the value function's input only spans the state space. On the other hand, the action space, which characterises the dynamics of the agent's interaction with the environment, is separately modelled. To this end, we propose to use the modified Markov Reward Process (mMRP) as the basis for modeling RL problems. In general, the Markov Reward Process (MRP)

can be regarded as a Markov chain with state values added. Specifically, our mMRP is formally defined by a 4-tuple $(\mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma)$ as follows.

- $\mathcal{S}$ is state set;
- $\mathcal{P}$: $\mathcal{P}_{ss'} = P[s_{t+1} = s' | s_t = s]$ is the state transition probability matrix, where $s, s' \in \mathcal{S}$;
- $\mathcal{R}$: $\mathcal{R}_{ss'} = \mathbb{E}[r_{t+1} | s_t = s, s_{t+1} = s']$;
- γ is the discount factor.

Note that for an MDP, the transition probability is denoted by $\mathcal{P}$: $\mathcal{P}^a_{ss'} = P[s_{t+1} = s' | s_t = s, a_t = a]$. Although $\mathcal{P}^a_{ss'}$ and the transition probability matrix $\mathcal{P}_{ss'}$ for the mMRP are both jointly decided by the agent's actions and the environment dynamics, the difference is that the transition probability in mMRP does not condition on any specific actions. In a way, the transition probability matrix for the formulated mMRP can be viewed as induced by an agent exploring the MDP by taking various actions. Then, the mMRP is built by the agent's exploration history, from which the heighbouring states of any given state, the transition probability into the heighbouring states ($\mathcal{P}_{ss'}$), and the associated rewards ($\mathcal{R}_{ss'}$) are identified. On the other hand, the mMRP differs from a typical MRP in that rewards only depend on current states for the MRP formulation, i.e., $\mathcal{R}_s = \mathbb{E}[r_{t+1} | s_t = s]$; while for our mMRP, the reward depends on both the current and the next state of a state transition.

For the mMRP formulation, one implicit assumption is that if two actions a_1 and a_2 cause the same state transition $(s \to s')$, it is assumed that $\mathbb{E}[r_{s,a_1}] = \mathbb{E}[r_{s,a_2}]$, where $r_{s,a}$ is the reward for the state action pair (s, a). Another implicit assumption here is that the environment is deterministic, as there is no guarantee that the a_1 and a_2 cause the state transition $(s \to s')$ in a stochastic environment. See more discussions on this in Section 8.3.5.

Based on the formulated mMRP, the policy of an RL agent specifies the reachable next state (s') given the current state s. We assume that the policy is deterministic, denoted by $\mu(s)$, i.e., $\mu : \mathcal{S} \rightarrow \mathcal{S}'$, where $\mathcal{S}'$ is the sub-space of $\mathcal{S}$ which includes the reachable next states from the current state. Therefore, without loss of generality, we assume that the agent's policy based on the mMRP always produces reachable next state from the given current state, see the discussion in Section 8.3.5 for how this can be achieved in practice. Finally, the long-term value function for the formulated mMRP under policy μ is denoted by $V^{\mu}(s)$, which is defined by $V^{\mu}(s) = \mathbb{E}[r_1 + \gamma r_2 + \ldots | s_0 = s, \mu]$, where $r_t \in \mathbb{R}$ is the reward at time t, s_0 is the initial state.

8.3.2 The State-Transition-Value (STV) Function under the Given Policy

Based on the mMRP, we denote the *state-transition-value (STV) function*, which quantifies the long-term return of a state transition $(s \rightarrow s')$ under policy μ, as follows,

$$\Phi^{\mu}(s, s') = r_{s,s'} + \gamma \Phi^{\mu}(s', \mu(s')), \tag{8.1}$$

where $r_{s,s'}$ is the reward for transition from s to s'. By definition, $V^{\mu}(s) = \Phi^{\mu}(s, \mu(s))$.

It can be seen that the STV function in (8.1) is defined in the format of a Bellman equation, which offers a natural way to conduct value iterative update to make it more accurate. Similar to the update process of SAV function, the STV function is learned model-free and it is suitable for off-policy learning [108]. The difference is that the update of STV function does not directly make use raw observation data, i.e., $(s, a, s', r_{s,a})$; but rather first use these raw data to construct the mMRP and then use the $(s, s', r_{s,s'})$ data obtained from the mMRP instead. In this sense, the employment of STV function for learning policy instead of the SAV function can be viewed as a way for reducing variance for off-policy learning. In particular, the

formulation of mMRP reduces the variance introduced by similar actions and the induced similar state transitions, which helps better generalise the return dynamics of the RL problem.

Based on the STV function, the return of the policy μ, $J(\mu)$, which corresponds to the optimisation goal for the formulated RL problem, is

$$J(\mu) = \int_{s\in\mathcal{S}} \rho^{\mu}(s)\Phi^{\mu}(s,\mu(s))\mathrm{d}s, \tag{8.2}$$

where $\rho^{\mu}(s)$ is the discounted state distribution under μ [109, 110]. Let the initial state and the initial state distribution be s_0 and $p_0(s_0)$, respectively. Then, $\rho^{\mu}(s) = \int_{s_0\in\mathcal{S}} \Sigma_{t=0}^{\infty}\gamma^t p_0(s_0)p(s_0 \rightarrow s,t,\mu)\mathrm{d}s_0$, where $p(s_0 \rightarrow s,t,\mu)$ denotes the probability of transitioning from state s_0 to state s after t steps. For the mMRP, like the initial transition probability matrix, the initial state distribution is determined by past observation data collected.

8.3.3 Policy-gradient-based Learning

The STV function defined in (8.1) is the basis for deriving policies for RL problems. In order to obtain an accurate STV function, it is iteratively updated using (s,s',r) tuples. In addition to updating the STV function, which may be directly used for generating policies for some discrete problems with small state space size. For problems with larger state spaces, a more intuitive way is to parameterise the policy and update its parameters, so that policies can be directly generated. This approach is referred to as policy-gradient method [111]. For generality, we assume that STV function and policy are parameterised by parameter set $\boldsymbol{\kappa}$ and $\boldsymbol{\theta}$, denoted by $\Phi_{\boldsymbol{\kappa}}$ and $\mu_{\boldsymbol{\theta}}$, respectively. For brevity, we use $\Phi_{\boldsymbol{\kappa}}$ and Φ, $\mu_{\boldsymbol{\theta}}$ and μ interchangeably. Next, we discuss how $\Phi_{\boldsymbol{\kappa}}$ and $\mu_{\boldsymbol{\theta}}$ are updated.

First, the STV function parameter set $\boldsymbol{\kappa}$ is updated by minimising the mean

squared TD(0) error [15] defined as $L = \left(r_{ss'} + \gamma\Phi^{\mu}_{\boldsymbol{\kappa}}(s', \mu(s')) - \Phi^{\mu}_{\boldsymbol{\kappa}}(s, s')\right)^2$. Second, recall that the return of the policy μ is defined as $J(\mu)$ in (8.2). Therefore, the policy-gradient method aims to maximise $J(\mu_{\boldsymbol{\theta}})$ by performing gradient ascent on parameter set $\boldsymbol{\theta}$ using the policy gradient $\nabla_{\boldsymbol{\theta}} J(\mu_{\boldsymbol{\theta}})$. In the following theorem, we present how such policy gradient is computed.

Theorem 8.1. If $V^{\mu_{\boldsymbol{\theta}}}(s)$ and $\nabla_{\boldsymbol{\theta}} V^{\mu_{\boldsymbol{\theta}}}(s)$ are continuous function of $\boldsymbol{\theta}$ and s, then the following holds,

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} J(\mu_{\boldsymbol{\theta}}) &\approx \int_{s\in\mathcal{S}} \rho^{\beta}(s)\nabla_{\boldsymbol{\theta}}\mu_{\boldsymbol{\theta}}(s)\nabla_{s'}\Phi^{\mu_{\boldsymbol{\theta}}}(s, s')\mathrm{d}s \\ &= \mathbb{E}_{s\sim\rho^{\beta}}\left[\nabla_{\boldsymbol{\theta}}\mu_{\boldsymbol{\theta}}(s)\nabla_{s'}\Phi^{\mu_{\boldsymbol{\theta}}}(s, s')|_{s'=\mu_{\boldsymbol{\theta}}(s)}\right],\end{aligned} \tag{8.3}$$

where β denotes the behaviour policy [108] used to generate training data, and $\rho^{\beta}(s)$ is the discounted distribution of states under the behaviour policy.

Let $\alpha_{\boldsymbol{\theta}}$ be the learning rate for updating $\boldsymbol{\theta}$. Then, the policy parameter is updated as follows,

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_{\boldsymbol{\theta}}\nabla_{\boldsymbol{\theta}}\mu_{\boldsymbol{\theta}}(s)\nabla_{s'}\Phi^{\mu_{\boldsymbol{\theta}}}(s, s'). \tag{8.4}$$

While updating policy parameters, other types of loss maybe considered as well, depending on the specific policy-gradient-based implementation employed. See Section 8.3.5 for more discussions. Figure 8.1 demonstrates the policy-gradient based updates for the policy and STV function in sasRL.

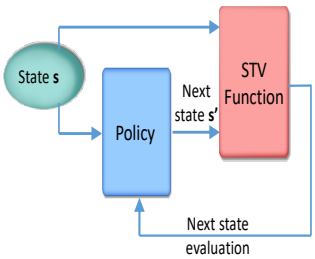


Figure 8.1: Policy and STV function updates by policy-gradient methods.

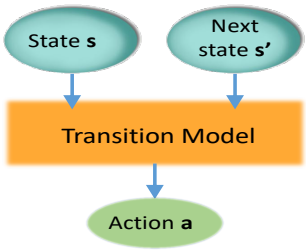


Figure 8.2: State transition model training by supervised learning.

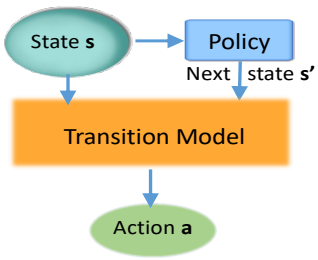


Figure 8.3: sasRL in operation with trained policy and transition model.

8.3.4 Deterministic State Transition Model and its Training

The mMRP formulation for sasRL does not explicitly model actions of the agents, and the policy developed based upon it indicates the target next state (s') given the current state (s). For RL tasks with state space consisting of hand-crafted features, it can be straightforward for the agent to determine the action that causes the state transition ($s \rightarrow s'$).

In other RL tasks where action cannot be determined from ($s \rightarrow s'$), we build a light-weight deterministic transition model to help the agent determine the action that can cause the state transition ($s \rightarrow s'$). This deterministic transition model can be represented by a DNN whose parameters are optimised using standard supervised learning techniques. Formally, define the deterministic transition model $\tau_{\boldsymbol{\omega}} : \mathcal{S} \times \mathcal{S} \rightarrow \mathcal{A}$, which is parameterised by a set of DNN weights $\boldsymbol{\omega}$. Then, we train the model by minimising the prediction error, $\mathcal{L}_{\boldsymbol{\omega}}$, by using the samples (s, s', a, r) which are collected for the RL training. In particular, the loss $\mathcal{L}_{\boldsymbol{\omega}}$ is defined as follows,

$$\mathcal{L}_{\boldsymbol{\omega}} = L(\tau_{\boldsymbol{\omega}}(s, s'), a), \tag{8.5}$$

where the type of loss L depends on the representation of the action vector (e.g., L uses binary cross entropy loss if the action vector consists of only 0 and 1 elements). Fig. 8.2 shows the supervised learning process for training the transition model. After all components of sasRL, i.e., the parameterised policy, the parameterised STV function, and the transition model, are trained, Fig. 8.3 describes sasRL in operation.

8.3.5 Limitations of the mMRP Formulation

The mMRP-based problem formulation has several assumptions, some may require special mechanisms in practical implementations, while others may limit sasRL's applicabilities in certain problems. Here, we briefly discuss these limitations.

First of all, although the learning of the STV function is model-free, which does not rely on any assumptions on environment nor its dynamics, it is not difficult to see that the policies produced by the STV function are only meaningful if the environment is deterministic. This is because in a stochastic environment, the predicted action by the transition model cannot guarantee the desired state transition given by the STV function.

Second, as briefly mentioned at the end of Section 8.3.1, depending on how sasRL is implemented, it is possible that the policies derived by the STV function is not feasible due to the limitations of the agent's actions. For example, in the grid world exit problem, if the state of the mMRP is defined as the agent's location. Then, it is possible that the agent cannot reach the desired next location (next state), which is determined by the learned policy, by taking a single action. Therefore, such limitations coming from the environment dynamics should be incorporated in the process of policy generation based on the STV function. In our embodiment of the sasRL in Section 8.5, we use the regularisation loss for regularising outputs of the the policy network, which are desired next states. There are various ways to achieve this, depending on how the sasRL architecture is implemented.

Third, since it is possible that multiple different actions can cause the same state transition, this creates some challenges for training the transition model. In particular, when the transition model is trained in a way that multiple targets (actions) are provided for the same input (current and next states), after training the transition model may learn to output an "average" of the targets it had seen. The impacts of this can vary, depend on the nature of the RL problems. It may be the case that the predicted actions of the transition model require regularisation, which is similar to the procedures mentioned in the paragraph above for regularising the generated policies of the STV function. In our experiments, we take an alternative approach to only train the same state-next state pair for only one target action. This is feasible as

a single action is sufficient to trigger the desired state transition in the deterministic environment.

8.4 Convergence Analysis

We conduct convergence analysis for sasRL based on existing convergence studies developed for the classic Q-learning algorithm. In the high-level, our analysis first identifies the similarity of the value function update procedure of Q-learning and policy-gradient based method. Then, we employ similar convergence analysis derived for Q-learning to compare the convergence properties of the SAV and STV functions. Finally, we compare the convergence properties of SAV and STV functions. It should be noted that all theoretical analyses in this section assume discrete state and action spaces for theoretical tractability. Nevertheless, as we show in Section 8.6, sasRL's yield superior performance for the three continuous RL scenarios tested.

The convergence analysis of the Q-learning algorithm is provided in [112]. We base part of our analysis on the main theorem from this work, which is stated as follows.

Theorem 8.2. Let $Q_t(s,a)$ and $Q^*(s,a)$ denote the t-th iteration of the Q-function during the update process as defined in [67], and the optimal Q-function, respectively. Assume that the conditions set out in [112] are met. Then, the following relation holds asymptotically with probability one: $|Q_t(s,a)\text{-}Q^*(s,a)| \leq \frac{B}{t^{R(1-\gamma)}}$, for some suitable constant $B > 0$ when $R(1-\gamma) < 1/2$. Here, $R = p_{\min}/p_{\max}$, where $p_{\min} = \min_{(s,a)} p(s,a)$, $p_{\max} = \max_{(s,a)} p(s,a)$, and $p(s,a)$ is the sampling probability of (s,a).

For our analysis, let $\mathcal{S}$ and $\mathcal{A}$ be the state and action spaces of the RL problem under the MDP formulation. Then, let $\beta(s)$ be the given sampling policy which

generates data for updating value function and policy parameters. At each time step, an action is chosen according to β and the state transition $s + a \rightarrow s'$ takes place. Let W be the total time steps for one data generation trajectory. All W state transitions are recorded in forms of W (s, a) and (s, s') pairs. Denote by $\nu(s, a)$ and $\nu(s, s')$ the number of occurrence of (s, a) and (s, s') pairs, respectively. Then, $p^{\beta}(s, a) = \mathbb{E}[\nu(s, a)/W]$ and $p^{\beta}(s, s') = \mathbb{E}[\nu(s, s')/W]$ are the the probabilities of recording (s, a) pairs under the sampling policy β. In addition, let $p^{\beta}_{\min\,/\,\max}(s, a)$ and $p^{\beta}_{\min\,/\,\max}(s, s')$ be the minimum/maximum $p^{\beta}(s, a)$ and $p^{\beta}(s, s')$, respectively. We define $R_1 = p^{\beta}_{\min}(s, a)/p^{\beta}_{\max}(s, a)$ and $R_2 = p^{\beta}_{\min}(s, s')/p^{\beta}_{\max}(s, s')$. Then, we summarise the convergence comparison results for SAV and STV functions as follows.

In order to build our convergence analysis on Theorem 8.2, we first identify the conditions upon which the parameter update process in policy-gradient based methods is similar to the Q-function update process in Q-learning. Then, we can employ Theorem 8.2 for further analysis which compares the convergence properties of SAV and STV functions, both of which use policy-gradient methods for parameter updates.

In particular, we start by providing an alternative view on the Q-function update for the classic Q-learning algorithm. In classic Q-learning algorithm where Q-tables are used to keep track of Q-values during the update process. The Q-function is updated as follows,

$$Q(s, a) = (1 - \alpha)Q(s, a) + \alpha\big(r + \max_{a'} Q(s', a')\big). \tag{8.6}$$

We take an alternative view on the Q-learning update process by reforming (8.6),

which renders,

$$Q(s,a) \approx Q(s,a) + \alpha\big(r + \max_{a'} Q(s',a') - Q(s,a)\big)\nabla_{(s,a)}Q(s,a). \tag{8.7}$$

Note that (8.7) is obtained if we view the values of the Q-table as parameters of the Q-function and the loss function is defined as $L = \big(r + \max_{a'} Q(s',a') - Q(s,a)\big)^2/2$. The approximation is due to the omission of the term related to $\nabla_{(s',a')}Q(s',a')$, which still preserves the convergence property of tabular Q-learning in certain cases [15]. Then, the Q-learning update procedure is essentially updating the parameters of the Q-function approximator (Q-table) to make it more accurate. Therefore, this is on a par with the value function update processes used by both STV and SAV functions. We recognise that there are differences between the approximations by the Q-table (in Q-learning) and by DNNs (parametrized STV and SAV functions). Nevertheless, based on the assumption that both the Q-table and parametrized STV and SAV function update processes achieve exact approximation of the corresponding value functions in the asymptotic sense, their parameter update procedures are identical. Another important characteristic in the Q-function update procedure is that the greedy policy is employed to estimate the Q-value at the next state s', i.e., $a' = \operatorname{argmax}_{a'} Q(s',a')$. For the policy-gradient based updates of STV and SAV functions, the equivalent update procedure requires that the value estimation for the next state is based on the action that maximises the value function.

Based on the above, for each update of the value (SAV or STV) function in policy-gradient based methods, we assume that the policy parameters generate policies that maximise the corresponding value function, and employ Theorem 8.2 to analyse the convergence properties of the value function update in policy-gradient methods.

Specifically, let $f_1(s,a)$ and $f_2(s,s')$ denote the parameterised SAV and STV

function, respectively. Then by Theorem 2, $|f_1(s,a) - f_1^*(s,a)| \leq \frac{B}{t^{R_1(1-\gamma)}}$ and $|f_2(s,s') - f_2^*(s,s')| \leq \frac{B}{t^{R_2(1-\gamma)}}$ hold with probability one, where $f_1^*(s,a)$ and $f_2^*(s,s')$ are the optimal SAV and STV function, respectively. We compare the convergence properties of SAV and STV functions by comparing $\frac{B}{t^{R_1(1-\gamma)}}$ and $\frac{B}{t^{R_2(1-\gamma)}}$. In particular, since T is the convergence time for updating the SAV function, we solve $\frac{B}{T^{R_1(1-\gamma)}} = \frac{B}{t^{R_2(1-\gamma)}}$ for t, which is defined as the convergence time for updating the STV function. This renders $t = T^{R_1/R_2}$. Therefore, the convergence time for updating STV function is $O(T^{1/k})$ where $k = R_2/R_1$. This means that in theory, to achieve the same level of regret for the RL problem, the number of updates of the value function using the SAV function or the STV function is T or T^{R_1/R_2}, respectively. As such, if $R_2 > R_1$, using STV is more advantageous than using SAV.

Efficient Training Condition ($k > 1$): The analysis above reveals the key threshold of $k = 1$, i.e., when $k > 1$, the value function update convergence under the mMRP formulation is faster than that under the MDP formulation for RL problems. Recall that k is closely related to the behaviour (sampling) policy, which is used to collect data for off-policy [108] updates of the value functions. In this paper, we argue that sasRL is most suitable for those problems where the action space is large and multiple actions can trigger the same or similar state transitions. Indeed, many RL problems of such a nature result in $k > 1$ under the given behaviour policies (see Section 8.6.3 for more discussions).

8.5 The Embodiment of sasRL

To demonstrate how sasRL can be implemented in practice, here we implement an instance of sasRL using the actor-critic framework [84], which is suitable for the policy-gradient based update process described in Section 8.3.3. The actor-critic framework offers a natural way to concurrently optimise policy parameters and the

Algorithm 3: sasRL training procedure

// `policy training`
Initialize STV function (critic) parameters $\boldsymbol{\kappa}$ and policy (actor) parameters $\boldsymbol{\theta}$;
Initialize delayed parameters $\boldsymbol{\kappa}' \leftarrow \boldsymbol{\kappa}$, and $\boldsymbol{\theta}' \leftarrow \boldsymbol{\theta}$;
Initialize replay buffer with (s, s', a, r) tuples generated by behaviour policy β.
while *the maximum number of iterations not reached OR not converged* **do**
    Pull a random minibatch of (s, s', a, r) from the replay buffer;
    Update the critic parameters $\boldsymbol{\kappa}$ by minimising the mean squared TD(0) error: $L = \left(r_{s,s'} + \gamma \Phi^{\mu_{\boldsymbol{\theta}'}}_{\boldsymbol{\kappa}'}(s', \mu_{\boldsymbol{\theta}'}(s')) - \Phi^{\mu_{\boldsymbol{\theta}}}_{\boldsymbol{\kappa}}(s, s')\right)^2$;
    Update the actor parameters $\boldsymbol{\theta}$ according to (8.4);
    Soft update the delayed parameters: $\boldsymbol{\kappa}' \leftarrow \varepsilon\boldsymbol{\kappa} + (1-\varepsilon)\boldsymbol{\kappa}'$, $\boldsymbol{\theta}' \leftarrow \varepsilon\boldsymbol{\theta} + (1-\varepsilon)\boldsymbol{\theta}'$;
    if *current policy is evaluated* **then**
        store new (s, s', a, r) samples collected from roll-out episodes in replay buffer.
    end
end
// `transition model training (optional) using data from the replay buffer`
while *the transition model training not converged* **do**
    Pre-process (see Appendix B for details) minibatch of data for training the transition model;
    Update transition model parameter $\boldsymbol{\omega}$ by minimising loss $L(\tau_{\boldsymbol{\omega}}(s, s'), a)$.
end

STV function parameters. There are several advantages for adopting an actor-critic approach, compared to more straightforward methods such as the Monte-Carlo REINFORCE [113] algorithm. The most obvious one is that actor-critic methods are intuitive as policies can be directly derived using the trained DNN with policy parameters, which is especially useful for RL tasks with large action spaces. In addition, the bias introduced by the actor-critic methods' bootstrapping procedure has been shown to reduce variance and accelerate learning [15]. Moreover, since actor-critic methods do not require whole trajectories, they can be implemented online or for non-episodic problems.

The structure of the actor-critic implementation of sasRL is similar to the update

process shown in Figure 8.1. Specifically, the actor and critic correspond to the policy and STV function, respectively. During training, the actor-critic model is used to concurrently update $\boldsymbol{\kappa}$ and $\boldsymbol{\theta}$ which are the weights of the actor and the critic, respectively. The training samples generated by the behaviour policy β are organised in (s, s', a, r) tuples. These tuples are stored in the replay buffer [59] to be used for training multiple times. The training procedure of sasRL is summarised in Algorithm 3. Recall that in Section 8.3.1, we assume that the policy μ always produces reachable next state. In practice, this is achieved by introducing a regularisation loss during the training of the actor to force it into producing valid next states. In particular, let $\hat{s}$ denote the output of the actor, given the input state s, i.e., $\hat{s} = \mu(s)$. Then, if $\hat{s}$ is not a reachable next state, it is mapped to the cloest reachable next state by the mapping g, which is based on the nearest heighbour mapping using L_2 distance. The mapping can be expressed as follows

$$g(\hat{s}) = \arg\min_{s' \in \mathcal{S}'} |s' - \hat{s}|_2. \tag{8.8}$$

Then the regularisation loss is defined as $L_{\text{reg}} = |s' - \hat{s}|_2$. Then, the policy parameter update rule in (8.4) is changed to

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_{\boldsymbol{\theta}} \big(\nabla_{\boldsymbol{\theta}} \mu_{\boldsymbol{\theta}}(s) \nabla_{s'} \Phi^{\mu_{\boldsymbol{\theta}}}(s, s') + \iota \nabla_{\boldsymbol{\theta}} L_{\text{reg}} \big), \tag{8.9}$$

where ι is the weighting factor.

The actor-critic part of sasRL is model-free, since both the actor and the critic learn directly from samples without explicitly requiring any modellings of the mMRP. The actor and critic in this sasRL embodiment are both implemented as multi-layer perceptrons (MLPs) [114]; their specifications are documented in Section B.2 in Appendix B. As for the optional light-weight transition model in sasRL, it is trained using (s, s', a) tuples. The input to the transition model is (s, s') pair and the output

is the action a that causes this state transition. Since the transition model is independent from the actor-critic network, sasRL has the option of training the transition model separately from that of the actor-critic network. Alternative, the transition model can be trained together with the actor-critic network, using the same mini-batch of (s, s', a, r) data in every iteration. The transition model is trained using conventional supervised learning procedure to minimise the loss specified by (8.5). The transition model is also built as an MLP.

8.6 Experiments

The main objective of our experiments is to evaluate the performance of sasRL which is pertinent to its structure and the mMRP problem formulation. Therefore, we strive to minimise the influence of other factors such as the design of the DNN and its hyper-parameters. For these considerations, our experiment scenarios do not involve heavy imagery or high-dimensional state definitions, for which extra efforts in parameter tuning and model design are needed. Since we argue that the formulation based on mMRP is more efficient in learning the value function, we compare the performance of sasRL with state-of-the-art DRL solutions based on the MDP formulation. In this chapter, we only provide high level descriptions of our experiments here, with further details in Appendix B.

8.6.1 Baselines

1). *DDPG: Deep Deterministic Policy Gradient [86]* is a model-free and off-policy DRL algorithm based on the deterministic policy gradient theorem [109]. DDPG employs several techniques to improve data usage efficiency and to stabilize the DRL training process, such as replay buffers and the soft parameter update procedure.

2). *SAC: Soft Actor-Critic [115]* is a model-free and off-policy RL algorithm

for learning stochastic policies. SAC is partially inspired by the desire to address DDPG's brittleness and hyperparameter sensitivity. To this end, SAC maximises the trade-off between the policy's performance and its randomness, which is measured by entropy.

3). *PPO: Proximal Policy Optimisation [116]* is a relatively light-weight, model-free, and on-policy RL algorithm for learning stochastic policies. The core idea is to ensure the policy update does not go too far, while striving for greater improvements per update. In particular, PPO relies on the clipping of the objective function, among other techniques, to achieve this goal. Another distinctive feature of PPO is that it requires consecutive data samples (i.e., trajectories) for policy update.

For fair and consistent comparison, we use reference implementations of these baseline algorithms from the Stable Baselines project [117].

8.6.2 Scenarios

1). *Grid world exit problem.* First, we consider a continuous grid world exit problem where the agent tries to avoid the landmine and exit the grid as soon as possible. For this problem, the state is defined as the agent's current location. Unlike traditional grid world problem where the agent's actions are discretized as jumping from squares to squares, the scenario we consider is a continuous control problem, as the agent is allowed to move freely within certain vicinity up to a limit for each time step. The continuity in the action space increases the complexity of the SAV function and also results in potentially more actions that cause the same state transition. For each time step, the agent experiences a large negative reward for hitting a mine location, or a large positive reward for moving into the exit location. In addition, a small negative reward applies at all time steps to penalize time consumption (because the agent is expected to exit as soon as possible).

2). *Berzerk-like game.* The second scenario is a simplified berzerk game [118]

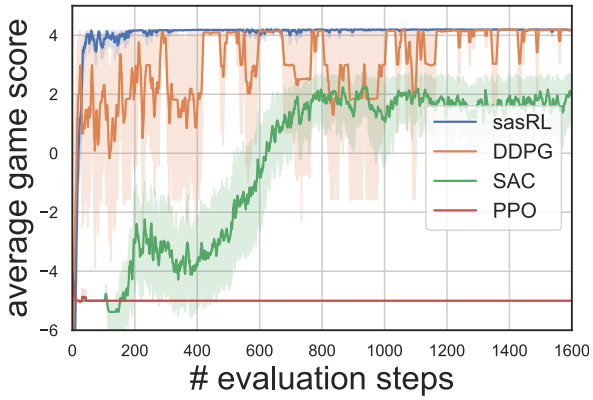


Figure 8.4: Comparative evaluation: Gridworld exit.

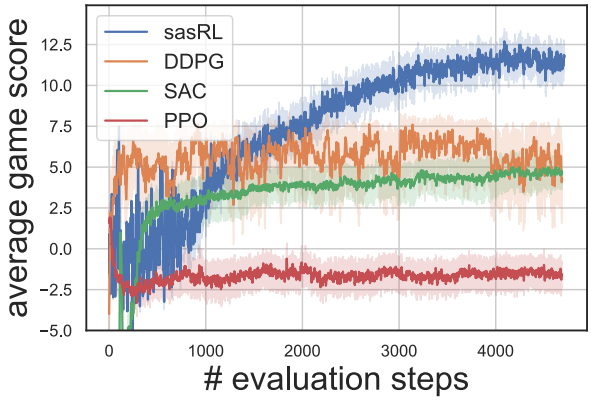


Figure 8.5: Comparative evaluation: Berzerk.

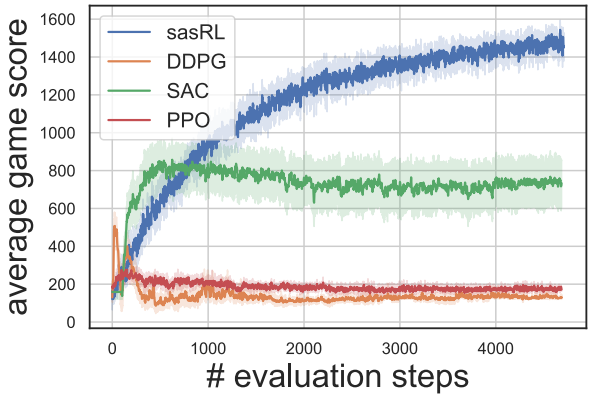


Figure 8.6: Comparative evaluation: Slot machine.

where the agent navigates through a maze with obstacles (walls) and patrolling robots. The walls are fixed and the robots patrol on routine routes. The goal of the agent is to kill as many robots as possible by firing bullets while it tries to exit

the room as soon as possible. At each time step, the agent is allowed to move freely within certain range, and one bullet is fired towards the direction of travel to kill the robot on its trajectory if any. For this problem, a state consists of the agent's location, the robots' locations, and the exit locations. The action of the agent is to move around its current location within the given limit. The reward of the agent is determined by a combination of factors detailed in Appendix B.

3). *The slot machine gambling game.* The third experiment considers the gambling game of a slot machine. A slot machine consists of several reels with printed symbols. The player spins the reels and receives a payout when all reels stop spinning. The payout is determined by the symbols on display on reels; see Appendix B for the detailed calculation of the payout. Note that the player has no knowledge of how symbols are arranged on reels and cannot see the symbols before all reels stop. For this scenario, the state is defined as the symbols on display when all reels stop. For finer granularity of control, the player is allowed to decide for how long each reel spins. Therefore, the action is defined as the timer values set for all reels. The reward is defined as the payout amount.

8.6.3 Discussions on the Experiment Scenarios

One common feature of these experiment scenarios is that the reward of a state transition is determined by the state transition, while the action is only relevant as it causes the state transition. These are the scenarios that we argue sasRL would be more efficient than the RL algorithms based on the MDP formulation. Moreover, it is likely that multiple actions can cause the same state transition. As a result, our empirical results show that $k \approx 2.72$ in this example under a random behaviour policy for collecting training samples. Therefore, faster convergence rate is expected for value function update under the mMRP. For the grid world and berzerk scenarios, due to the definition of state space, once the next state is given, the agent can directly

determine the corresponding action to take. Whereas for the slot machine scenario, the agent cannot know the action that can cause a desired state transition, since the inside structure of the reel is not available to the agent. In this case, the transition model is employed to help the agent understand the state transition dynamics in relationship to actions. Although action spaces are continuous in all scenarios, there are certain limits on each action. For example, the agent is only able to move within certain vicinity in any time step. In sasRL, to ensure that the actor generates feasible next state, its output passes through a deterministic nearest neighbour based mapping which maps the potential out-of-range next state to the feasible next state.

8.6.4 Comparative Evaluation and Results

Fig. 8.4-8.6 show the comparisons of sasRL against DDPG, SAC, and PPO for the three evaluation scenarios. In these figures, the horizontal axis is the number of evaluation steps, while the vertical axis is the accumulated reward collected for game episodes played using the developed policy. The evaluation takes place every several gradient update steps (see Appendix B for details). In our experiments, 10 instances (initialization of all DNN parameters) of these algorithms are trained and evaluated. For each evaluation episode, the maximum number of steps (cap) applies if the agent does not complete the episode when this cap is reached. In these figures, the solid lines and shaded areas are the average and the range (minimum/maximum) of the accumulated reward over all instances.

These evaluation results show that sasRL's performance is consistently superior when compared to the baselines. In particular, PPO fails in all three scenarios. The SAC algorithm produces the most stable results on average, which is expected since SAC is designed to address the brittle convergence problem that is seen in other RL algorithms. In contrast, although DDPG outperforms SAC in grid world and berzerk scenarios, its performance is unstable, which is echoed in [115]. However, despite

the inferior performance, DDPG shows fastest convergence rate in the berzerk scenario, and its convergence performance is comparable to sasRL in the grid world scenario. In summary, other than the grid world scenario where DDPG performs comparably to sasRL, sasRL outperforms all baselines for three evaluation scenarios.

8.6.5 Ablation Evaluation on Action Space Granularity

All evaluation scenarios we consider so far have continuous action spaces. Another interesting aspect to investigate is the impact of the granularity of state transitions on the performance of sasRL. To this end, we change the action space of the behaviour policy from continuous to discrete for generating training samples. Specifically, we use two levels of action granularity with the discrete action space, a coarse-grained, and a fine-grained, to generate training samples. Details on the definitions for two levels of action granularity can be found in Appendix B.

The experiment results are shown in Fig. 8.7-8.9. Overall, they show that training samples from continuous action space enables sasRL to have better performance. This is expected as fine-grained actions result in a diverse state transition sample pools, from which the DNN function approximator can reveal more structural details of the reward dynamics. In the case of discrete action space, the performance of sasRL under the "coarse" and "fine-grained" action spaces are comparable, for the grid world and the berzerk scenarios. A similar trend can be observed in both scenarios that the training curves for the "coarse" cases experience more fluctuations, whereas they steadily go upwards for the "fine-grained" cases. Therefore, fine-grained state transitions tend to stabilize training. In addition, the gap between the "coarse" and "fine-grained" discrete action cases in the slot machine scenario suggests that the granularity of action space significantly influences sasRL's ability to learn for some problems. Another interesting phenomenon is that for the slot ma-

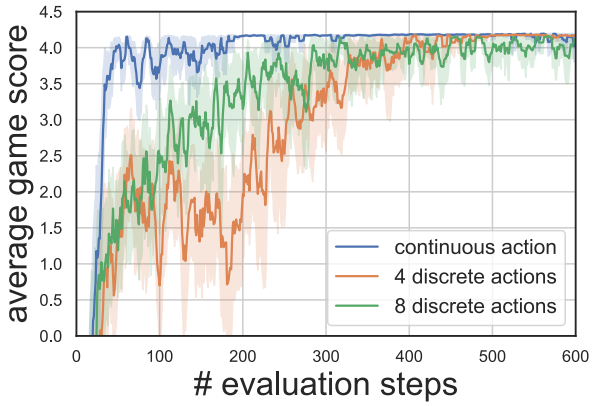


Figure 8.7: Ablation study: Grid world exit.

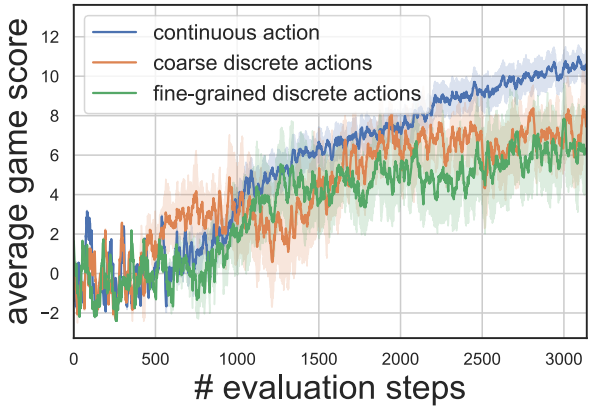


Figure 8.8: Ablation study: Berzerk.

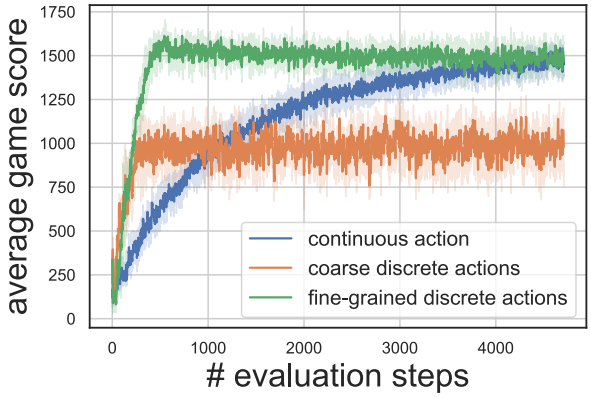


Figure 8.9: Ablation study: Slot machine.

chine scenario, sasRL trained on data from continuous action space converges slower than those trained on data from discrete action spaces. Intuitively, this is caused by the transition model, which is a lot easier to train when all actions are quantized as in the case of discrete action spaces.

8.7 Revisit SDN Application with sasRL

In this thesis, our experiences in employing RL techniques for solving control and management problems in SDN lead to the proposal of sasRL, which is a generic RL paradigm suitable for a wide rage of RL problems. In this section, we apply sasRL to a control application in SDN, which is similar to the service request scheduling problem discussed in Chapter 7. To this end, we first describe the application scenario in Section 8.7.1. Then, the optimisation problem is formally formulated in Section 8.7.2. Finally, Section 8.7.3 and Section 8.7.4 discusses evaluation settings and results, respectively.

8.7.1 The SDN Application Scenario

To evaluate sasRL's performance on SDN control tasks, we consider a simplified service request scheduling task, which is similar to the application scenario in Chapter 6. Some simplifications are made for removing unnecessary assumptions and constraints, as the focus here is on evaluating sasRL's performance.

Specifically, for our application scenario, the networking elements of interest are specialised servers that are placed at various locations throughout the distributed SDN domains which host network services. Multiple types of services are supported, and each server can support and process service requests for one type of service. Each server has a buffer to temporarily store requests waiting for processing, and servers process requests on a first-in-first-out (FIFO) basis. Users residing in SDN domains submit request for service to the corresponding domain controller. Service request received in one domain can be routed and processed by a server in other domains. Specifically, each domain controller, based on its knowledge of the infrastructure status, decides the forwarding rule for received requests. The purpose of such server selection is to minimise the delay before the request is processed. For

our experiment in this section, we only consider requests originated from one SDN domain.

Similar to previous chapters, the system is described by a discrete time-slotted model where service requests are submitted immediately before a time slot begins. At the beginning of each time slot, the controller determines forwarding rules for routing requests to the selected server. It is assumed that servers hosting the same service can have different processing power, which is reflected by the amount of time slots needed for process the request.

The status information of a server is defined as the amount of unfinished work on all servers immediately after the current time slot begins. The amount of unfinished work on a given server is equal to the total number of time slots required to complete all requests pending in the server's buffer. We assume that a service request forwarded from one domain to a server in the other domain does not incur additional delay or processing overhead.

For our experiment, we focus on one domain controller who handles request for service from users residing in its domain. The controller is aware of the status information of all servers in the distributed SDN network. Such information is obtained by the controller from local servers (i.e., servers that are within the same domain as the controller) through SDN's north-south interface, and remote servers (i.e., servers located in other domains) through synchronisation with other domain controllers.

The processing power of servers, i.e., the number of time slots required for processing a unit request, is influenced by a number of environment factors. As such, it is assumed that at any time slot, the controllers is not aware of the server processing power of any servers.

We use the example in Fig. 8.10 to demonstrate the application scenario. In this example, there are 7 servers offering 3 services across 3 domains in the distributed SDN network. We focus on Domain 1 where there are 4 users. At each time slot,

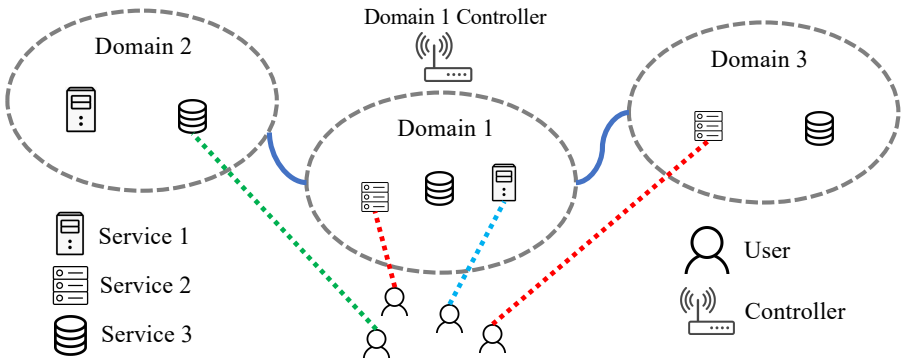


Figure 8.10: The SDN Application Scenario.

the controller of domain 1 knows the latest workload on all servers. However, due to environment uncertainty, the controller does not know the server processing capacity of any servers. Therefore, the goal of domain 1 controller is to develop request forwarding policy to minimise the processing delay of requests over a period of time.

8.7.2 Problem Statement

In this section, we formally define the the request scheduling problem and state the control objective of the controller.

Formally, let $\mathbf{n}_t$ and T denote the request set at time t and the time horizon of the problem. Then, the goal of request scheduling is formulated as an optimisation problem stated as follows

$$\begin{aligned} \min_{d_t} \quad & \frac{1}{T}\mathbb{E}\Big[\sum_{t=1}^{T}\sum_{i\in\mathbf{n}_t} l_t(i)|d_t(i)\Big] \\ \text{s.t.} \quad & |d_t(i)| = 1 \quad \text{for} \quad t = 1, 2, \cdots, T \end{aligned} \tag{8.10}$$

where $l_t(i)$ is the delay for request i submitted at time t and $d_t(i)$ is the forwarding decision for request i. In particular, a forwarding decision consists of percentages of the request being sent to all servers offering the requested service in the network.

Therefore, the constraint $|d_t(i)| = 1$ is in place to ensure that the request is fully served.

Objective: The control objective of the controller is to decide request forwarding rules in order to minimise the average request latency, which is defined as the number of time slots elapsed between the submission and the completion of processing a submitted request.

To solve the optimisation problem in (8.10), we employ RL-based techniques similar to Chapter 6 and Chapter 7, and use sasRL as the algorithm for solving the formulated RL problem. In particular, we define the state space as the concatenation of status information of all servers and requests for service. For example, at any time slot, the state vector contains the amount of unfinished work at all servers (in unit of the number of time slots), and requests for services. The action space corresponds to the forwarding rules for the requests received. Specifically, the action vector at a time slot consists of the percentages of total received requests being forwarded to each servers. Finally, the reward is defined as accumulated tanh-squashed reciprocal of delays of all requests forwarded. Therefore, the lower the delay, the higher the reward.

8.7.3 Experiment Settings

For the experiment, we simulate 3 services, each offered by 2 servers. Since it is assumed that each server only offers 1 service, there are 6 servers in the distributed SDN. The request arrival for 3 services are modelled by Bernoulli process, whereas the server processing power at different time slots are randomly and uniformly distributed. See Section B.5 in the Appendix for detailed simulation settings.

Based on the problem formulation in Section 8.7.2 , we compare the performance of sasRL with DDPG using the same settings detailed in Section B.2 and [117]. For DDPG, in addition to the regular training losses specified by the algorithm, we add

a new regularisation loss to the actor network to regularise its output. For example, the regularisation loss will be incurred if the action vector produced by DDPG's actor network does not assign all requests received at a time slot. As for sasRL, we employ the embodiment implementation described in Section 8.5 in our experiment. Therefore, the regularisation loss is also applied during training, as discussed in Section 8.3.5, which penalises unrealistic next state predictions. For example, according to the time-slotted system described in Section 8.7.1, the waiting time on any server decreases at most by 1 when moving from current time slot to the next time slot. Therefore, the prediction of next state vectors where values decrease for more than 1 incurs regularisation losses. In addition, we proportionally rescale the action prediction that does not forward all requests (percentage sum is less than 1) and/or forwards more requests than received (percentages sum is more than 1).

As for the transition model of sasRL, the input are the current state vector, next state vector and their time slot indices. The output is the predicted action which causes the state transition. Here, the time slot index is included for training the transition model, in the hope that it would assist the transition model in learning temporal dynamics of the environment, e.g., the time-varying server processing power.

8.7.4 Evaluation Results and Analysis

The trainings for sasRL and DDPG take place in an off-line manner, where $10,000$ pre-recorded (s, s', a, r) samples generated by random actions are used. Both algorithms train on data samples for 150 epochs, and the policy is evaluated for 10 runs after each training epoch. The evaluation results are shown in Fig. 8.11.

From Fig. 8.11, we can see that the policy learned by sasRL begins to generate useful forwarding rules after 80 epochs and the training stabilises after about 100 epochs. In comparison, the policy produced by DDPG does not show sign of improvement over the 150 training episode. For this experiment, sasRL's superior

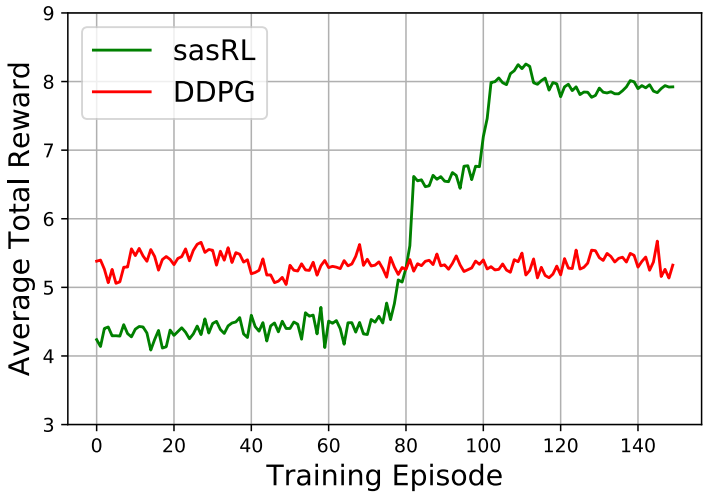


Figure 8.11: Evaluation Results: DDPG Vs. sasRL.

performance is expected for the following reason. First, the reward of a state transition is determined by changes in state vectors. Therefore, excluding the action space and employ STV function reduce the complexity of learning without losing any important information. Second, as the training of the transition model takes into account the time steps of state transitions, the transition model can potentially capture the impacts of temporal dynamics of actions on state transitions, i.e., it implicitly attempts to learns the temporal pattern of server's processing power, which is supposed to improve the transition model's accuracy.

CHAPTER 9

Conclusions and Future Work

9.1 Conclusions

The synchronisation of controllers plays a critical role in supporting superior network performance in distributed SDN. Although there have been numerous design proposals for controller synchronisation policy designs from both industrial and academic research communities, two research issues are largely overlooked. First, there is generally a lack of fundamental understanding on how various factors, including the level of controller synchronisation, impact the performance of distributed SDN. Second, most proposals on controller synchronisation policy design focus on ensuring that the synchronisation process is anomaly-free. However, synchronisation policy design with the focus on improving fine-grained network performance is not sufficiently studied.

To address these open issues, this thesis is dedicated to filling these gaps identified. First, in order to obtain fundamental understandings on the role controller synchronisation plays in enhancing the performance of distributed SDN, we have developed two analytical approaches for modelling and quantifying the relationship between SDN's network performance and various factors such as the network structural properties and the levels of controller synchronisation. The two analytical approaches are developed using different methods with separate sets of assumptions, and the analytical results obtained accordingly are complementary to each other, as they reveal insights from different angles with varying degrees of granularities. In

addition, they arrive at the same conclusion for several findings, thus confirm the validity of these analytical results. To the best of knowledge, our work in this thesis is the first to study distributed SDN from the graph-theoretical perspective. The analytical results shed light on the relationships among network performance and controller synchronisation levels and SDN domain's topological properties, which lay the foundation for synchronisation protocol designs and optimisations.

Second, we have targeted at designing control and management policies for distributed SDN by employing a combination of RL and DL techniques. We start our endeavours in this direction by investigating a well-known service placement problem in the context of SDN. In contrast to traditional optimisation techniques, we define the service placement problem as a serial decision-making problem, which is solved by RL-based techniques via iterative updates. Motivated by the initial success in applying RL-based methods for network management in SDN, we move on to investigate the controller synchronisation policy design via RL. In this regard, we have first tackled the problem of RL-based designs of SDN controller synchronisation policy for improving inter-domain routing performance. Although experiment results are positive, we recognise that this study is limited to the routing application and it comes with several simplifying assumptions. Therefore, in the follow-up works, to enable finer-grained controls and to overcome the constraints initially imposed, we developed and employed more sophisticated DRL approaches. While improving the capabilities of our DRL-based controller synchronisation control and management algorithms, we also consider the scalability and robustness of the policies developed. As such, instead of only developing centralised policies, we have investigated the possibility of having a distributed controller synchronisation policy. Specifically, we identify conditions upon which a set of distributed policies can achieve comparable performance of a centralised policy. Furthermore, we have employed state-of-the-art DL techniques to enhance the temporal data, which our RL algorithms depend on to

learn the policy. This significantly improves the robustness of our DRL-based policy generation algorithms.

Finally, from our experiences with existing RL algorithms, we realise that the learning processes in existing RL algorithms are very inefficiency-prone. Based on our observations and investigations, we have proposed a novel RL paradigm, named state action separable RL (sasRL), which aims at improving RL's training efficiency. As the name suggests, sasRL achieves higher learning efficiency by separating the action space from the value function learning process, which effectively breaks a complex model-free RL problem into a simpler model-free RL problem and a supervised learning problem. It should be noted that sasRL is a generic RL paradigm, which is not limited to networking applications. Therefore, sasRL constitutes another major contribution to generic RL techniques.

9.2 Future Work: Decentralised Policy Learning in SDN

In this thesis, one assumption we have made while investigating RL-based controller synchronisation policy design is that the policy operates in a centralised manner. That is to say, one central controller is responsible for collecting data required from all domains and learning the policy from the data collected. At run time, the central controller coordinates the behaviours of all other controllers. The centralised learning paradigm has the obvious advantage of the ability to make globally optimal decisions. However, such a centralised approach does not scale well and it also creates a single node of failure. Therefore, further research is needed to explore decentralised or distributed approaches for designing the RL-based controller synchronisation policy design. Therefore, in this section, we briefly discuss about some ideas which can enable decentralised or even distributed RL-based policies.

9.2.1 Meta-RL-based Approach

We plan to investigate the potential distributed RL approach in distributed SDN by meta-reinforcement learning (meta-RL) [119]. Meta-RL recently emerges as a promising research field, which has been viewed as a potential gateway to artificial general intelligence (AGI). Broadly speaking, meta-RL aims at designing RL algorithms that can adapt an RL model, which has been trained by data from previous tasks, to accommodate new tasks in a rapid manner with little new data provided. The intuition behind this is to exploit some common structures and dynamics among individual tasks in order to give the RL model some positive inductive bias to help it quickly adapt to particular goals of new tasks. This is appropriate for developing distributed control and management policies in distributed SDN networks where policy developments in domains can be seen as different but related sub-tasks.

In particular, let each domain controller be viewed as an RL agent with its designated network control task. Initially, all agents share a common RL model with initial parameters, which represents the control policy. Agents begin learning individually (individual phase) before they merge information (merging phase) learned by each other. For each setting of domain controller parameters, the underneath distributed optimization process is still used to obtain the corresponding optimal control variables. During the individual phase, each agent learns from its local domain controller parameters. During the merging phase, agents exchange some training data (per design of the meta-RL algorithm) and update the RL-model parameters shared by all agents. The goal of the merging phase is to ensure that information about the optimal control policy learned by individual agents (domain controllers) will be combined and captured in the final RL-model before the next individual phase and continued inference (control) start. The process of updating the final-RL model during the merging phase shall also ensure the fast adaptation of the shared RL model by individual agents for the next individual phase.

Therefore, the meta-RL based approach offers a way to meta-learn across domains, it also enables distributed decision-making within individual domains.

9.2.2 RL-Attention Based Approach

The concept of attention in learning algorithms stems from the simple observation on the way humans perceive information. In particular, when processing visual information, humans pay more attentions to certain distinctive parts of the visual presentation, which play key roles in identifying the meaning of the presentation. The same phenomenon exists in the interpretation of the meanings in sentences, where certain key words and the co-location of some words in a sentence can determine the message the sentence is conveying.

W.r.t. our goal of developing distributed RL-based control and management policy for distributed SDN, the idea of attention can be applied to treat multiple network views generated by various domains. Specifically, recall that due to the distributed nature and the limited synchronisation among domain controllers, the view of the SDN network can be different from domain to domain. We assume that in order to develop distributed control and management policies, each domain develops its own policy based on the domain controller's network view. Ideally, the optimal global policy is developed by the single up-to-date global network view. In reality, the various network views of domains and the subsequent local policies contribute to the global policy to various degrees. Therefore, we can introduce an attention module in the network, whose role is to attend to various network views with different amount of attentions, based on the value the corresponding network view contributes to the global objective. Then, each domain's local policy is adjusted according to the feedback from the attention module. In the meantime, the attention module makes use of the attention weights obtained to develop global control and management policies for the whole network, which are updated once in a while based on new inputs from

all domains.

9.2.3 Model-based Transfer Learning Approach

In this section, we look at the possibility of a model-based transfer learning approach for improving the performance of the collection of local policies by transferring models of local dynamicities for designing global policies.

Here, we assume that global and local network control and management policies share the common objective, i.e., to improve the overall performance of the distributed SDN, but the dynamicities of different domains may vary. All domains can build models of the dynamicities within their own domains based on local data and observations. As local controllers can freely access all local data, it is assumed that there are sufficient data to enable the modelling of local dynamicity. The core idea here is to transfer the knowledge in the local dynamicity models (local model for short) to aid the global policy development. In particular, the central goal of knowledge transfer is to identify how to best combine sub-regions of different local models for building an implicit global dynamicity model (global model for short). This goal may be achieved by employing a Bayesian method which first projects the global model as a mixture of various local models. This is done by expressing the global model as the weighted sum of local models. The weighting vector can be initialised by certain distributions, which are viewed as priors. Then, posterior updates on the priors are conducted using global data and observations collected. The goal of such updates is to reason the similarities between various areas of the global model and their counterparts in local models. During the process, the weighting vector is updated to faithfully reflect the relationships between the global model and various local models w.r.t. to the policy development task considered. Finally, the targeted transfer of knowledge from local models to the global model takes place according to the updated weighting vector. Note that the learned global model can be implicit,

as we expect DNN-based function approximations maybe used, which renders the opaque representations of models.

Bibliography

[1] N. Feamster, J. Rexford, and E. Zegura, "The road to SDN: An intellectual history of programmable networks," *SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 87–98, 2014. 1, 16

[2] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu *et al.*, "B4: Experience with a globally-deployed software defined WAN," *SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 3–14, 2013. 1, 23

[3] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking control of the enterprise," in *SIGCOMM Computer Communication Review*, vol. 37, no. 4, 2007, pp. 1–12. 1

[4] S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali, "On scalability of software-defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 136–141, 2013. 2

[5] V. Kotronis, X. Dimitropoulos, and B. Ager, "Outsourcing the routing control logic: Better Internet routing based on SDN principles," in *ACM HotNets-XI*, 2012. 2, 23

[6] V. Kotronis, R. Klöti, M. Rost, P. Georgopoulos, B. Ager, S. Schmid, and X. Dimitropoulos, "Stitching inter-domain paths over IXPs," in *ACM SOSR*, 2016. 2, 3

[7] G. Petropoulos, F. Sardis, S. Spirou, and T. Mahmoodi, "Software-defined inter-networking: Enabling coordinated QoS control across the Internet," in *IEEE ICT*, 2016. 2, 3

[8] Z. Chen, J. Bi, Y. Fu, Y. Wang, and A. Xu, "MLV: A multi-dimension routing information exchange mechanism for inter-domain SDN," in *IEEE ICNP*, 2015. 2, 3

[9] P. Thai and J. C. de Oliveira, "Decoupling policy from routing with software defined interdomain management: Interdomain routing for SDN-based networks," in *IEEE ICCCN*, 2013. 2, 3

[10] A. Panda, C. Scott, A. Ghodsi, T. Koponen, and S. Shenker, "CAP for networks," in *ACM HotSDN*, 2013. 2, 13, 81, 82

[11] R. Mahajan and R. Wattenhofer, "On consistent updates in software defined networks," in *ACM HotSDN*, 2013. 3

[12] K.-T. Förster, R. Mahajan, and R. Wattenhofer, "Consistent updates in software defined networks: On dependencies, loop freedom, and blackholes," in *IFIP Networking Conference (IFIP Networking) and Workshops*, 2016. 3

[13] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for network update," *ACM SIGCOMM Computer Communication Review*, vol. 42, pp. 323–334, 2012. 3

[14] K.-T. Foerster, S. Schmid, and S. Vissicchio, "Survey of consistent software-defined network updates," *IEEE Communications Surveys & Tutorials*, 2018. 3

[15] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998. 4, 161, 162, 164, 170, 175, 177

[16] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015. 4

[17] P. Amaral, J. Dinis, P. Pinto, L. Bernardo, J. Tavares, and H. S. Mamede, "Machine learning in software defined networks: Data collection and traffic classification," in *IEEE 24th International Conference on Network Protocols (ICNP)*, 2016. 4

[18] A. S. Muqaddas, P. Giaccone, A. Bianco, and G. Maier, "Inter-controller traffic to support consistency in ONOS clusters," *IEEE Transactions on Network and Service Management*, vol. 14, no. 4, pp. 1018–1031, 2017. 13, 81, 107

[19] Q. Qin, K. Poularakis, G. Iosifidis, and L. Tassiulas, "Sdn controller placement at the edge: Optimizing delay and overheads," in *IEEE INFOCOM 2018*. 13, 81, 107

[20] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008. 16

[21] L. Schiff, S. Schmid, and P. Kuznetsov, "In-band synchronization for distributed SDN control planes," *ACM SIGCOMM Computer Communication Review*, vol. 46, no. 1, pp. 37–43, Jan. 2016. 16

[22] F. Benamrane, R. Benaini *et al.*, "An east-west interface for distributed sdn control plane: Implementation and evaluation," *Computers & Electrical Engineering*, vol. 57, pp. 162–175, 2017. 19

[23] Z. Latif, K. Sharif, F. Li, M. M. Karim, S. Biswas, and Y. Wang, "A comprehensive survey of interface protocols for software defined networks," *Journal of Network and Computer Applications*, vol. 156, p. 102563, 2020. 20

[24] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow *et al.*, "Onos: Towards an open, distributed SDN OS," in *ACM HotSDN*, 2014. 20, 23, 65, 82, 121, 155

[25] Y. Lai and W. S. Lai, "A graph-theoretic model of routing hierarchies," in *IEEE WAINA*, 2009. 22

[26] B. Awerbuch and Y. Shavitt, "Topology aggregation for directed graphs," *IEEE/ACM Transactions on Networking*, vol. 9, no. 1, pp. 82–90, 2001. 22

[27] B. Awerbuch, Y. Du, and Y. Shavitt, "The effect of network hierarchy structure on performance of atm pnni hierarchical routing," *Computer Communications*, vol. 23, no. 10, pp. 980–986, 2000. 22

[28] R. Cherukuri, D. Dykeman, and M. Goguen, "PNNI draft specification," in *ATM Forum*, 1995, pp. 94–0471. 22

[29] I. Castineyra, N. Chiappa, and M. Steenstrup, "The Nimrod routing architecture," Tech. Rep., 1996. 22

[30] K. Poularakis, G. Iosifidis, G. Smaragdakis, and L. Tassiulas, "One step at a time: Optimizing SDN upgrades in ISP networks," in *IEEE INFOCOM*, 2017. 23

[31] A. Tootoonchian and Y. Ganjali, "Hyperflow: A distributed control plane for openflow," in *ACM INM/WREN*, 2010. 23

[32] K. Phemius, M. Bouet, and J. Leguay, "Disco: Distributed multi-domain SDN controllers," in *IEEE NOMS*, 2014. 23

[33] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: Scaling flow management for high-performance net-

works," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 254–265, 2011. 23, 82

[34] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: A framework for efficient and scalable offloading of control applications," in *ACM HotSDN*, 2012. 23, 82

[35] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with DIFANE," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 4, pp. 351–362, 2010. 23

[36] S. Vissicchio, O. Tilmans, L. Vanbever, and J. Rexford, "Central control over distributed routing," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 43–56, 2015. 23

[37] K.-K. Yap, M. Motiwala, J. Rahe, S. Padgett, M. Holliman, G. Baldus, M. Hines, T. Kim, A. Narayanan, A. Jain *et al.*, "Taking the edge off with espresso: Scale, reliability and programmability for global internet peering," in *ACM SIGCOMM*, 2017. 23

[38] B. Schlinker, H. Kim, T. Cui, E. Katz-Bassett, H. V. Madhyastha, I. Cunha, J. Quinn, S. Hasan, P. Lapukhov, and H. Zeng, "Engineering egress with edge fabric: Steering oceans of content to the world," in *ACM SIGCOMM*, 2017. 23

[39] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, no. 6684, pp. 440–442, 1998. 23, 24

[40] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon, "Network motifs: Simple building blocks of complex networks," *Science*, vol. 298, no. 5594, pp. 824–827, 2002. 23

[41] A. L. Barabási, "Scale-free networks: A decade and beyond," *Science*, vol. 325, no. 5939, pp. 412–413, 2009. 23

[42] P. Erdös and A. Rényi, "On the evolution of random graphs," *Publ. Math. Inst. Hung. Acad. Sci*, vol. 5, no. 1, pp. 17–60, 1960. 24, 52, 76

[43] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999. 24, 52

[44] M. E. Newman, S. H. Strogatz, and D. J. Watts, "Random graphs with arbitrary degree distributions and their applications," *Physical review E*, vol. 64, no. 2, p. 026118, 2001. 24, 33, 34

[45] B. Jiang, P. Nain, D. Towsley, and S. Guha, "On a class of stochastic multilayer networks," in *ACM SIGMETRICS*, 2018. 24

[46] B. S. Everitt, *Mixture Distributions*. Wiley Online Library, 1985. 46

[47] "University of Oregon route views project," University of Oregon, 2005. [Online]. Available: http://www.routeviews.org/ 52

[48] "Rocketfuel: An ISP topology mapping engine," University of Washington, 2002. [Online]. Available: http://www.cs.washington.edu/research/networking/rocketfuel/interactive/ 52, 74, 96

[49] "Center for applied Internet data analysis (CAIDA)," CAIDA, 2017. [Online]. Available: http://www.caida.org/home/ 52

[50] "Stanford network analysis platform (SNAP)," Standford University, 2017. [Online]. Available: https://snap.stanford.edu/snap/ 53, 77, 121

[51] The Opendaylight Controller. [Online]. Available: https://www.opendaylight.org 65, 82

[52] M. E. Newman, "Clustering and preferential attachment in growing networks," *Physical review E*, vol. 64, no. 2, p. 025102, 2001. 80

[53] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, "Logically centralized?: state distribution trade-offs in software defined networks," in *ACM HotSDN*, 2012. 82, 83

[54] A. S. Muqaddas, A. Bianco, P. Giaccone, and G. Maier, "Inter-controller traffic in ONOS clusters for SDN networks," in *IEEE ICC*, 2016. 82, 97, 121, 155

[55] E. Sakic and W. Kellerer, "Impact of adaptive consistency on distributed sdn applications: An empirical study," *IEEE Journal on Selected Areas in Communications*, 2018. 83

[56] Z. Guo, M. Su, Y. Xu, Z. Duan, L. Wang, S. Hui, and H. J. Chao, "Improving the performance of load balancing in software-defined networks through load variance-based synchronization," *Computer Networks*, vol. 68, pp. 95–109, 2014. 83

[57] M. Aslan and A. Matrawy, "Adaptive consistency for distributed sdn controllers," in *2016 17th IEEE International Telecommunications Network Strategy and Planning Symposium (Networks)*. 83, 97

[58] E. Sakic, F. Sardis, J. W. Guck, and W. Kellerer, "Towards adaptive state consistency in distributed sdn control plane," in *IEEE ICC*, 2017. 83

[59] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015. 83, 90, 92, 99, 118, 162, 178

[60] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, 2016. 83, 90

[61] Z. Zhang, L. Ma, K. K. Leung, L. Tassiulas, and J. Tucker, "Q-placement: Reinforcement-learning-based service placement in software-defined networks," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. 83

[62] S.-C. Lin, I. F. Akyildiz, P. Wang, and M. Luo, "Qos-aware adaptive routing in multi-layer hierarchical software defined networks: a reinforcement learning approach," in *2016 IEEE International Conference on Services Computing (SCC)*. 83

[63] V. Kotronis, A. Gämperli, and X. Dimitropoulos, "Routing centralization across domains via sdn: A model and emulation framework for bgp evolution," *Computer Networks*, vol. 92, pp. 227–239, 2015. 84

[64] C. White, *Markov decision processes*. Springer, 2001. 88

[65] D. P. Bertsekas and J. N. Tsitsiklis, "Neuro-dynamic programming: an overview," in *Proceedings of the 34th IEEE Conference on Decision and Control*, vol. 1. IEEE Publ. Piscataway, NJ, 1995, pp. 560–564. 90, 112

[66] M. T. Hagan, H. B. Demuth, M. H. Beale, and O. De Jesús, *Neural network design*. Pws Pub. Boston, 1996, vol. 20. 90, 112, 147

[67] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992. 91, 113, 115, 116, 140, 141, 150, 161, 162, 173

[68] M. J. Kearns and S. P. Singh, "Finite-sample convergence rates for Q-learning and indirect algorithms," in *Advances in Neural Information Processing Systems*, 1999, pp. 996–1002. 91

[69] H. V. Hasselt, "Double Q-learning," in *NIPS*, 2010. 92, 118

[70] L.-J. Lin, "Reinforcement learning for robots using neural networks," Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, Tech. Rep., 1993. 93, 118

[71] M. Kumar and N. Yadav, "Multilayer perceptrons and radial basis function neural network methods for the solution of differential equations: a survey," *Computers & Mathematics with Applications*, vol. 62, no. 10, pp. 3796–3811, 2011. 93, 118, 157

[72] F. Chollet *et al.*, "Keras: The python deep learning library," *Astrophysics Source Code Library*, 2018. 93, 119, 157

[73] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: a system for large-scale machine learning." in *OSDI*, vol. 16, 2016, pp. 265–283. 93, 119, 157

[74] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *2010 The 27th ICML*. 93, 119, 157

[75] S. B. Thrun, "Efficient exploration in reinforcement learning," Tech. Rep., 1992. 94

[76] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy, "Deep exploration via bootstrapped dqn," in *Advances in neural information processing systems*, 2016, pp. 4026–4034. 94

[77] R. Y. Chen, S. Sidor, P. Abbeel, and J. Schulman, "Ucb exploration via q-ensembles," *arXiv preprint arXiv:1706.01502*, 2017. 94

[78] M. G. Azar, R. Munos, and B. Kappen, "On the sample complexity of reinforcement learning with a generative model," *arXiv preprint arXiv:1206.6461*, 2012. 94

[79] Q. Duan, Y. Yan, and A. V. Vasilakos, "A survey on service-oriented network virtualization toward convergence of networking and cloud computing," *IEEE Transactions on Network and Service Management*, vol. 9, no. 4, pp. 373–392, 2012. 102

[80] J. Abley and K. Lindqvist, "Operation of anycast services," Tech. Rep., 2006. 102, 131

[81] S. M. Das, H. Pucha, and Y. C. Hu, "Mitigating the gateway bottleneck via transparent cooperative caching in wireless mesh networks," *Ad Hoc Networks*, vol. 5, no. 6, pp. 680–703, 2007. 104

[82] C. Labovitz, G. R. Malan, and F. Jahanian, "Internet routing instability," *IEEE/ACM transactions on Networking*, no. 5, pp. 515–528, 1998. 107

[83] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015. 109

[84] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *NIPS 2000*. 113, 162, 176

[85] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *NIPS 2000*. 113

[86] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015. 113, 179

[87] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," in *AAAI 2018*. 113

[88] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," *arXiv preprint arXiv:1511.06581*, 2015. 113

[89] A. Tavakoli, F. Pardo, and P. Kormushev, "Action branching architectures for deep reinforcement learning," in *AAAI 2018*. 113, 114, 116, 149

[90] S. Borst, V. Gupta, and A. Walid, "Distributed caching algorithms for content distribution networks," in *IEEE INFOCOM*, 2010. 122

[91] H. El-Rewini, T. G. Lewis, and H. H. Ali, *Task scheduling in parallel and distributed systems*. Prentice-Hall, Inc., 1994. 135

[92] K. P. Murphy, "A survey of POMDP solution techniques," 2000. 136

[93] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997. 147, 148

[94] K. Poularakis, Q. Qin, L. Ma, S. Kompella, K. K. Leung, and L. Tassiulas, "Learning the optimal synchronization rates in distributed SDN control architectures," in *IEEE INFOCOM*, 2019. 155

[95] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014. 157

[96] Y. Bengio *et al.*, "Learning deep architectures for AI," *Foundations and trends in machine learning*, vol. 2, no. 1, pp. 1–127, 2009. 162

[97] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013. 162

[98] M. T. Hagan, H. B. Demuth, and M. Beale, "Neural network design," *Co., Boston, Mass., USA*, 1995. 162

[99] A. D. Edwards, H. Sahni, R. Liu, J. Hung, A. Jain, R. Wang, A. Ecoffet, T. Miconi, C. Isbell, and J. Yosinski, "Estimating q (s, s') with deep deterministic dynamics gradients," *arXiv preprint arXiv:2002.09505*, 2020. 164

[100] V. François-Lavet, Y. Bengio, D. Precup, and J. Pineau, "Combined reinforcement learning via abstract representations," in *AAAI Conference on Artificial Intelligence*, 2019, pp. 3582–3589. 165

[101] R. Sutton, "The bitter lesson," *Incomplete Ideas (blog)*, vol. 13, 2019. 165

[102] A. Zhang, H. Satija, and J. Pineau, "Decoupling dynamics and reward for transfer learning," *arXiv preprint arXiv:1804.10689*, 2018. 165

[103] V. Liu, R. Kumaraswamy, L. Le, and M. White, "The utility of sparse representations for control in reinforcement learning," in *AAAI Conference on Artificial Intelligence*, 2019, pp. 4384–4391. 165

[104] Y. Chandak, G. Theocharous, J. Kostas, S. Jordan, and P. S. Thomas, "Learning action representations for reinforcement learning," *arXiv preprint arXiv:1902.00183*, 2019. 165

[105] Y. Chen, Y. Chen, Y. Yang, Y. Li, J. Yin, and C. Fan, "Learning action-transferable policy with action embedding," *arXiv preprint arXiv:1909.02291*, 2019. 165

[106] J. Oh, S. Singh, and H. Lee, "Value prediction network," in *Advances in Neural Information Processing Systems*, 2017, pp. 6118–6128. 165

[107] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu, "Reinforcement learning with unsupervised auxiliary tasks," *arXiv preprint arXiv:1611.05397*, 2016. 165

[108] T. Degris, M. White, and R. S. Sutton, "Off-policy actor-critic," *arXiv preprint arXiv:1205.4839*, 2012. 168, 170, 176, 230

[109] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *International Conference on Machine Learning*, 2014, pp. 1387–1395. 169, 179

[110] C. Nota and P. S. Thomas, "Is the policy gradient a gradient?" *arXiv preprint arXiv:1906.07073*, 2020. 169

[111] R. S. Sutton, S. Singh, and D. McAllester, "Comparing policy-gradient algorithms," *IEEE Transactions on Systems, Man, and Cybernetics*, 2000. 169

[112] C. Szepesvári, "The asymptotic convergence-rate of Q-learning," in *Advances in Neural Information Processing Systems*, 1998, pp. 1064–1070. 173

[113] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992. 177

[114] M. W. Gardner and S. Dorling, "Artificial neural networks (the multilayer perceptron) â A review of applications in the atmospheric sciences," *Atmospheric environment*, vol. 32, no. 14-15, pp. 2627–2636, 1998. 178

[115] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *arXiv preprint arXiv:1801.01290*, 2018. 179, 183

[116] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017. 180

[117] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable baselines," https://github.com/hill-a/stable-baselines, 2018. 180, 189

[118] J. Tyo and Z. Lipton, "How transferable are the representations learned by deep Q agents?" *arXiv preprint arXiv:2002.10021*, 2020. 180

[119] J. X. Wang, Z. Kurth-Nelson, D. Kumaran, D. Tirumala, H. Soyer, J. Z. Leibo, D. Hassabis, and M. Botvinick, "Prefrontal cortex as a meta-reinforcement learning system," *Nature neuroscience*, vol. 21, no. 6, pp. 860–868, 2018. 195

APPENDIX A

Proofs

A.1 Proofs for Lemmas, Theorems, and Corollaries in Chapter 3

A.1.1 Proof for Theorem 3.1

Theorem 3.1 is derived by further analysis of the APC expressions obtained in Type-1 Network, which is summarised in Theorem 3.9. Specifically, in (3.19),

$$\log(\frac{n\tau' + 1 - \gamma}{\zeta_1 \gamma}) = \log(\frac{n\tau'}{\zeta_1 \gamma} - \frac{\gamma - 1}{\zeta_1 \gamma}) \leq \log(\frac{n\tau'}{\zeta_1 \gamma}), \tag{A.1}$$

as $\gamma \geq 1$. Next, for η_1 in (3.19), we know

$$\eta_1 = \lfloor \frac{\log(m/z_1')}{\tau' \log(z_2'/z_1')} + \frac{2}{\tau'} \rfloor - 1 \leq \frac{\log(m/z_1')}{\tau' \log(z_2'/z_1')} + \frac{2}{\tau'} - 1. \tag{A.2}$$

There are four cases for (A.2). First, when $\tau = 1$, then $\eta_1 \leq \frac{\log(m/z_1')}{\log(z_2'/z_1')} + 1$. Second, when $\tau = 2$, then $\eta_1 \leq \frac{\log(m/z_1')}{2\log(z_2'/z_1')}$. Third, when $2 < \tau \leq \frac{\log(m/z_1')}{\log(z_2'/z_1')} + 2$, we have $\tau = \tau'$; therefore, $\eta_1 \leq \frac{\log(m/z_1')}{\tau \log(z_2'/z_1')} + \frac{2}{\tau} - 1 \approx \frac{\log(m/z_1')}{\tau \log(z_2'/z_1')}$. Fourth, when $\tau > \frac{\log(m/z_1')}{\log(z_2'/z_1')} + 2$, then $\eta_1 = 0 < \frac{\log(m/z_1')}{\tau \log(z_2'/z_1')}$. Therefore, $\exists$ constant c_0 such that

$$\eta_1 \leq \frac{c_0 \log(m/z_1')}{\tau \log(z_2'/z_1')}. \tag{A.3}$$

Since $\eta_2 \leq \tau'$, when $\gamma \leq \frac{n\tau'+1}{\zeta_1+1}$, using (A.1) and (A.3), we get

$$\begin{aligned}
&\frac{\eta_1 \xi \log(\frac{n\tau'+1-\gamma}{\zeta_1\gamma})}{\log(\zeta_2/\zeta_1)} + \frac{\log(n\eta_2/\zeta_1)}{\log(\zeta_2/\zeta_1)} + \eta_1(\xi+1) + 1 \\
\leq &\frac{c_0 \xi \log(m/z_1') \log(\frac{n\tau'}{\zeta_1\gamma})}{\tau \log(z_2'/z_1') \log(\zeta_2/\zeta_1)} \\
&+ \frac{\log(n\tau'/\zeta_1)}{\log(\zeta_2/\zeta_1)} + \frac{c_0(\xi+1)\log(m/z_1')}{\tau \log(z_2'/z_1')} + 1 \\
= &O\Big(\frac{\log(m/z_1') \log(\frac{n\tau'}{\zeta_1\gamma})}{\tau \log(z_2'/z_1') \log(\zeta_2/\zeta_1)} + \frac{\log(n\tau'/\zeta_1)}{\log(\zeta_2/\zeta_1)}\Big).
\end{aligned} \tag{A.4}$$

Similarly, when $\gamma > \frac{n\tau'+1}{\zeta_1+1}$, we have

$$\begin{aligned}
&\frac{\log(n\eta_2/\zeta_1)}{\log(\zeta_2/\zeta_1)} + \eta_1(\xi+1) + 1 \\
\leq &\frac{\log(n\tau'/\zeta_1)}{\log(\zeta_2/\zeta_1)} + \frac{c_0(\xi+1)\log(m/z_1')}{\tau \log(z_2'/z_1')} + 1 \\
= &O\Big(\frac{\log(m/z_1')}{\tau \log(z_2'/z_1')} + \frac{\log(n\tau'/\zeta_1)}{\log(\zeta_2/\zeta_1)}\Big),
\end{aligned} \tag{A.5}$$

thus completing the proof. □

A.1.2 Proof for Theorem 3.4

To prove Theorem 3.4, we consider two trees that are constructed w.r.t. a line network of k domains ($\mathcal{F}$) and its RDPN ($\mathcal{F}_R$), where $\mathcal{F}$ and $\mathcal{F}_R$ have the same node set. The constructed trees w.r.t. $\mathcal{F}_R$ and $\mathcal{F}$ are denoted by $\mathcal{T}_1$ and $\mathcal{T}_2$, respectively. We first describe how $\mathcal{T}_1$ and $\mathcal{T}_2$ are constructed and then show how they are related. To construct $\mathcal{T}_1$ w.r.t. $\mathcal{F}_R$, we randomly pick a node r in $\mathcal{F}_R$ as the root. From this randomly chosen root r, tree $\mathcal{T}_1$ is formed in a way that $\mathcal{T}_1$ spans all vertices in $\mathcal{F}_R$ and the number of nodes in level-i (nodes in level-i are $(i-1)$-connections away from r, $i = 1, 2, \ldots$) in $\mathcal{T}_1$ is the same as the *average number of nodes* that are $(i-1)$-hop away from r in $\mathcal{F}_R$. Note that $\mathcal{T}_1$ is constructed based on the statistical properties of $\mathcal{F}_R$ rather than a particular instance of $\mathcal{F}_R$. Then following the same

rule, tree $\mathcal{T}_2$ w.r.t. $\mathcal{F}$ is constructed rooted at r. Let $\varsigma_i^{(1)}$ and $\varsigma_i^{(2)}$ denote the set of nodes that are i-hop away from r in trees $\mathcal{T}_1$ and $\mathcal{T}_2$, respectively. Then the average path cost from a random node to node r in $\mathcal{F}_R$ and $\mathcal{F}$ is computable based on $\mathcal{T}_1$ and $\mathcal{T}_2$ via $\sum_i (i\varsigma_i^{(1)}/|kn|)$ and $\sum_i (i\varsigma_i^{(2)}/|kn|)$ (recall that both $\mathcal{F}_R$ and $\mathcal{F}$ contain kn nodes), respectively.

Although $\mathcal{F}_R$ and $\mathcal{F}$ have the same degree distribution, nodes in $\varsigma_i^{(2)}$ are more likely to have edges connecting to each other than nodes in $\varsigma_i^{(1)}$ do. This is caused by the structural constraints (inter-domain connections) imposed on the line network $\mathcal{F}$, which is not present in its RDPN $\mathcal{F}_R$. Such effect is reflected on the tree constructions of $\mathcal{T}_2$. In particular, starting from $\mathcal{T}_1$, to get $\mathcal{T}_2$, some nodes in $\varsigma_i^{(1)}$ are effectively moved to $\varsigma_j^{(2)}$ with $j > i$. Therefore, $\forall v$ in $\mathcal{T}_1$, in $\mathcal{T}_2$, v either remains unchanged or moves further away from the root. Hence, let ρ' be the expected cost of the shortest path (in terms of hop counts) between any two random nodes in $\mathcal{F}$, we have $\rho_R \leq \rho'$ when $|M| = |M_R| = 1$. On the other hand, when $|M| = |M_R| > 1$, we can view root r above as a set of nodes in $\mathcal{F}_R$ (or $\mathcal{F}$) with the cardinality the same as that of M_R and M (i.e., a root set); then the depth of a node v in the tree represents the shortest path (in terms of hop counts) from v to the closest node in this "root set". In this way, the above argument still applies when $|M| = |M_R| > 1$, and again we get $\rho_R \leq \rho'$.

Furthermore, the above discussion assumes pure random selection of the root set in $\mathcal{T}_2$, which includes situations where some nodes in the root set are in transit-domains (defined in Definition 5), i.e., $\mathcal{A}_2, \ldots, \mathcal{A}_{k-1}$. Let $\mathcal{M}$ denote such root set in $\mathcal{F}$. Then, let ρ'' denote the average distance between node v ($v \in \mathcal{A}_1$) and node set $\mathcal{M}$. However, we are only concerned with the average distance between node v ($v \in \mathcal{A}_1$) and node set M ($M \subseteq \mathcal{A}_k$) in $\mathcal{F}$, which is defined as ρ. Apparently, $\rho > \rho''$ holds according to Theorem 3.6. Since ρ' is the average value over all scenarios, i.e., ρ' is the average of ρ and ρ'', there is $\rho > \rho'$. Thus, we have $\rho_R \leq \rho' \leq \rho$.

A.1.3 Proof of Lemma 3.5

In order to derive the number of 1-hop and 2-hop nodes from a random node in the RDPN of a line network with k domains, the first step is to obtain the degree distribution of the RDPN which incorporates the added degree from the inter-domain connections on top of the intra-domain connections.

Let χ be the random variable of the degree distribution of a node in a domain before inter-domain connections are established (i.e., intra-domain connections). Denote the random variables of the added degrees of nodes in end-domains and transit-domains (defined in Definition 5) in a line network by ψ and Υ, respectively, after the inter-domain connections are established. Then, when $\beta \ll n$, ψ and Υ follow binomial distribution with the following parameters: $\psi \sim \mathrm{B}\left(\beta, \frac{1}{n}\right)$ and $\Upsilon \sim \mathrm{B}\left(2\beta, \frac{1}{n}\right)$. Then the overall degree distributions for a random node in a line network of length k ($k \geq 2$) is represented by $(\chi + \frac{2}{k}\psi + \frac{k-2}{k}\Upsilon)$.

The average number of 1-hop nodes from a random node within a domain, denoted by z_1, is $z_1 = \mathbb{E}[\chi] = \sum_{k=0}^{\infty} k p_k$, where p_k is the percentage of nodes with degree k. For the RDPN, whose degree distribution is captured by random variable $(\chi + \frac{2}{k}\psi + \frac{k-2}{k}\Upsilon)$, we have $\zeta_1 = \mathbb{E}[\chi] + \frac{2}{k}\mathbb{E}[\psi] + \frac{k-2}{k}\mathbb{E}[\psi]$. The mean of χ is z_1. As for ψ and Υ, since they follow binomial distribution, their means are $\frac{\beta}{n}$ and $\frac{2\beta}{n}$, respectively. Thus, we have $\zeta_1 = z_1 + \frac{2\beta(k-1)}{nk}$. For the calculation of the average number of 2-hop nodes from a random nodes, denoted by z_2, [28] gives an expression: $z_2 = \sum_{k=0}^{\infty} k(k-1)p_k$. However, it is difficult to directly apply this expression to derive ζ_2. Instead, we use $z_2 = \sum_{k=0}^{\infty} k(k-1)p_k \approx \sum_{k=0}^{\infty} k^2 p_k$ as an approximation. The calculation now becomes tractable, because for a random variable x, the following result holds: $\mathbb{E}[x^2] = \sigma_x^2 + \mathbb{E}^2[x]$, where σ_x^2 is the variance of x. Therefore, we can calculate ζ_2 in the following way: $\zeta_2 = \mathbb{E}[(\chi + \frac{2}{k}\psi + \frac{k-2}{k}\Upsilon)^2] = \sigma^2_{\chi+\frac{2}{k}\psi+\frac{k-2}{k}\Upsilon} + \zeta_1^2$. Since χ, ψ and Υ are independent random variables, $\sigma^2_{\chi+\frac{2}{k}\psi+\frac{k-2}{k}\Upsilon} = \sigma_\chi^2 + (\frac{2}{k})^2\sigma_\psi^2 + (\frac{k-2}{k})^2\sigma_\Upsilon^2$. Given that the variance of a random vari-

able $x \sim \mathrm{B}(n,p)$ is $np(1-p)$, we calculate σ_χ^2, σ_ψ^2, and σ_Υ^2 accordingly. Since we assume $\beta \ll n$, we have $\beta/n^2 \approx 0$ and $\beta^2/n^2 \approx 0$; therefore, we conclude that $\zeta_2 = z_2 + z_1\frac{4\beta(k-1)}{nk}$. □

A.1.4 Proof of Theorem 3.6

Consider two line networks of length k and $k+1$, whose domains are labelled as $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_k$ and $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_{k+1}$, respectively. Let $V_{\text{in}}(\mathcal{C}_i)$ ($V_{\text{out}}(\mathcal{C}_i)$) be the set of gateways in domain $\mathcal{C}_i$ connecting to domain $\mathcal{C}_{i-1}$ ($\mathcal{C}_{i+1}$). Without loss of generality, we assume that $\mathcal{A}_1 = \mathcal{B}_1$ and $\mathcal{A}_k = \mathcal{B}_{k+1}$. This implies that $V_{\text{out}}(\mathcal{A}_1) = V_{\text{out}}(\mathcal{B}_1)$ and $V_{\text{in}}(\mathcal{A}_k) = V_{\text{in}}(\mathcal{B}_{k+1})$. Therefore, $L_k(\beta)$ and $L_{k+1}(\beta)$ are determined by the pair-wise distance between $V_{\text{out}}(\mathcal{A}_1)$ and $V_{\text{in}}(\mathcal{A}_k)$, and $V_{\text{out}}(\mathcal{B}_1)$ and $V_{\text{in}}(\mathcal{B}_{k+1})$, respectively. When $k \geq 3$, there exist at least one transit domain, apart from the end-domains in a line network (see Definition 5 for the concept of transit/end-domains). Since all transit domains have the same statistical parameters, each transit domain offers the same probability of finding a path with certain path cost in that domain. Furthermore, more transit domains introduce more inter-domain edges. Thus, there is no higher possibility of finding a path with lower path cost due to the presence of more transit domains. Therefore, more transit domains result in higher expectation of pair-wise distance between $V_{\text{out}}(\mathcal{B}_1)$ and $V_{\text{in}}(\mathcal{B}_{k+1})$. Moreover, these arguments apply to both Type-1 and Type-2 networks. This concludes the proof. □

Remark: Note that in Theorem 3.6, there are two uncovered cases. First, $k = 1$. Since we are not interested in determining APC for two random nodes within the same domain, this case does not exist. Second, $k = 2$. From numerical results, we observe that $L_2(\beta)$ may be slightly greater than $L_3(\beta)$ when β satisfies certain conditions. This could be intuitively understood as follows. When $k = 3$, the transit-domain between the source and destination domain acts as a bridge, which could reduce the average distance between the egress gateway sets in the source domain

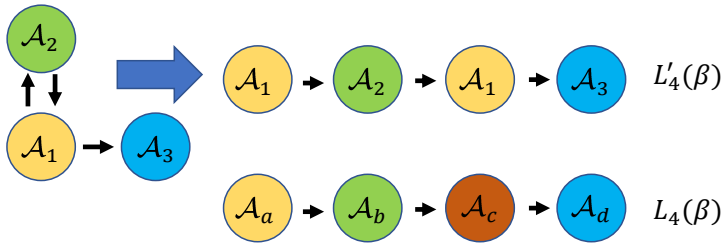


Figure A.1: Non-simple vs. simple domain-wise path for path constructions.

and the ingress gateway sets in the destination domain. In one extreme example, if the transit-domain consists of only one node, then any egress gateway in the source domain can reach any ingress gateway in the destination domain, for which the APC is calculated as $\mathbb{E}[D_2^{(\beta)}]$ (see the definition of $D_k^{(\beta)}$ in Theorem 3.11); In comparison, the APC in the scenario where $k = 2$ is calculated by $\mathbb{E}[M_2^{(\beta)}]$ (see the definition of $M_k^{(\beta)}$ in Theorem 3.11). Apparently, there is: $\mathbb{E}[M_2^{(\beta)}] > \mathbb{E}[D_2^{(\beta)}]$. Nevertheless, the case that two random nodes residing in two neighbouring domains ($k = 2$) only happens with probability z'_1/m, which can be ignored as $z'_2 \gg z'_1$ and m is large.

A.1.5 Proof of Corollary 3.7

We start the proof by comparing $L'_{k'}(\beta)$ and $L_{k'}(\beta)$. We consider the simplest form of domain repetition where only one domain is traversed twice. We use Fig. A.1 to facilitate the proof, where $k' = 4$. Suppose that a random node in domain $\mathcal{A}_1$ needs to communicate with a random node in domain $\mathcal{A}_3$ and the selected domain-wise path is $\mathcal{A}_1 - \mathcal{A}_2 - \mathcal{A}_1 - \mathcal{A}_3$. Apparently, this is not a simple path because domain $\mathcal{A}_1$ is traversed twice; the corresponding APC is denoted by $L'_4(\beta)$. We also consider a similar scenario with a simple domain-wise path of the same path cost, $\mathcal{A}_a - \mathcal{A}_b - \mathcal{A}_c - \mathcal{A}_d$, whose APC is denoted by $L_4(\beta)$. We observe that the computation of $L'_4(\beta)$ is the same as that in $L_4(\beta)$ except that there are effectively less inter-domain path options. Specifically, for the case of non-simple domain-wise paths, when selecting the gateway in $\mathcal{A}_2$ to go back to $\mathcal{A}_1$, there are only $\beta - 1$

potential options, since the ingress and egress gateway sets fully overlap in $\mathcal{A}_2$, and ingress gateway in $\mathcal{A}_2$ is determined already and it cannot be used as the egress gateway; otherwise, entering $\mathcal{A}_2$ would only increase APC compared to entering $\mathcal{A}_3$ directly from $\mathcal{A}_1$. By contrast, the ingress gateway selection in $\mathcal{A}_c$ has β options. Hence, $L'_4(\beta) > L_4(\beta)$. Such analysis remains valid in cases where there are more repeated domains in the domain-wise path. We also know from Theorem 3.6 that $L_{k'}(\beta) > L_{k'-1}(\beta)$. Finally, since $k' \geq k \geq 3$, we have $L_k(\beta) < L'_{k'}(\beta)$, thus completing the proof. □

A.1.6 Proof of Theorem 3.10

Recall that in our two-layer model, gateways are randomly selected. Therefore, let $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_\beta$ be i.i.d. random variables, denoting the minimum path cost between two random nodes in a domain with the same pdf as $\mathcal{D}$ in (3.22). We have

$$M^{(\beta)} = \min(\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_\beta),$$

and $L(\mathcal{P}_1) = M^{(\beta)}$. As a special case, when $\beta = 1$, i.e., $\exists$ only one gateway in S, then $M^{(\beta)} = \mathcal{D}$. When $\beta > 1$, the probability $\Pr(M^{(\beta)} \leq d) = \Pr(\min(\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_\beta) \leq d)$, i.e., at least one of $\{\mathcal{D}_i\}_{i=1}^{\beta}$ is smaller than or equal to d. Therefore, let $F_{\mathcal{D}}(x)$ be the cdf of $\mathcal{D}$, and $F_{M^{(\beta)}}(x)$ the cdf of $M^{(\beta)}$. Then

$$F_{M^{(\beta)}}(x) = 1 - (1 - F_{\mathcal{D}}(x))^{\beta}.$$

Therefore, the pdf of $M^{(\beta)}$ is

$$f_{M^{(\beta)}}(x) = \begin{cases} (1 - F_{\mathcal{D}}(x-1))^{\beta} & \\ \quad - (1 - F_{\mathcal{D}}(x))^{\beta} & \text{for } x \geq 1, \\ \\ 1 - (1 - F_{\mathcal{D}}(0))^{\beta} & \text{for } x = 0. \end{cases}$$

□

A.1.7 Proof of Theorem 3.11

Similar to (3.7), $\mathbb{E}[\mathcal{P}_1] = \mathbb{E}[\mathcal{P}_2] = \ldots = \mathbb{E}[\mathcal{P}_{\mu-1}] \neq \mathbb{E}[\mathcal{P}_{\mu}]$, where $\mu = \lceil q/\tau \rceil$. Therefore, it suffices to determine $\mathbb{E}[\mathcal{P}_1]$ and $\mathbb{E}[\mathcal{P}_{\mu}]$.

In order to obtain $\mathbb{E}[\mathcal{P}_1]$ and $\mathbb{E}[\mathcal{P}_{\mu}]$, we only need to compute $\mathbb{E}[M_k^{(\beta)}]$ and $\mathbb{E}[D_k^{(\beta)}]$. Thus, we define random variable $X^{(k)} := D_{k-1}^{(\beta)} + \mathcal{D} + 1$ (the pdf of $\mathcal{D}$ is in (3.22)). Let $X_1^{(k)}, X_2^{(k)}, \ldots, X_{\beta}^{(k)}$ be i.i.d. r.v. following the same distribution as $X^{(k)}$. Then similar to the proof of Theorem 3.10, we have

$$D_k^{(\beta)} = \min(X_1^{(k)}, X_2^{(k)}, \ldots, X_{\beta}^{(k)}),$$

where $D_1^{(\beta)} = \mathcal{D}$. Analogously, let $Y^{(k)} := D_{k-1}^{(\beta)} + M^{(\beta)} + 1$, and $Y_1^{(k)}, Y_2^{(k)}, \ldots, Y_{\beta}^{(k)}$ be i.i.d. r.v.s following the same distribution as $Y^{(k)}$. Then $M_k^{(\beta)} = \min(Y_1^{(k)}, Y_2^{(k)}, \ldots, Y_{\beta}^{(k)})$, where $M_1^{(\beta)} = M^{(\beta)}$ is defined in Section A.1.6 (proof of Theorem 3.10). Then following the same method in Sections A.1.6–A.1.7, $\mathbb{E}[D_k^{(\beta)}]$ and $\mathbb{E}[M_k^{(\beta)}]$ are computable. Note that it is expensive to compute $D_k^{(\beta)}$ and $M_k^{(\beta)}$, as they are defined in a recursive way; more efficient computation methods are discussed in Section A.1.10.

A.1.8 Proof of Corollary 3.12

When $\beta = 1$, $L_k(1) = \mathbb{E}[D_k^{(1)}] = k\mathbb{E}[\mathcal{D}] + k - 1$, and $L_{k+1}(1) = \mathbb{E}[D_{k+1}^{(1)}] = (k+1)\mathbb{E}[\mathcal{D}] + k$. Therefore, $L_{k+1}(1) - L_k(1) = \mathbb{E}[D_{k+1}^{(1)}] - \mathbb{E}[D_k^{(1)}] = \mathbb{E}[\mathcal{D}] + 1$. □

A.1.9 Proof of Corollary 3.13

In a line network with k domains $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_k$, when $\beta \to \infty$, every node in domain $\mathcal{A}_i$ directly connects to all nodes in domain $\mathcal{A}_{i+1}$ ($i \leq k-1$). As a result, the path cost incurred within each domain on the line network is 0. Thus, the APC is the sum of link preference of all traversed inter-domain edges, which is k for $L_{k+1}(\beta)$ and $k-1$ for $L_k(\beta)$, thus completing the proof. □

A.1.10 Efficient Computation of $\mathbb{E}[D_k^{(\beta)}]$

Since $D_k^{(\beta)}$ is defined in a recursive way, the computation of $\mathbb{E}[D_k^{(\beta)}]$ is relatively complex (see the expression of $X^{(k)}$ in the proof of Theorem 3.11). As such, we establish an efficient strategy to estimate $\mathbb{E}[D_k^{(\beta)}]$. Specifically, let $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_k$ denote i.i.d. random variables following the same distribution as $\mathcal{D}$. Then we define random variable $Z^{(k)} := \sum_{i=1}^{k} \mathcal{D}_i + k - 1$. For the two-layer network model, when the length of the line network is increased by 1, the number of path options w.r.t. two random nodes at the end-domains grows β-fold. Therefore, let $Z_1^{(k)}, Z_2^{(k)}, \ldots, Z_{\beta^{k-1}}^{(k)}$ be i.i.d. random variables following the same distribution as $Z^{(k)}$. Define $\widetilde{D}_k^{(\beta)} := \min(Z_1^{(k)}, Z_2^{(k)}, \ldots, Z_{\beta^{k-1}}^{(k)})$. We then use $\mathbb{E}[\widetilde{D}_k^{(\beta)}]$ to approximate $\mathbb{E}[D_k^{(\beta)}]$. Since $\widetilde{D}_k^{(\beta)}$ does not rely on $\widetilde{D}_{k-1}^{(\beta)}$, $\mathbb{E}[\widetilde{D}_k^{(\beta)}]$ is easily computable using the method in Sections A.1.6–A.1.7.

Remark: As for computation complexity, there are two types of computation when calculating $\mathbb{E}[D_k^{(\beta)}]$ and $\mathbb{E}[\widetilde{D}_k^{(\beta)}]$, i.e., the convolution operation for calculating the pdfs of $X^{(k)}$ and $Z^{(k)}$ (Operation-1), and the operation defined in Theorem 3.10

for calculating the pdfs of $\mathbb{E}[D_k^{(\beta)}]$ and $\mathbb{E}[\widetilde{D}_k^{(\beta)}]$ (Operation-2). The calculation of $\mathbb{E}[D_k^{(\beta)}]$ involves $k-1$ Operation-1 and k Operation-2; whereas the calculation of $\mathbb{E}[\widetilde{D}_k^{(\beta)}]$ involves k Operation-1 and only 1 Operation-2. Therefore, the established method doubles the computation efficiency. Moreover, such efficient approximation method is highly accurate, as validated in evaluation section in the thesis. Intuitively, this is because the process of establishing inter-domain connections in our network model is purely random, which enables us to use the method above to estimate the number of path construction options between two random nodes in the end-domains of a line network of length k.

A.2 Proofs for Lemmas,Theorems, and Corollaries in Chapter 4

A.2.1 Proof of Theorem 4.1

First, we replace relevant notations in (A.8) of Proposition A.2 by the notations listed in Table 4.1, there is $\log(\frac{\phi+(1-\gamma)/z_1}{\gamma}) = \log(\frac{\phi}{\gamma}) + \log(1+\frac{1-\gamma}{n})$. Let $f(\gamma) = \log(1+\frac{1-\gamma}{n})$. It can be observed that $\max(f(\gamma)) = f(1) = 0$. Since $\gamma \leq \frac{n+1}{z_1+1}$, $\min(f(\gamma)) = f(\frac{n+1}{z_1+1}) = \log(\frac{(n+1)z_1}{n(z_1+1)}) < 0$. However, when n and z_1 are relatively large, there is $\min(f(\gamma)) = \log(\frac{(n+1)z_1}{n(z_1+1)}) \approx 0$. Thus, $f(\gamma)$ can be ignored. As such, we use $\frac{\log(\frac{\phi}{\gamma})}{\log(z_2/z_1)} + 1$ for the estimation of minimum APC between the ingress gateway and the egress gateway set, for $\gamma \leq \phi$. Let ξ be the probability that the ingress gateway is also an element in the egress gateway set. Then,

$$l_1 = \begin{cases} \xi\big(\frac{\log(\frac{\phi}{\gamma})}{\log(z_2/z_1)} + 1\big) & \text{if } \gamma \leq \phi, \\ \xi & \text{otherwise,} \end{cases}$$

According to Lemma A.1, the minimum APC between ingress and egress gate-

way sets (the cardinality of both sets are γ) is the same as the minimum APC between an ingress gateway and an egress gateway set whose cardinality is γ^2. Similarly, with both Proposition A.2 and Lemma A.1, we have

$$l_2 = \begin{cases} \zeta\left(\frac{\log(\frac{\phi}{\gamma^2})}{\log(z_2/z_1)} + 1\right) & \text{if } \gamma \leq \sqrt{\phi}, \\ \zeta & \text{otherwise.} \end{cases}$$

Since $\xi \geq \zeta$ and we are concerned with APC lower bound, for simplier expressions in later derivations, we also use ζ in l_1, and thus

$$l_1 \geq \begin{cases} \zeta\left(\frac{\log(\frac{\phi}{\gamma})}{\log(z_2/z_1)} + 1\right) & \text{if } \gamma \leq \phi, \\ \zeta & \text{otherwise.} \end{cases}$$

A.2.2 Proof of Theorem 4.2

The overall APC lower bound is obtained by adding the minimum APC lower bounds of all Type-1 domains, Type-2 domains, and the destination domain on the domain-wise path between the source and destination domains. Specifically, there are μ Type-1 domains, $(m - \mu - 1)$ Type-2 domains, and 1 destination domain on the domain-wise route. The minimum APCs for Type-1 and Type-2 domains are provided in Theory 4.1. As for the destination domain, its minimum APC is the same as Type-1 domains when there are more than one domain in the RC where the destination node belongs; or its minimum APC is estimated by $\frac{\log(\phi)}{\log(z_2/z_1)} + 1$, it constitutes an RC itself. For simpler expressions, the overall APC lower bound is calculated by assuming that there are μ Type-1 domains and $(m - \mu)$ Type-2 domains on the domain-wise path with m domains, i.e., we treat the APC of the destination domain as that of a Type-2 domain. This is valid because L^{LB} is a lower bound expression.

Therefore, we have:

$$\begin{aligned} L^{\text{LB}} &= \frac{\mu\zeta \log\left(\frac{\phi}{\gamma}\right)}{\log(z_2/z_1)} + \frac{(m-\mu)\zeta \log\left(\frac{\phi}{\gamma^2}\right)}{\log(z_2/z_1)} \\ &\quad + m - 1 \\ &= \frac{\mu\zeta \log(\gamma) + m\zeta \log(\frac{\phi z_2/z_1}{\gamma^2})}{log(z_2/z_1)} - 1. \end{aligned}$$

A.2.3 Proof of Theorem 4.3

We derive the bound of the gap between APC and APC lower bound by comparing the APC lower bound to the scenario where no synchronized information is used for routing along the domain-wise path, i.e., routing falls back to the default BGP-like inter-domain routing mechanism. In such a scenario, each domain makes independent routing decisions which minimises the APC in its own domain. This is the worst case since every domain is essentially an RC itself, i.e., $\mu = m$. The APC of each domain except the destination domain is estimated by the minimum APC in Type-1 domains. The APC in the destination domain is approximated by $\frac{\log(\phi)}{\log(z_2/z_1)} + 1$. Furthermore, in the worst case, we assume that the link preference of all links equals the maximum link weight ω. Therefore,

$$\begin{aligned} L - L^{\text{LB}} &< \frac{\omega\zeta(m-1)\log(\frac{\phi}{\gamma})}{\log(z_2/z_1)} + \frac{\omega\log(\phi)}{\log(z_2/z_1)} + \omega \\ &\quad - L^{\text{LB}} \\ &= \frac{\rho\zeta\log(\gamma) + m(\omega-1)\zeta\log(\phi)}{\log(z_2/z_1)}, \end{aligned}$$

where $\rho = (2-\omega)m + \omega - \mu$.

A.2.4 Proof of Theorem 4.4

When $\sqrt{\phi} < \gamma \leq \phi$, the minimum APC for all Type-2 domains reduces to ζ, whereas the minimum APC for Type-1 domains remains unchanged. Thus,

$$\begin{aligned} L^{\text{LB}} &= \frac{\mu\zeta \log\left(\frac{\phi}{\gamma}\right)}{\log(z_2/z_1)} + (m-\mu)\zeta + m - 1 \\ &= \frac{\mu\zeta \log(\frac{\phi}{\gamma z_2/z_1}) + m(\zeta+1)\log(\frac{z_2}{z_1})}{\log(z_2/z_1)} - 1. \end{aligned}$$

A.2.5 Proof of Theorem 4.5

This proof uses the same method described in the proof of Theorem 4.3.

$$\begin{aligned} L - L^{\text{LB}} &< \frac{\omega\zeta(m-1)\log(\frac{\phi}{\gamma})}{\log(z_2/z_1)} + \frac{\omega\log(\phi)}{\log(z_2/z_1)} + \omega \\ &\quad - L^{\text{LB}} \\ &= \frac{\tau\zeta\log(\frac{\phi}{\gamma}) + \omega\log(\phi)}{\log(z_2/z_1)} - \zeta(m-\mu), \end{aligned}$$

where $\tau = \omega(m-1) - \mu$.

A.2.6 Proof of Theorem 4.6

When $\gamma > \phi$, the APC lower bounds for both Type-1 and Type-2 doamins reduce to ζ. Thus, the overall APC lower bound for the domain wise path becomes $L^{\text{LB}} = \zeta\mu + \zeta(m-\mu) + m - 1 = (1+\zeta)m - 1$.

A.2.7 Proof of Theorem 4.7

This proof uses the same method described in the proof of Theorem 4.3, except that minimum APC in Type-1 domains now reduces to ζ.

$$\begin{aligned} L - L^{\text{LB}} &< \omega\zeta(m-1) + \frac{\omega\log(\phi)}{\log(z_2/z_1)} + \omega - L^{\text{LB}} \\ &= \frac{\omega\log(\phi)}{\log(z_2/z_1)} + \zeta\big((m-1)\omega - m\big). \end{aligned}$$

A.2.8 Lemma A.1 and its proof

Under the two-layer network model, let $\mathcal{I}$ and $\mathcal{E}$ denote the ingress and egress gateway sets in a Type-2 domain on the domain-wise path, where $\gamma = |\mathcal{I}| = |\mathcal{E}|$ is the cardinality of ingress and egress gateway sets. Let $\mathcal{L}_{\mathcal{IE}}$ denote the APC between set $\mathcal{I}$ and $\mathcal{E}$, and $\mathcal{L}_{\gamma^2}$ the APC between a randomly chosen node and a random chosen node set with cardinality γ^2 in this domain.

Lemma A.1. For any Type-2 domain, there is $\mathcal{L}_{\mathcal{IE}} = \mathcal{L}_{\gamma^2}$.

Proof. Let random variable (r.v.) $M^{(\gamma)}$ denote the shortest distance from an arbitrary node w to the closest gateway in the candidate gateway set S within a domain, where S contains all gateways connecting to the same neighbouring domain. In addition, let r.v. $\mathcal{D}$ be the shortest distance between two random nodes in one domain. By definition, we have $M^{(\gamma)} = \min(\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_\gamma)$, where $\mathcal{D}_i$s ($1 \le i \le \gamma$) are i.i.d. r.v.s which share the same distribution as r.v. $\mathcal{D}$. Then, let $M_{\mathcal{L}_{\mathcal{IE}}}$ and $M_{\mathcal{L}_{\gamma^2}}$ be the R.V.s of $\mathcal{L}_{\mathcal{IE}}$ and $\mathcal{L}_{\gamma^2}$. According to the definition of $\mathcal{L}_{\mathcal{IE}}$, we have $M_{\mathcal{L}_{\mathcal{IE}}} = \min(M_1^{(\gamma)}, M_2^{(\gamma)}, \ldots, M_\gamma^{(\gamma)})$, where $M_j^{(\gamma)}$s ($1 \le j \le \gamma$) are i.i.d. r.v.s which share the same distribution as r.v. $M^{(\gamma)}$. Then $M_{\mathcal{L}_{\mathcal{IE}}} = \min(\mathcal{D}_1, \mathcal{D}_2 \ldots, \mathcal{D}_{\gamma^2}) = M_{\mathcal{L}_{\gamma^2}}$. Therefore, there is $\mathbb{E}[M_{\mathcal{L}_{\mathcal{IE}}}] = \mathbb{E}[M_{\mathcal{L}_{\gamma^2}}]$, and thus $\mathcal{L}_{\mathcal{IE}} = \mathcal{L}_{\gamma^2}$. □

A.2.9 Proposition A.2

Proposition A.2. [32], [34] In an undirected connected graph $\mathcal{H}$ with n_0 vertices and the vertex degree satisfying a given distribution, let x_i be the average number of vertices that are i-hop away from a random vertex in $\mathcal{H}$. Suppose all edge weights are 1, and $x_2 \gg x_1$. Then,

a)

$$x_i = (x_2/x_1)^{i-1} x_1; \tag{A.6}$$

b) APC between two arbitrary nodes in $\mathcal{H}$ is

$$\frac{\log(n_0/x_1)}{\log(x_2/x_1)} + 1; \tag{A.7}$$

c) APC between an arbitrary node and η arbitrary nodes in $\mathcal{H}$ (see Definition 11) is

$$\begin{cases} \frac{\log(\frac{n_0+1-\eta}{x_1\eta})}{\log(x_2/x_1)} + 1 & \text{if } \eta \leq \frac{n_0+1}{x_1+1}, \\ 1 & \text{otherwise.} \end{cases} \tag{A.8}$$

A.3 Proofs for Lemmas,Theorems, and Corollaries in Chapter 8

A.3.1 Proof for Theorem 8.1

First, we consider the scenario of the mMRP starting from a specific initial state state s_0. Then, the policy gradient is calculated by $\nabla_{\boldsymbol{\theta}} J(\mu_{\boldsymbol{\theta}}) = \int_{s_0 \in \mathcal{S}} p_0(s_0) \nabla_{\boldsymbol{\theta}} V^{\mu_{\boldsymbol{\theta}}}(s_0) \mathrm{d}s_0$, where $p_0(s_0)$ is the initial distribution of state s_0. Therefore, we first derive the expression of $\nabla_{\boldsymbol{\theta}} V^{\mu_{\boldsymbol{\theta}}}(s)$. Before we proceed with the derivation, we define some notations used as follows. $p(s'|s, \mu_{\boldsymbol{\theta}}(s))$ is the probability of transition into s' from s, given the policy $\mu_{\boldsymbol{\theta}}$. Although $\mu_{\boldsymbol{\theta}}$ is a deterministic policy, the next state s' is also

influenced by other environment factors. For example, if the next state suggested by $\mu_{\boldsymbol{\theta}}$ is not a feasible next state, a mapping is needed to map the raw next state indicated by the policy to a feasible next state s'. In addition, $p(s \to s', t, \mu_{\boldsymbol{\theta}})$ is the probability of transitioning from state s to state s' in t steps under policy $\mu_{\boldsymbol{\theta}}$.

$$
\begin{aligned}
\nabla_{\boldsymbol{\theta}} V^{\mu_{\boldsymbol{\theta}}}(s) &= \nabla_{\boldsymbol{\theta}}(r_{s,s'} + \gamma \Phi^{\mu_{\boldsymbol{\theta}}}(s', \mu(s'))) \\
&= \nabla_{\boldsymbol{\theta}} \mu_{\boldsymbol{\theta}}(s) \nabla_{s'} r_{s,s'} + \nabla_{\boldsymbol{\theta}} \int_{s' \in \mathcal{S}} \gamma p(s'|s, \mu_{\boldsymbol{\theta}}(s)) V^{\mu_{\boldsymbol{\theta}}}(s') \mathrm{d}s' \\
&= \nabla_{\boldsymbol{\theta}} \mu_{\boldsymbol{\theta}}(s) \nabla_{s'} r_{s,s'} \\
&\quad + \int_{s' \in \mathcal{S}} \gamma \Big(p(s'|s, \mu_{\boldsymbol{\theta}}(s)) \nabla_{\boldsymbol{\theta}} V^{\mu_{\boldsymbol{\theta}}}(s') + \nabla_{\boldsymbol{\theta}} \mu_{\boldsymbol{\theta}}(s) \nabla_{s'} p(s'|s, \mu_{\boldsymbol{\theta}}(s)) V^{\mu_{\boldsymbol{\theta}}}(s') \Big) \mathrm{d}s' \\
&= \nabla_{\boldsymbol{\theta}} \mu_{\boldsymbol{\theta}}(s) \nabla_{s'} \Big(r_{s,s'} + \int_{s' \in \mathcal{S}} \gamma p(s'|s, \mu_{\boldsymbol{\theta}}(s)) V^{\mu_{\boldsymbol{\theta}}}(s') \mathrm{d}s' \Big) \\
&\quad + \int_{s' \in \mathcal{S}} \gamma p(s'|s, \mu_{\boldsymbol{\theta}}(s)) \nabla_{\boldsymbol{\theta}} V^{\mu_{\boldsymbol{\theta}}}(s') \mathrm{d}s'
\end{aligned}
\tag{A.9}
$$

For simpler expression, we denote $\alpha(s) = \nabla_{\boldsymbol{\theta}} \mu_{\boldsymbol{\theta}}(s) \nabla_{s'} \Big(r_{s,s'} + \int_{s' \in \mathcal{S}} \gamma p(s'|s, \mu_{\boldsymbol{\theta}}(s)) V^{\mu_{\boldsymbol{\theta}}}(s') \mathrm{d}s' \Big)$, which can be further simplified as $\alpha(s) = \nabla_{\boldsymbol{\theta}} \mu_{\boldsymbol{\theta}}(s) \nabla_{s'} \Phi^{\mu_{\boldsymbol{\theta}}}(s, s')$. Then, we have the following.

$$
\begin{aligned}
\nabla_{\boldsymbol{\theta}} V^{\mu_{\boldsymbol{\theta}}}(s) &= \alpha(s) + \int_{s' \in \mathcal{S}} \gamma p(s'|s, \mu_{\boldsymbol{\theta}}(s)) \nabla_{\boldsymbol{\theta}} V^{\mu_{\boldsymbol{\theta}}}(s') \mathrm{d}s' \\
&= \alpha(s) + \int_{s' \in \mathcal{S}} \gamma p(s \to s', 1, \mu_{\boldsymbol{\theta}}) \nabla_{\boldsymbol{\theta}} V^{\mu_{\boldsymbol{\theta}}}(s') \mathrm{d}s' \\
&= \alpha(s) + \int_{s' \in \mathcal{S}} \gamma p(s \to s', 1, \mu_{\boldsymbol{\theta}}) \Big(\alpha(s') + \int_{s'' \in \mathcal{S}} \gamma p(s' \to s'', 1, \mu_{\boldsymbol{\theta}}) \nabla_{\boldsymbol{\theta}} V^{\mu_{\boldsymbol{\theta}}}(s'') \mathrm{d}s'' \Big) \mathrm{d}s' \\
&= \alpha(s) + \int_{s' \in \mathcal{S}} \gamma p(s \to s', 1, \mu_{\boldsymbol{\theta}}) \alpha(s') \mathrm{d}s' \\
&\quad + \int_{s'' \in \mathcal{S}} \int_{s' \in \mathcal{S}} \gamma^2 p(s \to s', 1, \mu_{\boldsymbol{\theta}}) p(s' \to s'', 1, \mu_{\boldsymbol{\theta}}) \nabla_{\boldsymbol{\theta}} V^{\mu_{\boldsymbol{\theta}}}(s'') \mathrm{d}s' \mathrm{d}s'' \\
&= \alpha(s) + \int_{s' \in \mathcal{S}} \gamma p(s \to s', 1, \mu_{\boldsymbol{\theta}}) \alpha(s') \mathrm{d}s' + \int_{s'' \in \mathcal{S}} \gamma^2 p(s \to s'', 2, \mu_{\boldsymbol{\theta}}) \nabla_{\boldsymbol{\theta}} V^{\mu_{\boldsymbol{\theta}}}(s'') \mathrm{d}s'' \\
&= \vdots \\
&= \int_{x \in \mathcal{S}} \Sigma_{t=0}^{\infty} \gamma^t p(s \to x, t, \mu_{\boldsymbol{\theta}}) \alpha(s) \mathrm{d}x
\end{aligned}
\tag{A.10}
$$

Then, we replace s with s_0 in $\nabla_{\boldsymbol{\theta}} J(\mu_{\boldsymbol{\theta}}) = \int_{s \in \mathcal{S}} p_0(s) \nabla_{\boldsymbol{\theta}} V^{\mu_{\boldsymbol{\theta}}}(s) \mathrm{d}s$, and use the result in (A.10), we obtain

$$
\begin{aligned}
\nabla_{\boldsymbol{\theta}} J(\mu_{\boldsymbol{\theta}}) &= \int_{s_0 \in \mathcal{S}} p_0(s_0) \nabla_{\boldsymbol{\theta}} V^{\mu_{\boldsymbol{\theta}}}(s_0) \mathrm{d}s_0 \\
&= \int_{s_0 \in \mathcal{S}} \left(\int_{s \in \mathcal{S}} \Sigma_{t=0}^{\infty} \gamma^t p_0(s_0) p(s_0 \to s, t, \mu_{\boldsymbol{\theta}}) \mathrm{d}s \right) \alpha(s_0) \mathrm{d}s_0 \\
&= \int_{s_0 \in \mathcal{S}} \rho^{\mu_{\boldsymbol{\theta}}}(s_0) \nabla_{\boldsymbol{\theta}} \mu_{\boldsymbol{\theta}}(s_0) \nabla_{s_0'} \Phi^{\mu_{\boldsymbol{\theta}}}(s_0, s_0') \mathrm{d}s_0 \\
&= \mathbb{E}_{s \sim \rho^{\mu_{\boldsymbol{\theta}}}} \Big[\nabla_{\boldsymbol{\theta}} \mu_{\boldsymbol{\theta}}(s) \nabla_{s'} \Phi^{\mu_{\boldsymbol{\theta}}}(s, s')|_{s' = \mu_{\boldsymbol{\theta}}(s)} \Big].
\end{aligned}
\tag{A.11}
$$

Note that the results in (A.11) is suitable for calculating on-policy policy gradient since the states are sampled by the policy being updated, i.e., $\mu_{\boldsymbol{\theta}}$. For off-policy policy gradient scenarios, where the policy-gradient, denoted by $\nabla_{\boldsymbol{\theta}} J_{\beta}(\mu_{\boldsymbol{\theta}})$, is calculated using states sampled from the behaviour policy β, (A.11) is updated as follows.

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} J_\beta(\mu_{\boldsymbol{\theta}}) &= \int_{s\in\mathcal{S}} \rho^\beta(s)\nabla_{\boldsymbol{\theta}} V^{\mu_{\boldsymbol{\theta}}}(s)\mathrm{d}s \\ &\approx \int_{s\in\mathcal{S}} \rho^\beta(s)\nabla_{\boldsymbol{\theta}}\mu_{\boldsymbol{\theta}}(s)\nabla_{s'}\Phi^{\mu_{\boldsymbol{\theta}}}(s,s')\mathrm{d}s \\ &= \mathbb{E}_{s\sim\rho^\beta}\left[\nabla_{\boldsymbol{\theta}}\mu_{\boldsymbol{\theta}}(s)\nabla_{s'}\Phi^{\mu_{\boldsymbol{\theta}}}(s,s')|_{s'=\mu_{\boldsymbol{\theta}}(s)}\right]\end{aligned} \tag{A.12}$$

Here, the second term related to the partial derivative of in $\Phi^{\mu_{\boldsymbol{\theta}}}(s,s')$ is dropped, and thus the approximation. This approximation is first used and justified in [108], and has since been commonly used in various policy gradient derivations.

APPENDIX B

Experiment and Implementation Details for sasRL

B.1 Experiment Scenario Details

1). *Grid world exit problem.* The state vector spans two-dimensional real coordinate space, i.e., $\mathcal{S} \in \mathbb{R}^2$. All state vector elements are decimal numbers between 0 and 1, which are the Cartesian coordinates of the agent on the two-dimensional plane. The landmine and exit locations are small regions on the Cartesian plane with each taking a 0.1×0.1 square area. For our experiments, we implement 1 landmine location and 1 exit location only for faster training. It is assumed that the agent always starts from the origin of the two-dimensional Cartesian plane. The action vector is also two-dimensional with its elements specifying the amounts of movement of the agent along the horizontal and vertical directions on the Cartesian plane at a time step. Note that the range of each element of the action vector is between -0.09 and 0.09. The reward of the agent consists of three parts: (i) a negative reward of -0.1 at each time step to penalize time consumption, as the goal of the agent is to leave find the exit as soon as possible; (ii) a negative reward of -2 if the agent lands in the landmine location; (iii) a positive reward of 5 if the agent finds the exit and leaves the grid world. The maximum number of time steps per game is set to 50. The game ends automatically if the agent finds the exit before the time step limit is reached. The agent is put back to the origin when it hits a landmine while the old time step

count still applies.

2). *Berzerk-like game.* The state vector consists of 12 elements, which correspond to the coordinates of the agent, 3 patrolling robots, 1 wall, and 1 exit. In particular, the robots' patrolling routines are fixed, i.e., at each time step they move to the next locations on the pre-defined routes. The action vector includes two elements which are the amount of movements of the agent along the horizontal and vertical directions, respectively. Note that at every time step, a bullet is fired towards the agent's direction of travel. The bullet can kill 1 robot if there are any on the bullet's trajectory within the same time slot of the bullet's firing. The reward consists of 4 parts as follows: (i) a negative reward of -0.2 at each time step to penalize time consumption; (ii) a negative reward of -5 if the agent is killed by colliding with a robot; (iii) a positive reward of 3 if a robot is killed by the bullet; (iv) a positive reward of 6 if the agent exits the map alive. The maximum number of time steps per game is set to 50. The game ends automatically if either the agent is killed by collisions with a robot or the agent leaves the current map through the exit location before the time step limit is reached.

3). *The slot machine gambling game.* The slot machine in this experiment has 10 reels and 10 symbols on each reel. Note that the same symbol can appear multiple times on a reel. The state is defined as the symbols on display when all reels stop spinning. Therefore, the state vector consists of 10 elements. The value of each state element is a decimal number between 0 and 1, which is uniquely mapped to the symbol on display. Each symbol is therefore represented by the decimal numbers within a window of size 0.1. For example, the decimal values between 0 and 0.1 represent the first symbol, values between 0.1 and 0.2 represent the second symbol, etc. The action vector has 10 elements which correspond to the amount of spinning for each reel. In particular, the value of every action element is a decimal number between -1 and 1, which indicates the amount of spinning for each reel. A positive

value indicates spinning clockwise and a negative value the opposite. For example, if the value of an action element is 1, that means the corresponding reel spins for a full cycle and the same symbol will be on display as a result when the reel stops. The reward is associated with the type and the number of symbols on display when all reels stop spinning. In particular, each symbol is associated with a different value. For each symbol, the number of that symbol on display among all reels must reach a threshold of 3 to trigger a payout. The payout of a symbol is then calculated as the product of the number of that symbol and the value of the symbol. The reward is calculated as the sum of payouts triggered by all symbols that reach the threshold. For instance, assume that symbol a carries a value of 3, symbol b carries a value of 4, and symbol c carries a value of 5. Then, assume that among 10 symbols on display when all reels stop, there are 5 symbol a, 3 symbol b, and 2 symbol c. The reward is $5 \times 3 + 3 \times 4 = 27$. For each game, the player can spin all reels 20 times.

B.2 Implementation Details for the sasRL

The actor, the critic, and the transition model in the sasRL embodiment used in our experiment are all implemented by multilayer perceptrons (MLPs).

The actor has one input layer, two hidden layers, and one output layer. The number of neurons in both the input layer and the output layer is the same as the dimension of the state vector. There are 64 neurons in both hidden layers. The rectified linear unit (ReLU) is selected as the activation function for all neurons in hidden layers, whereas sigmoid is employed as the activation function for the output layer.

The critic has two input layers, three hidden layers, and one output layer. The input layer takes the current and the next state vectors as inputs, it then concatenate the two inputs to be fed to the hidden layers. Therefore, the number of neurons in

both input layers equals the dimension of the state vector. There are 128, 64, and 32 neurons in three hidden layers, respectively. The output of the the critic corresponds to the predicted STV. Therefore, there is 1 neuron in the output layer. The rectified linear unit (ReLU) is selected as the activation function for all neurons in hidden layers, whereas no activation function is used in neuron in the output layer.

The transition model also has two input layers, two hidden layers, and one output layer. The input layer takes the current and the next state vectors as inputs. Therefore, the number of neurons in the input layer is twice the dimension of the state vector. There are 64 and 32 neurons in two hidden layers, respectively. The output of the the transition model corresponds to the predicted action that causes the input state transition. Therefore, the number of neurons in the output layer is equal to the dimension of the action vector. The rectified linear unit (ReLU) is selected as the activation function for all neurons in hidden layers, whereas the hyperbolic tangent activation (tanh) is used the neuron in the output layer. Note that for the training of the transition model, the training data are pre-processed to avoid the situations where the same state transition is caused by multiple actions, resulting in various training targets for the same input (s, s'). Specifically, the pre-processing procedure reserves only one action as the training target for a state transition.

We use the Keras deep learning library for building and training the MLPs, which represent the actor, the critic, and the transition model described above. In particular, we use mean square error (mse) and the Adam optimizer for estimating and minimizing training losses. The settings for the Adam optimizer are as follows: $\texttt{lr} = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \texttt{clipnorm} = 1.0$. The training procedure uses stochastic gradient descents operating on minibatches of data (the minibatch size is 32) for gradient updates. The policy being trained is evaluated every 20 gradient update steps. For one evaluation, the corresponding game is played according to current policy and the score at the end of the game is recorded.

For DNN weight initializations, we use the Glorot uniform initializer (also known as Xavier uniform initializer) with default settings implemented in Keras. The soft update procedure updates the delayed version of the actor and critic parameters by 10% (i.e., $\varepsilon = 0.1$ in line 9 of Algorithm 1), after each update of the corresponding non-delayed parameters. The discount factor for the STV function is set to $\gamma = 0.8$.

B.3 Details on the Ablation Study for Action Space Granularity

For the grid world scenario, the coarse action space corresponds to 4 canonical actions, i.e., up, down, left, or right, whereas the fine-grained action space adds a further 4 diagonal actions, thus resulting in 8 canonical actions. As for the berzerk scenario, the agent's horizontal and vertical movements are quantized to certain number of actions. The agent can pick action(s) from the allowed action set for moving towards horizontal and/or vertical directions at a time slot. The number of quantized actions along each direction is 5 (or 10) for the coarse-grained (or the fine-grained) discrete action case. Finally, for the slot machine scenario, the default continuous action space means that the player can decide freely for how long each reel spins, as long as the a given maximum limit is not exceeded. For the discrete action space, the given maximum limit is quantized to several time intervals; the agent decides the time to spin for each reel by choosing from these time intervals.

B.4 Details on the Replay Buffer

The size of the replay buffer is fixed to 50000. Initially, the reply buffer is filled with data samples generated by the random behaviour policy. During training, new samples are added when all samples in the replay buffer are used for training once. For

each replay buffer update, 500 existing samples (or 1% of all samples) are replaced with new data generated by a mix of random policy and the policy being trained. The replay buffer is randomly shuffled every time after new samples are added to it.

B.5 Detailed Settings for the Experiment in Section 8.7

The server processing power (the number of time slots needed for processing an unit of request) of 3 services at a time slot are uniformly drawn from the sets as follows. Service 1: $\{1, 2, 3\}$ for the first server and $\{3, 4, 5\}$ for the second server; Service 2: $\{2, 3, 4\}$ for the first server and $\{5, 6, 7\}$ for the second server; Service 3: $\{4, 5, 6\}$ for the first server and $\{9, 10, 11\}$ for the second server. Request arrivals at all times slots for 3 services are modelled by Bernoulli distribution with $p = 0.8$, $p = 0.5$, $p = 0.3$, respectively.