12-2019

Toward Self-Reconfigurable Parametric Systems: Reinforcement Learning Approach

Ting-Yu Mu
Western Michigan University, tingyu.mu@icloud.com

TOWARD SELF-RECONFIGURABLE PARAMETRIC SYSTEMS: REINFORCEMENT LEARNING APPROACH

by

Ting-Yu Mu

A dissertation submitted to the Graduate College
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
Computer Science
Western Michigan University
December 2019

Doctoral Committee:

Ala Al-Fuqaha, Ph.D., Chair
Ajay Gupta, Ph.D.
Fahad Saeed, Ph.D.
J. Michael Tarn, Ph.D.

ACKNOWLEDGEMENTS

I would like to begin by presenting my gratitude to my advisor, Dr. Ala Al-Fuqaha for all his dedication and contribution to this work, and my academic career. Who, without his dedicated efforts and efficient guidance this work would have not been possible. I also like to thank Dr. Ajay Gupta, Dr. Fahad Saeed, and Dr. J. Michael Tarn for all their precious insight and contributions into this research, and the valuable knowledge and suggestions they have put into this work and helped me learn throughout my works.

Secondly, I would like to thank my family, friends, and love ones, especially my wife Shiao-Min Lu for all their support and encouragement, without which, this work would have not been possible.

Lastly, I would like to thank my father Ching-Ping Mu, and mother Ching-Chih Pao, for all they have done for me both for my education and personal life; without which, none of this would be possible.

Ting-Yu Mu

TOWARD SELF-RECONFIGURABLE PARAMETRIC SYSTEMS: REINFORCEMENT LEARNING APPROACH

Ting-Yu Mu, Ph.D.

Western Michigan University, 2019

For the ongoing advancement of the fields of Information Technology (IT) and Computer Science, machine learning-based approaches are utilized in different ways in order to solve the problems that belong to the Nondeterministic Polynomial time (NP)-hard complexity class or to approximate the problems if there is no known efficient way to find a solution. Problems that determine the proper set of reconfigurable parameters of parametric systems to obtain the near optimal performance are typically classified as NP-hard problems with no efficient mathematical models to obtain the best solutions. This body of work aims to advance the knowledge of machine learning techniques for the adaptive applications that depend on the set of configurable parameters, particularly the reinforcement learning approach, to address such problems. This work focuses on applying reinforcement learning to two chosen applications: (1) the MapReduce framework, and (2) the flow management in Software-Defined Networking (SDN). To demonstrate the effectiveness of this work, three studies were conducted: 1- The literature review of SDN technologies, architectures, applicable applications, underlying protocols, and deployments. 2- The use of reinforcement learning algorithm to search for the set of reconfigurable parameters that yields the near optimal performance automatically and adaptively, including the development of the simulator for evaluating the results and applying the obtained results to the real world application, and 3- The feasibility for the development of the emulation-based environment for the utilization of reinforcement learning in flow management in SDN, in order to address the limited capacity issue of the Ternary Content-Addressable Memory (TCAM) installed in an OpenFlow enabled switch. The MapReduce simulator was implemented using Network Simulator 3 (NS-3) simulation engine and the SDN emulation was built on top of Mininet network emulator. The results of these studies provide information on the effectiveness and efficiency of reinforcement learning algorithms for improving the performance of the parametric systems adaptively and automatically.

TABLE OF CONTENTS

Table of Contents - Continued

LIST OF TABLES

LIST OF FIGURES

List of Figures - Continued

CHAPTER 1

INTRODUCTION

With the evolution of Information Technology, adaptive frameworks were developed not only to address the proposed issues, but also have the capability to respond to environmental changes, including available computing and networking resources that the frameworks running with. To provide the adaptability of the framework, sets of reconfigurable parameters are utilized, and properly configuring these parameters are essential in achieving better performance and high stability of the framework. However, an optimal solution comes with the expensive cost for extremely long searching time, since there is no effective mathematical models to capture the dynamics of the heterogeneity of the computing resources and the applications executing on the environment. Based on the fact that the search space is comprised of several performance critical multi-valued parameters and their complex relationships, the machine learning based approaches fit this scenario better since they can capture the characteristics of the application and its relationships among those parameters from various scheduled tasks [153], [162]. Therefore, the goal of this work is to characterize the effectiveness of the machine learning based approach, specifically the reinforcement learning algorithms, to perform the best-effort operation to obtain the set of reconfigurable parameters that yields the near optimal performance of the given framework. This chapter provides a brief introduction to the field of machine learning and discusses some common nomenclature. Additionally, the motivations and objectives are also discussed.

1.1 Machine Learning

From the beginning of the technological era, computer scientists have vision of making computers to reason and make intelligent decisions like humans do, by performing objects classification and patterns extraction from complex information without explicit instructions. To one aspect of the goal, machine learning is a model that takes the input and produces the desired outcome, and the model can be considered as the process of the approximation [2], [4] that we would like the machines to perform.

Artificial intelligence (AI) is the common term that usually appears with machine learning. As depicted in Figure 1, the core building block of AI is machine learning, which contains statistical algorithms that are capable to produce generalizable models by processing the given dataset. Although there are other approaches that can be classified under AI, but they would not be necessarily considered as the subset of machine learning. Generally speaking, there are multiple techniques to create AI, and machine learning is one of them. As stated by Professor Tom Mitchell from Carnegie Mellon University for the definition of artificial intelligence: "*A computer program is said to learn from experience **E** with respect to some class of task **T** and performance measure **P**, if its performance at tasks in **T**, as measured by **P**, improves with experience **E***" [1].

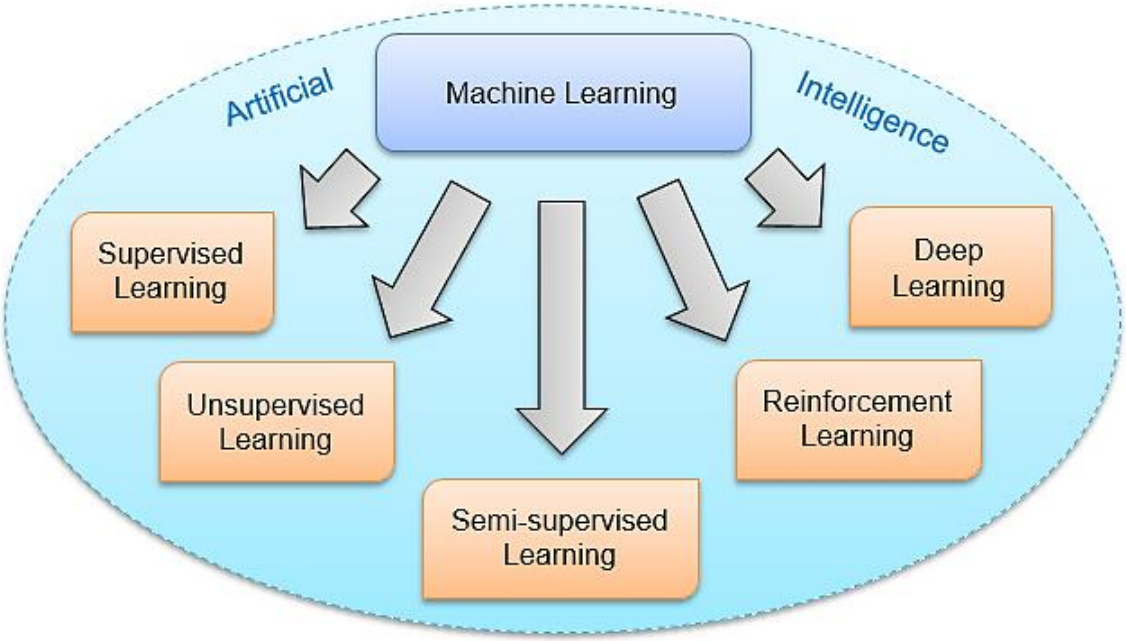


Figure 1. The relationship between artificial intelligence and machine learning, and the type of learning techniques

Within the machine learning domain, the learning techniques can be further categorized into the following types:

- *Supervised Learning*: This type of machine learning model is used to predict the outcome based on the given dataset. The objective of this learning model is to infer a mapping or function from the labeled training data. Each training data consist of a vector of input data with its corresponding output value. Specifically,

this type of learning algorithm utilizes a technique namely *inductive bias*, which is defined as the set of assumptions used by learning agent to predict outputs of the given inputs that are not part of training data (unlabeled).

- *Unsupervised Learning*: Contrary to supervised learning, unsupervised learning does not have supervisor or labeled training data, and the objective of this model is to discover a hidden structure in the given unlabeled data. The structure is usually obtained by clustering the input data based on the relationships. A good representation of this model is the *k*-means clustering algorithm, where it emphasizes on partitioning *n* observations into *k* clusters in which each cluster has the nearest mean among its corresponding observations.
- *Semi-supervised Learning*: In this type of learning, the given data is the combination of labeled and unlabeled dataset used to produce an appropriate model for data classification. In most cases, the minority of the data is labeled, whereas the majority of the data is unlabeled, and this mixture of data types has proven to have the considerable significance for improving learning model accuracy when the unlabeled data is utilized in conjunction with the labeled data [2], [4]. Intuitively, this learning process can be considered as the way we solve problems where the labeled data represents the solved problems, and the unlabeled data is the unsolved problems that we are going to solve for which they might or might not be related to the solved problems.
- *Reinforcement Learning*: This machine learning technique, differs from aforementioned techniques, does not need to know the correct input/output pairs at the beginning phase, with clear objective used to drive learning agent to establish the policy that maximizes the long-term cumulative satisfaction from unknown environment [3]. Through the interactions with the unknown environment, reinforcement learning has additional responsibility for making decisions based on the feedback supplied by the environment for the corresponding rewards. Unlike supervised learning, the results produced by reinforcement learning are not immediate and may require sequence of steps to be performed before the final result is acquired. In particular, the following steps are utilized by reinforcement learning:

1. The input state space as observed by the learning agent.
2. The function for making decisions to drive the agent to perform an action.
3. The agent receives reward from the environment based on action performed.
4. The reward information of the state-action pair is stored as the experienced matrix.

- *Deep Learning*: Deep learning technique is considered as the subset of *Artificial Neural Networks* (ANNs). In the relationship with artificial neural networks, this model is an advancement of ANNs that can be employed to process large amount of dataset to derive the representations of the data [5]. The term "deep", as depicted in Figure 2, is defined by multiple layers used in neural network model as there is no constant value that defines how many hidden layers should be used. The learning process of this model can be supervised, semi-supervised or unsupervised [4], [5], for which the features of the data is extracted and transformed by using multiple layers of nonlinear processing units. As represented in Figure 2, each connected layers (except for input layer) uses the output from the previous layer as input, and the multiple hidden layers can be used as the representation of the different levels of abstraction, based on the applications of the learning.

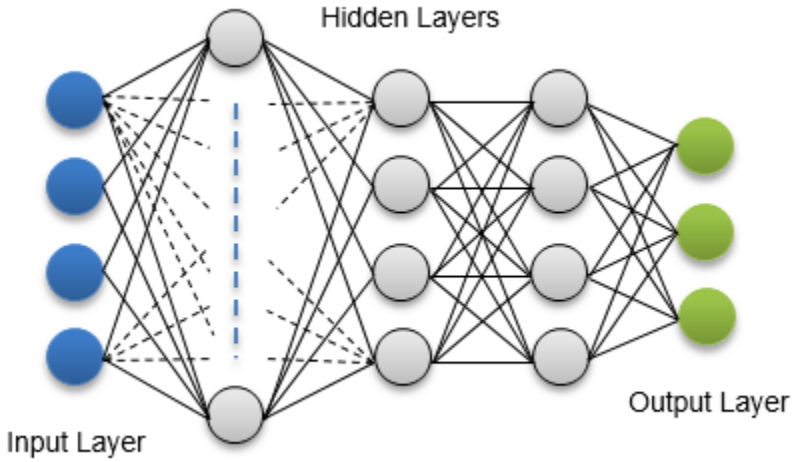


Figure 2. The graphical representation of a deep neural network

Generally, when utilizing machine learning in the context of the application, two types of datasets are required: one is manually constructed dataset where the input data and the corresponding output data are available, and it is crucial that each input has an expected output as this is the required condition for supervised learning to establish the rule or policy; another type of dataset is where we have the input data and there is no expected output as we are expecting the machine learning to perform the prediction or approximation for the output. In addition, those datasets can be further classified into training, validation, and testing dataset respectively, where training dataset is used by the machine learning to gain the knowledge from, the validation dataset is used to verify the established rule for the accuracy tuning of the output, and the testing dataset is used to assess the performance of the established rule or policy.

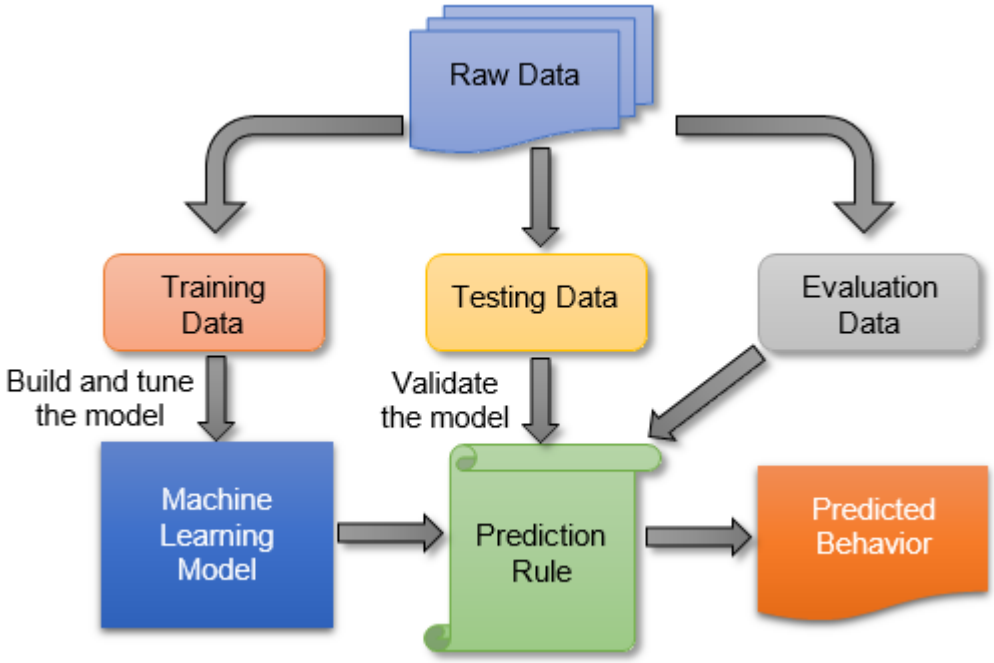


Figure 3. The relatiohships between phases and how the trained model can be applied to predict the behavior

To perform machine learning, there are three phases involved in the process: **Phase 1** is training phase, where the learning model is trained with <*input*, *output*> pair in order to build the learning rule. **Phase 2** is validation and test phase used to evaluate the quality of the trained learning model in terms of measurement in error rate, accuracy,

etc. **Phase 3** is the application phase where the trained model is deployed onto the real-world application for production process. The relationships between the phases are illustrated in Figure 3.

To further elaborate the types of problems that machine learning can be employed, those problems can be categorized as follow:

- *Classification*: This is a problem to determine or establish a grouping technique for a given dataset based on its target output or attribute, and the entire dataset can be classified to belong to a class by using this grouping technique, which is also known as discrimination mechanism [2], [4].
- *Clustering*: Differs from classification, clustering is a technique to extract the interesting patterns of the given dataset without an explicit definition of the target output. As in aforementioned type of unsupervised learning, this technique emphasizes on partitioning *n* observations into *k* clusters in which each cluster has the nearest mean among its corresponding observations.
- *Prediction or regression*: Similar to classification, this problem is used to determine the way things would happen in the future by using the information derived from past experience or knowledge, which is also called as rule extraction.
- *Simulation*: This is the problem where we need to address the uncertainty of the given dataset since we do not know how the data is correlated and there is no equation to describe the relationship among the data attributes. This type of problem can be used to solve challenging issues including robots design, behavior prediction, big data information extraction, etc.

Based on the preceding discussed characteristics of machine learning, these techniques are frequently utilized to address challenging problems where constructing the mathematic model to extract information from dynamics of the application is extremely difficult. In addition, the machine learning based usage is rapidly increasing among research fields and business organizations. Every organization is actively devising the strategies on how to utilize their data properly to increase the profits and establish new businesses to accommodate the evolving economy.

Moreover, in the field of data analytics, machine learning is also employed as the method to construct complex models or algorithms as the guideline for prediction, which is also known as predictive analytics [4]. These techniques allow researchers, scientists, and analysts to unveil the *hidden insights* through the use of machine learning based on the historical and future trends of the data.

1.2 Motivations for Study

The use of reinforcement learning in our approach is inspired by the modern robotic learning research where a robot is allowed to obtain skills and necessary capabilities to adapt the external environment that robot interacts with, and the responses to different stimuli provided by the environment. Besides, the robot is also capable of using learned experience to further strengthen its future behavior through the use of reinforcement.

To further extend the scope of the application, we believe the use of reinforcement learning algorithm can also help us achieve the objective of self-reconfigurable mechanism in tuning parametric systems that allow the learning agent to explore the configurations space through the given feedback provided by the environment based on the action performed by the agent. The goal of the agent is to establish the policy that maximizes the cumulative long term reward and uncover the hidden relationship between the system dynamics and its configurable parameters without the deep knowledge knowing how the system works. As a result, self-tuning capability based on the system's dynamics is needed by the adaptive parametric systems, and studies are needed to interpret how the reinforcement learning can be utilized to improve the performance of parametric systems in terms of processing time and the resources utilization rate, which are the aims of this work.

1.3 Objectives of Research

To implement adaptive self-reconfigurable parametric systems, there exists a need for a better understanding of how reinforcement learning algorithms can be utilized to improve the performance and hardware resources utilizations of the system. The objective of this research is to advance the knowledge of the efficacy for deploying reinforcement learning algorithms on tuning the performance of the parametric system

automatically and adaptively. Another objective is to utilize a similar approach to manage flow entries of SDN to minimize the communication overhead between controller and the OpenFlow enabled network switches, and further improve the utilization efficiency of TCAM capacity installed in the switch. To accomplish these objectives, this dissertation research was organized and advanced in three projects.

In the first project, the literature review of Software-Defined Networking (SDN) technologies was conducted to elaborate the potential of the current evolving network architecture, which represents a new approach to orchestrate networks based on the needs of the dynamic nature of current networking capabilities. The review covers the architectures, applications, corresponding protocols, and deployments associated with SDN.

In the second project, the Reinforcement Learning (RL) based scheme, namely RL-MRCONF, was proposed and developed to automatically tune the reconfigurable parameters of MapReduce framework. Specifically, two variations of RL algorithms were studied; one is traditional reinforcement learning algorithm and another one is deep reinforcement learning algorithm. In addition, the MapReduce simulator was also implemented to demonstrate the effectiveness of our proposed approach. The simulation results show that the RL-MRCONF is capable of reconfiguring MapReduce parameters automatically and dynamically in response to varying job types and underlying computing resources.

In the third project, a novel approach to enable machine learning capabilities to manage flow entries of SDN enabled networking environment in response to the partitioned and aggregated network traffic patterns was created by using reinforcement learning to reduce network reconfiguration overhead. It is significant to determine which flow entries (forwarding rules) should be stored in the flow table of the OpenFlow enabled switch when considering the limited capacity of Ternary Content-Addressable Memory (TCAM) installed in the switch. The emulation results demonstrate that our RL based approach is capable of reducing network reconfiguration overhead in the long-term and further improve the table-hit ratio of the SDN enabled network.

CHAPTER 2

LITERATURE REVIEW – EMPOWERING NETWORKING RESEARCH AND EXPERIMENTATION THROUGH SOFTWARE-DEFINED NETWORKING

2.1 Introduction

The trend for researching Future Internet is driven by the drawbacks of present day's Internet and the demand for more sophisticated applications and services. Future Internet focuses on developing novel network and architecture principles, allowing the applied technologies to go beyond the existing standards or models to resolve the issues identified in today's Internet. Recently, Software-Defined Networking (SDN) has emerged as the exciting technology challenges the current networking paradigm in terms of management and orchestration of present computer networks, which allows SDN to have potential to evolve not only how networks are implemented but can inspire novel applications that are limited by existing network structures.

In order to address the known issues of the current computer networking structures: (1) Difficulty in optimizations, (2) shortcomings in security, robustness, manageability, and scalability, and (3) increasing capital expenses; SDN was developed by delivering significant benefits to end users to accomplish the needs for managing the dynamics of computer networks as in [8]:

- Managing the networking devices in logically centralized manner;
- Eliminating the need to manually configure each device over the network;
- Utilizing the common programming environments and APIs to improve programmability;
- Improving the security and reliability of network services by using centralized network devices management;
- Lowering operational costs by improving efficiency for services provisioning.

The purpose of this chapter is to provide readers with fundamental information pertaining to SDN. What SDN is, and the associated applications utilizing SDN to empower the research of computer networks and experiments. In addition, the underlying OpenFlow protocol utilized by SDN, and some experiments on evaluating the benefits obtained by utilizing SDN, including the recent researches and associated literature papers, are also discussed and summarized in this chapter, with the goal of broadening

innovations and promoting new researches in evolving the capabilities of future computer networks.

2.2 Overview of SDN

SDN is a novel networking paradigm of computer networks proposed as the approach to advance networks with programmable capability. The fundamental concept of SDN emphasizes on simplifying and optimizing the network management operation by decoupling the data plane (the mechanism that forwards the data packets to the specified destination) from the control plane (the mechanism that makes decisions regarding where the packet is sent). The transparency of the programming interface that enables the development of applications to dynamically control and manage the connectivity between the participated network elements is the primary goal of SDN.

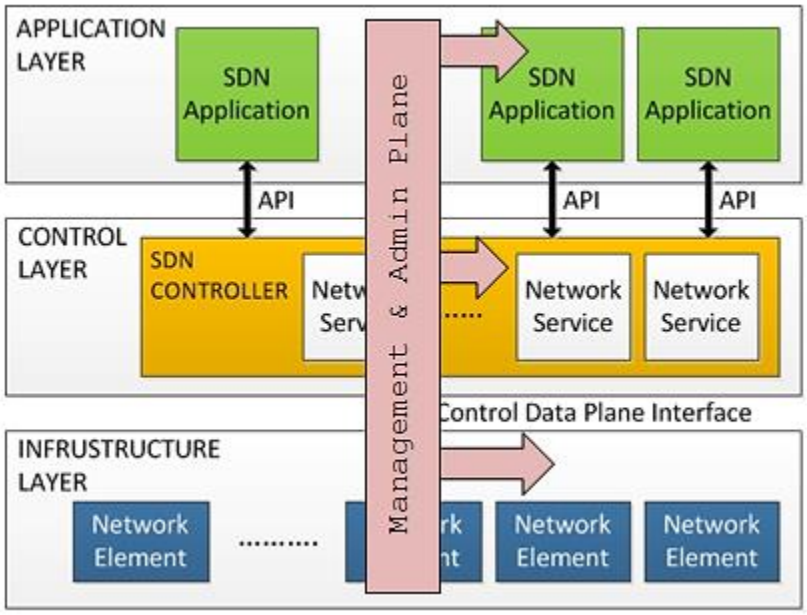


Figure 4. The overview of Software-Defined Networking Architecture

An overview of the SDN architecture is depicted in Figure 4. The layers of application, control, and infrastructure have been referred as the application, control, and data planes respectively [8]. The application plane consists of several SDN applications that utilize open APIs to interact with the SDN controller. Those APIs are referred as *Northbound Interface* (NBI) *Drivers* and it is used as the communication gateway to send

application requirements to the control plane, and receives network activities such as states and events from the control plane. In the control plane, the SDN controller is employed to disclose the statistics, instrumentation, and events from the underlying data plane network elements up to the application plane, and interprets the requests of the applications down to the data plane. Multiple network elements, as illustrated at the bottom of Figure 4, represent the data plane that utilize the *Control Data Plane Interface* (CDPI) to reveal their *SDN Datapaths* and capabilities, providing the SDN controller the capability to enforce low-level network control over those datapaths.

In addition, a *Management & Admin* plane is used to interact with all these three planes and is responsible for orchestrating the network elements, making proper assignment of SDN controller to the SDN datapaths, and managing network policies that define the controllability of the assigned SDN controller or SDN application. As the result, the SDN programmability is further achieved by utilizing the SDN controller that controls the communication between upper-level applications and low-level network elements. Due to the benefits provided by SDN as in openness, flexibility, functionality, and programmability, several research works proposed several network programming languages that are compatible with the OpenFlow [8] protocol. The idea of using high-level abstraction for developing network applications has been cultivated by those research works. Examples of such programming initiatives include *Frenetic* [130], [131], *Pyretic* [132], and *Nettle* [133]. Those languages are briefly discussed as follow:

- **Frenetic**: Frenetic is designed to mitigate the barrier between the low level design of OpenFlow interface and the orchestration of the high-level concept on the switching hardware, and this is achieved by supporting a domain-specific language for SDNs. Frenetic proposes a set of functional abstraction to enable modular program development and reduces the difficulties of the controller-to-network programming by using a "see-every-packet" abstraction feature. Frenetic operates at the abstraction on the packet-level and it is considered as a *functional reactive programming* (FRP) language.
- **Pyretic**: A Python-based programming language implemented as the extension of Frenetic that includes all the features of Frenetic. It emphasizes on the high-level abstraction of the network policy and refines the support of modular programming

by using “policies-as-abstraction-functions” feature to integrate multiple network policies as *parallel* and *sequential* compositions when utilizing policy composition operators.

- **Nettle**: Based on the principles of functional reactive programming, Nettle is a domain-specific programming language and has the ability to capture/monitor the communication patterns between SDN controller and OpenFlow switches discretely and continuously. It supports programming of the OpenFlow-enabled networks with high-level network abstraction concept with declarative manner.

OpenFlow protocol has realized the programmability of SDN that provides a flexible, dynamic, and standardized programmable interface allowing OpenFlow-enabled switches to be managed by the logically centralized SDN controller.

2.3 Network Function Virtualization

The ability to synthesize the multiple hardware and software networking resources as the logical software-based virtual network entity executing on top of commercial computing servers is defined as *Network Function Virtualization* (NFV) [124], which is achieved by decoupling network services in terms of network address translation (NAT), firewalling, intrusion detection, domain name service (DNS), caching, etc., from proprietary hardware devices for easy integration and management.

The NFV technology was initiated by *European Telecommunication Standard Institute* (ETSI) with the goal of reducing both capital costs (CAPEX) and operating costs (OPEX) in daily information technology (IT) business operations. NFV is capable of delivering network functions on various locations of the network to provide a fully virtualized infrastructure, without the need for physically installing, deploying or configuring the network devices. As the result, NFV provides several benefits based on [124], [126]:

- Conserves IT equipment expenditures and energy consumption by integrating network appliances;
- Dynamically optimizes network topology based on the real-time traffic utilization to further improve scalability and resiliency of the network;
- Delivers the customized network services on demand;

- Shortens IT orchestration cycles to improve operating performance and efficiency.

The concept of functional entities and the reference points within the functional domain are utilized by ETSI NFV group when defining the architecture of NFV, and there is no specific implementation for these entities. The detail NFV information is described in [125] and further discussed in [126]. Additionally, NFV technology is often mistakenly considered as synonym of SDN technologies due to the similarity of the characteristics in software controlled computer networking capabilities. However, both SDN and NFV are two different independent technologies but they are complementary to each other [124], [126]. The comparison between SDN and NFV with the aspect of what can be achieved for being utilized jointly are presented in Table 1.

Table 1. The Difference Between SDN and NFV with Combined Benefits.

	SDN	NFV	SDN + NFV
Characteristics	• Decouples the control and data planes [8] • Provides programmable networking and logically centralized management [8]	• Transfers the network functions from the proprietary devices into software-based implementation [124] • Decreases CAPEX, OPEX, and energy consumption [124]	• SDN provides programmable interface between each *Virtualized Network Function* (VNF), which can be managed by SDN controller or VNF orchestrator [126], [127]. • NFV provides software-based network functions that can be used by SDN to deliver services on the certain network locations as required [125].
Standardization Group	Open Networking Foundation (ONF)	ETSI NFV Group	
Equipment	Commercial servers and switches	Commercial servers and switches	Commercial servers and switches
Applications	Cloud computing and networking	Network traffic engineering	
Usage	Data centers	Network service providers	

Non-SDN based deployments can be utilized to achieve NFV functionalities, but in order to accomplish better network performance, compatibility, and manageability; join operations in the aspect of SDN control and data planes separation with NFV are preferred [124], [125], [127].

2.4 SDN Technologies

The core concept behind SDN is the separation of the control plane from its underlying data plane. This approach can be traced back to early 2000s [11], where researchers focus on developing a solution to address increasing network traffic volumes, and to improve network reliability, manageability, and performance. Overall, the realization of this core concept of SDN has been enabled by the following technologies:

ForCES: *Forwarding and Control Element Separation* (ForCES) [16] [17] is an architectural framework used to define the communication protocol between control plane and data plane, which have been represented as two major elements: *Control Element* (CE) and *Forwarding Element* (FE). This design can improve flexibility of network elements and network scalability by adding additional CEs and FEs to the existing network topology without the need to perform full-scale updates to the infrastructure. The architecture of ForCES is presented in Figure 5.

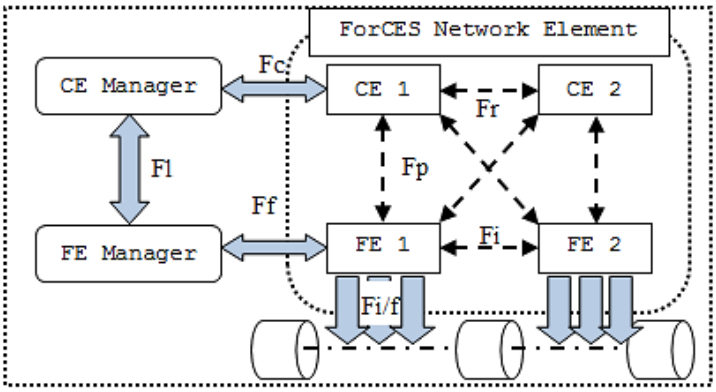


Figure 5. The architecturial diagram of ForCES

In the domain of ForCES, multiple instances of CE and FE are permitted, and each FE is equipped with one or more physical media interfaces as the communication channel to the external world. In addition, the network element components are interconnected and one element can forward the network packet to any other elements, for which the load balancing, resource sharing, distributed control, and other purposes, can be further achieved by using this redundant CEs and FEs mechanism. Distributed control is utilized when multiple CEs are concurrently active but only certain CEs are capable of processing specific requests. The *Logical Function Block* (LFB), defined by

multiple FEs, is the building block controlled by CEs via the ForCES communication protocol. This allows CEs to control and manage the FEs regarding how the packets can be processed by them.

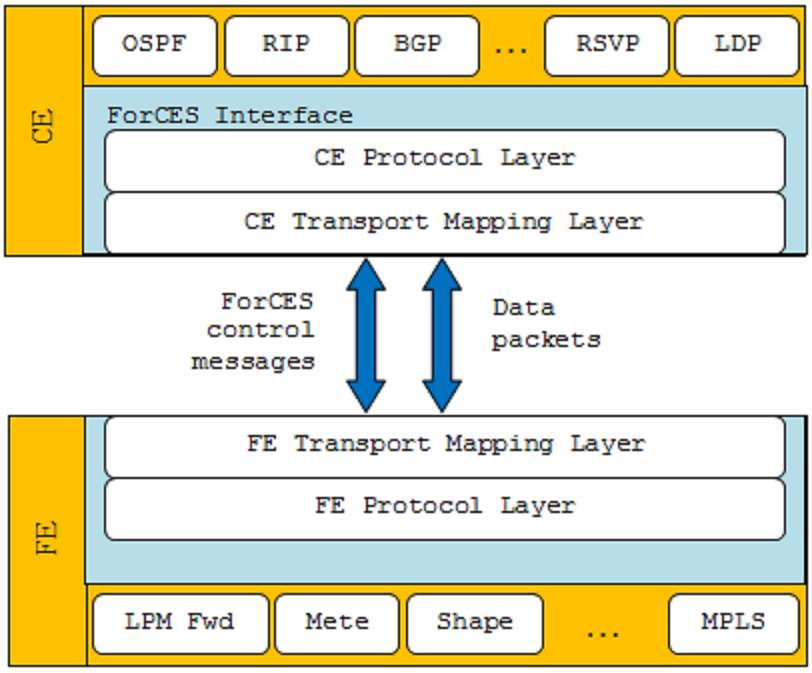


Figure 6. The ForCES interface with CE and FE functions

The functions of CE and FE in ForCES, as depicted in Figure 6, consist of multiple layer 3 and layer 2 protocols, which are employed by CE and FE respectively. ForCES interface resides on both CE and FE as it is used to carry both control messages and network data packets. Moreover, two protocol layers in both CE and FE are connected by *Transport Mapping Layer* (TML), and the implementation of TMLs varies depending on the capabilities of underlying transport and media. However, as long as both endpoints support the identical TML, the interoperability between TMLs is guaranteed. Moreover, the *Protocol Layer* (PL) is used as the bridge to carry various events and requests between CE and FE, such as event subscription, FE activation, etc. The information exchange standardization is defined by ForCES protocol between CE(s) and FE(s) only.

OpenFlow: OpenFlow [8], [14], a communication protocol, is defined and implemented in the *Control Data Plane Interface* (CDPI) of the SDN structure.

OpenFlow enables the direct access to the underlying forwarding plane of network element as in routers or switches, and the route of network data packets can be constructed programmatically to travel through those network elements. Currently, the core concept of SDN has been complied in the OpenFlow protocol involving two categories: one is the wired protocol defined by an OpenFlow switch (OF-SWITCH) with most recent version in 1.5.1 [8]; another one is the remote management protocol defined by the *OpenFlow configuration and management protocol* (OF-CONFIG) with most recent version in 1.2 [8], both specifications are maintained and published by the *Open Networking Foundation* (ONF).

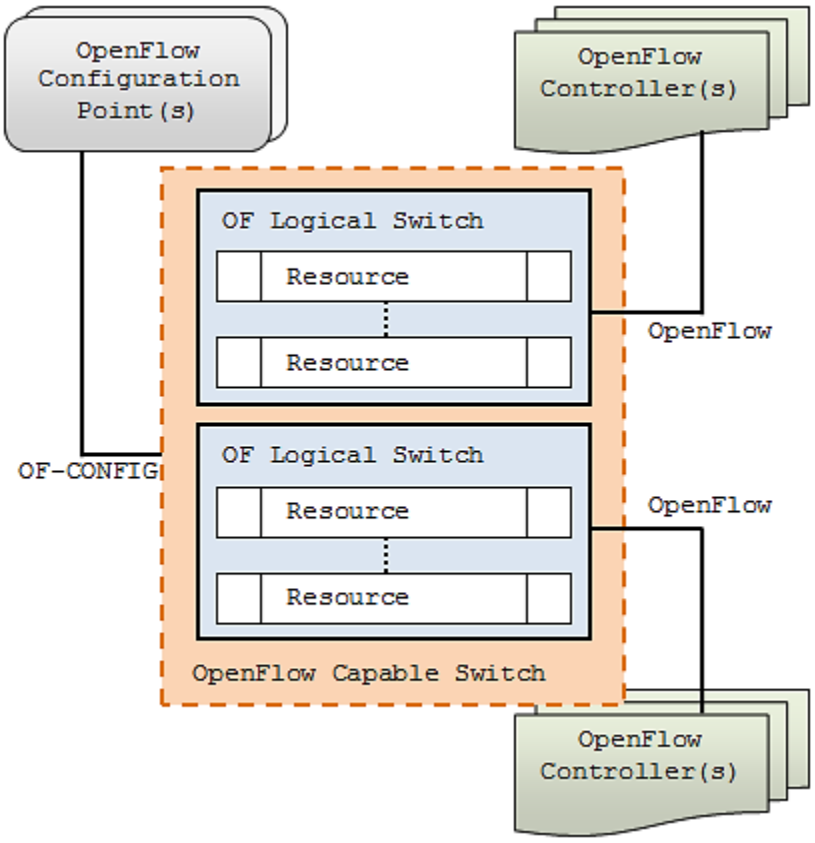


Figure 7. The relationship between OF-CONFIG components

As a companion protocol to OpenFlow, the OF-CONFIG is developed to configure OpenFlow switches remotely. The abstraction view of an OpenFlow switch in OF-CONFIG is defined by an *OpenFlow Logical Switch* (OFLS) to allow OpenFlow controller to communicate and control the OFLS via the OpenFlow protocol. An

OpenFlow Capable Switch (OFCS) is developed to integrate one or more OpenFlow switches as an organizational operating context. The dynamic association of the OpenFlow related resources with a specific OFLS can be achieved by partitioning hosted OpenFlow Logical switches within the OFCS. The relationship between OF-CONFIG protocol components is illustrated in Figure 7.

An *OpenFlow Configuration Point* (OFCP) is defined as the service to transmit the messages of OF-CONFIG to an OFCS, and the service can be either implemented programmatically using software, or can be supported by a traditional network framework. The communication between the OFCPs and OFCSs is defined by OF-CONFIG, and the communication protocol between *OpenFlow Controllers* (OFC) and OFLSs is defined by OpenFlow.

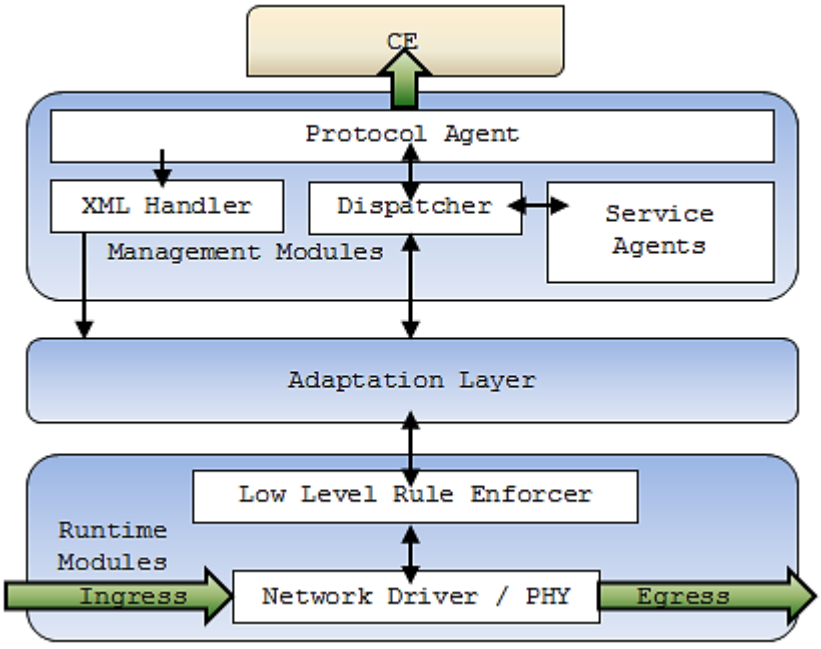


Figure 8. The ProGFE structure diagram

ProGFE: Based on the LFBs defined by ForCES, *Programming Generic Forwarding Element* (ProGFE) [48] provides an elastic and reconfigurable network system, and each LFB controls how ProGFE functions. An XML-based API is utilized to configure ProGFE dynamically in order to adapt the dynamic nature of network in providing various network functions, protocols, and topologies. One or more LFBs are

connected using a directed graph and can be used to define processing operation of ProGFE. Additionally, the path between two LFBs is used to represent frame type-based processing of the system.

Two groups, namely management and run-time modules as depicted in Figure 8, can be used to classify the modules of ProGFE. The communication to CE is controlled by management module to manage the LFB information, and the internal data structure is used to define how those run-time module process the frame. In general, ProGFE has three main modules: *Protocol Agent*, *XML Handler*, and *Low Level Rule Enforcer* (LLRE). The protocol agent uses general protocols, such as HTTP, Wired, and more, to communicate with the CE. The XML handler is responsible for paring/translating the LFB XML information and passing the processed information to relevant data structure. Furthermore, the LLRE is responsible executing the meta-program capable of processing pre-defined LFB types, and the meta-program is used to describe various of LFBs and their associated parameters defined by the CE.

2.5 Programmable Data Planes

The renovation of network devices as in switches, routers, middleboxes, and commodity computing server technologies have driven the researchers not only focusing on achieving high packet processing efficiency but putting more attention on sophisticated network services. The obstacles have been encountered in present day's evolving Internet by current closed and proprietary design of the network equipment. To mitigate these barriers, there are multiple efforts have been conducted to address the issues and demonstrate the improved flexibility and extensibility of current networks:

Open vSwitch: Open vSwitch (OVS) [21], [22], [23], is an open source and software-based multilayer switch that executes on top of virtualized environment or hypervisor. OVS enables the capability of large scale network deployment automation through software implementation while preserving the compatibility of existing protocols and standard management interfaces. With the advantages of easy adaption to increasing employment of virtualization technologies, OVS is developed for the deployments of multi-server virtualization environments [21].

Recently, several OVS developers have been worked with the development team of OpenStack to develop a flexible virtual network service framework namely "Quantum" project [21]. OpenStack has gained more attention in the industry as it provides an open source cloud infrastructure management platform that supports computing resources orchestration, as in Infrastructure as a Service (IaaS), to provide cloud computing services on demand. Based on the current limitations of the OpenStack network architecture in terms of complex agents, flat networking and VLAN-based networking, those drawbacks have limited the number of available VLANs to be 4096 and the number of MACs that can be stored in the switch table is also limited. In order to address these issue, the objective of the Quantum project is to provide a simple, clean, and flexible API for service providers or users to manage their own network topologies. In addition, OVS utilizes the concept of "ports" for the instantiation and management of the virtual networks, and a port can be attached to any network interfaces, regardless of the type (virtual or physical).

Click: A software-based modular router, Click [134], [135] provides the framework for implementing configurable routers that enables users to develop packet processing functions as the *configuration* via a set of interconnected *elements*. Each element in Click is used to represent a simple processing computation instead of a complex one. The directed graph is utilized to describe the connections between each connected elements and the route for which the packet should traverse. The ports are used in each element to send or receive data and the deployment of new network services or functions is simply by adding or rearranging the elements. The open and simple design of Click has gained more popularity in network research as it provides an array of reusable open source modules.

RouteBricks: RouteBricks [136] is another software-based router architecture designed to utilize the advantage of router functional parallelization by employing multiple server machines or a single server with multiple cores. The performance of the router can be further improved by using parallelization in terms of adding more servers to increase computing resources. The goal of RouteBricks is to make networks easier to build, deploy, and program by utilizing commodity server machines, and it allows developers to rapidly build and program/reprogram of networks using general hardware

and software resources. Besides, RouteBricks has been presented with fast network processing by using software executing on clusters of computing resources, and further reduces the operating costs with improved programmability and flexibility when compared with proprietary network devices.

ClickOS: In order to adapt the increasing utilization of network function utilization in current networks, ClickOS [137] is implemented to run Click on top of the Xen-based virtual machine (MiniOS). With the purpose of dynamic instantiation, update, and orchestration of network functionality, ClickOS is designed to utilize the concept of middleboxes to improve the programmability of data plane. It provides the advantages of rapid instantiation (about 30 milliseconds), light weighted (6MB when running), good isolation (running on top of virtual machine), high speed (10 Gb/s pipe network), and improved flexibility.

2.6 SDN Infrastructure

The SDN architecture can be further classified into three major components: Application Plane, Control Plane, and Data Plane. The core concept of SDN is defined by the collaboration of Control Plane and Data Plane which includes the implementation of SDN and the interaction between the controller and the underlying network elements. In this section we will discuss a variety of recent development of SDN controllers, including the Southbound APIs that are responsible for the communication between Control and Data Planes, the Northbound APIs that are used as the bridge between SDN applications and the controller, the flow table mechanism that performs packet forwarding, and different type of SDN switches employed in such architecture.

The SDN Controller: As the "brain" of SDN, the SDN controller is implemented using a software system or a collection of systems incorporated to support the programmability of the network. It decouples the control plane from the data plane and utilizes the logically centralized control plane to manage the dynamics of the network. SDN controller employs the high-level data model to manage network state as the relationships between managed resources, policies, and other network services. In addition, the common API (northbound interface) is supported in the form of *Representational State Transfer* (REST) or Java to disclose the services provided by the

controller to the SDN application and manages the interactions between the application and the controller.

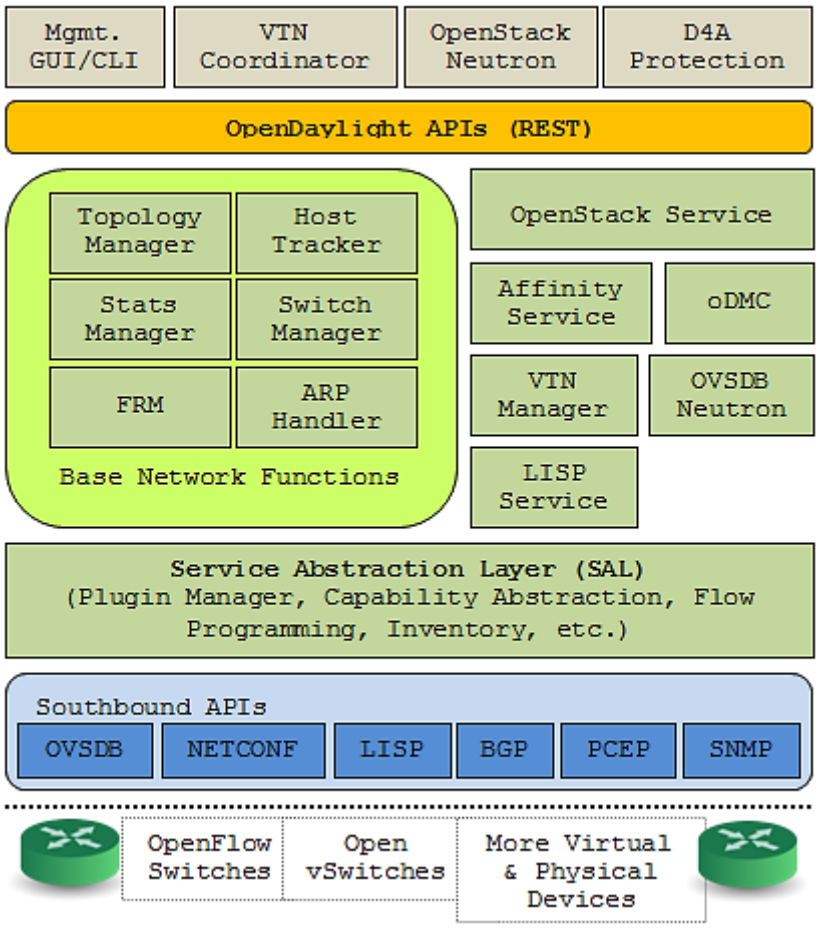


Figure 9. The block diagram of OpenDaylight framework architecture

As illustrated in Figure 9, the OpenDaylight [41] framework is a collaborative project developed by Linux Foundation. Based on the openness characteristic of the project, the framework integrates all the open resources, open source codes, and open projects from open community developers. The top layer of the framework, consists of network logic and business applications, is responsible for managing and monitoring network behavior; a collection of common APIs to the application layer (the northbound APIs) is the major component of the middle layer, which is also in charge of managing the underlying physical network elements within the network with one or more protocols

(southbound APIs); all the bottom layer devices, both physical and virtual, are used to connect all the endpoints in the network.

The OpenFlow network is managed using a set of common functionalities exposed by the controller, and the upper level applications provide various features based on the user requirements. The core component of the controller is responsible for processing I/O from switches and the OpenFlow messages is translated in the form of events, providing an event-driven application as in Floodlight [39] SDN controller. The Floodlight controller allows built-in modules to communicate with their implementation of OpenFlow services, and the OpenFlow protocol is used by the controller to interact with the underlying forwarding devices. There are several types of SDN controller platforms available, as presented in Table 2, can be used as the panel for the public to have a peak into the realm of SDN.

Table 2. The Open Source SDN Controllers.

Controller	Summary
NOX [36]	The first-generation controller implemented in Python/C++ that is compatible with OpenFlow 1.0 (0x01)
POX [37]	The Python based general open source controller that supports OpenFlow 1.0 (0x01)
Trema [38]	The Ruby/C based framework that provides basic libraries and functional component as the interface to interact with OpenFlow 1.0 (0x01) for developing OpenFlow controllers.
Floodlight [39]	The Java implemented OpenFlow controller based on the Beacon controller developed by Stanford University
Ryu [40]	A component-based OpenFlow controller framework written in Python, integrated with OpenStack by using OpenStack Quantum plug-in and currently supports OpenFlow protocol 1.4 (0x05)

The Southbound APIs: The southbound APIs, as in SDN architecture, are used to control the communication between the SDN controller and the forwarding devices, orchestrate the efficient control over the network, and enable the controller to adapt the real-time dynamic needs of the network.

The OpenFlow protocol implemented by ONF is the first and most well-known southbound interface that supports network management over the communication of control-data planes. The detailed information regarding OpenFlow protocol has been discussed in the previous section. In addition to OpenFlow, the LISP (*Locator ID*

Separation Protocol) is also advocated by ONF as one of the southbound API that enables flow mapping, and MPLS (*Multiprotocol Label Switching*) protocol is also being studied to operate in an SDN domain.

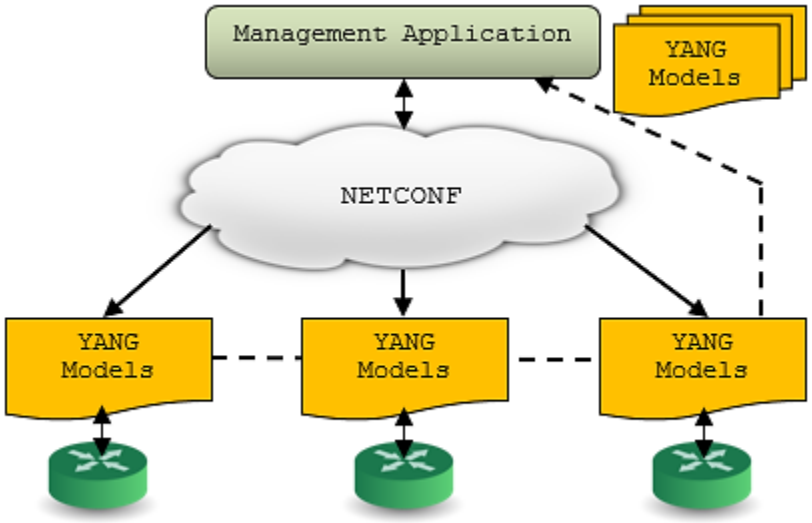


Figure 10. The relationship between NETCONF and YANG

NETCONF [13], [29], [30], standardized by IETF network management protocol, is another southbound API that provides functionalities to modify and delete the configuration of the network devices over the network, it uses XML-based encoding with SSH-based encryption, and performs operations over Remote Procedure Call (RPC). Additionally, the robust transaction integrity defined by ACID (*Atomicity, Consistency, Independency, and Durability*) properties are also supported by NETCONF along with its paired data model *YANG* [13], [34]. YANG model is used to indicate which data of the network devices can be modified. Figure 10 represents the relationship between NETCONF and YANG, and YANG is visible externally to be utilized in defining the structure, syntax, and the semantics of the configuration data.

When the communication between NETCONF enabled device and manager is established, the `<hello>` message is exchanged over the communication channel, presents the capabilities of each connected device, and the type of YANG data model compatible with the device. The `<hello>` message enables the NETCONF manager to know which operations can be performed on the device. The single `<edit-config>` ACID enforced transaction is used by the NETCONF manager to deploy a set of

configuration changes, or an entirely new configuration to the device. To activate a change and test it, the `Confirm-commit` command is used by the manager and it also provides the flexibility to ignore the configuration if it is not working properly and the connection to the device can be dropped. There are some examples of basic operations supported by NETCONF include *<get>*, *<get-config>*, *<edit-config>*, *<copy-config>*, *<commit>*, *<discard-changes>*, *<delete-config>*, etc.

The tree structure is utilized by the YANG data modeling language in the form of modules and sub-modules, which can import data from other external modules or sub-modules. The augmentation is supported in this tree hierarchy for adding data nodes to the hierarchy defined by another module. Moreover, a type mechanism is also utilized in defining a set of built-in types in YANG. Currently, the OF-CONFIG of the OpenFlow protocol integrates both NETCONF and YANG data model to remotely control the configuration of all the connected OpenFlow switches [8] [13] in providing different perspective to manage the network resources and operations.

The Northbound APIs: In the SDN structure, the *Northbound Interface* (NBI), as implemented between SDN applications and SDN controller, is also an open and vendor independent interface. The NBI provides an abstracted view in describing the network states, requirements, and behaviors. The driver-agent pair model is used when implementing the interface to provide an easy information extraction process from the network devices to the external applications.

Despite the fact that northbound APIs has not been standardized but they are developed as on-demand basis for targeted applications. As the starting point, both Floodlight [39] and OpenDaylight [41] are good instances for demonstrating the cases for employing REST APIs as the NBI. The *Northbound Interfaces Working Group* has been constituted at ONF to develop a single standardized northbound interface, but this process remains in challenging due to the variety of all the available interfaces in this area.

Flow table: The flow entries (forwarding rules) of SDN is administered using the flow table for packet matching process. This matching process is performed by checking the fields in each flow entry (*match fields*, *priority*, *counters*, *instructions*, *timeouts*, and *cookie*). Intuitively, the headers from layer 2 and layer 3 protocols of the TCP/IP suite are

contained in the match field. The precedence of the matching process of the flow entry is defined by priority field; the counters are updated whenever the flow entry is matched with the packets; the action set or pipeline processing is controlled by the instructions field; the timeout value represents the maximum time before the entry is expired; the flow statistics are filtered by SDN controller using cookie field, which is ignored when packets are processed.

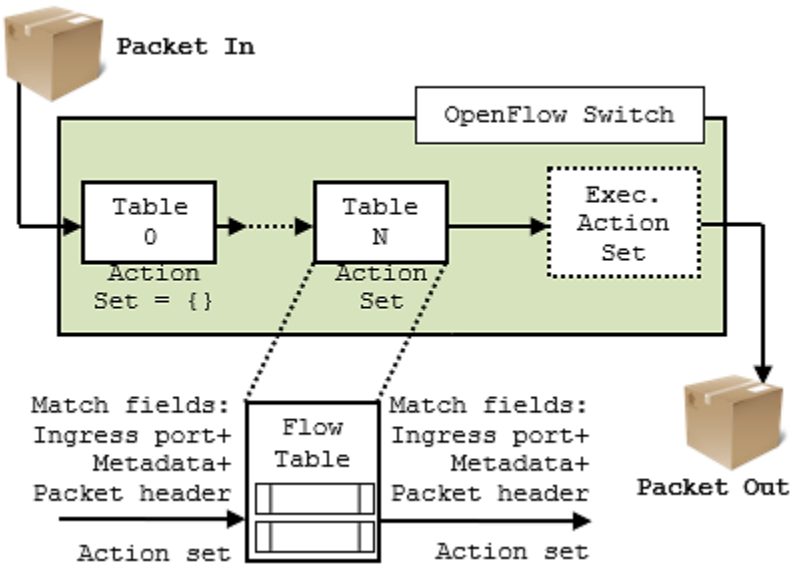


Figure 11. The packet and flow tables pipeline process

The mechanism of the pipeline process defines the interaction between packets and multiple flow tables, as illustrated in Figure 11. Whenever a packet arrives at an OpenFlow switch, the packet is further processed against each flow table by using following procedures:

- Searching the matching flow entry with highest priority;
- Performing instructions by updating the packet and the match fields, along with the action set and the metadata (a value of register used to convey information from one table to another);
- Passing the match data and action set to the next table.

Sequential number is labeled on each flow table starting at 0 to indicate the forward-only pipeline process due to a flow entry can only forward the packet to the flow

table with larger number than its own flow table. The pipeline process stops when the packet cannot be forwarded to another flow table.

For the case when a packet cannot be matched with a flow entry within the flow table, a *table-miss* event will occur, and the corresponding table miss action may vary depends on the configuration of the flow table. In order to handle this scenario, a table-miss flow entry is utilized to control how an unmatched packet is processed such as dropping the packets, passing the packets to another flow table, or forwarding them to the controller for further process. Both match and priority fields are used to represent a table-miss flow entry by ignoring all the match fields and using 0 in the priority field to indicate the lowest priority. By default, the table-miss flow entry does not exist in the flow table and it may be added or removed by the controller, or it can be managed by the expiry mechanism.

To remove flow entries from the flow table, there are three approaches: the request from controller, flow entry expiry mechanism, or eviction mechanism supported by switch. Two attributes `idle_timeout` and `hard_timeout` are utilized by the entry expiry mechanism and it is enforced and executed by the switch. A non-zero `idle_timeout` field is used to define the removal of the flow entry when the entry is not being matched for the given time frame, whereas a non-zero `hard_timeout` is used to indicate the removal of the flow entry regardless of the value in the match field of the flow.

SDN switch: In the SDN domain, the OpenFlow enabled switches can be discussed as follow:

OF-SWITCH: As depicted in Figure 12, the OF-SWITCH is an OpenFlow switch consists of one or more *flow tables*, one *group table*, and an encrypted OpenFlow *channel* to the external controller. The OpenFlow protocol enables the controller to efficiently manage the flow tables of the switch by inserting, modifying, and removing the set of flow entries based on the network packets and traffic conditions.

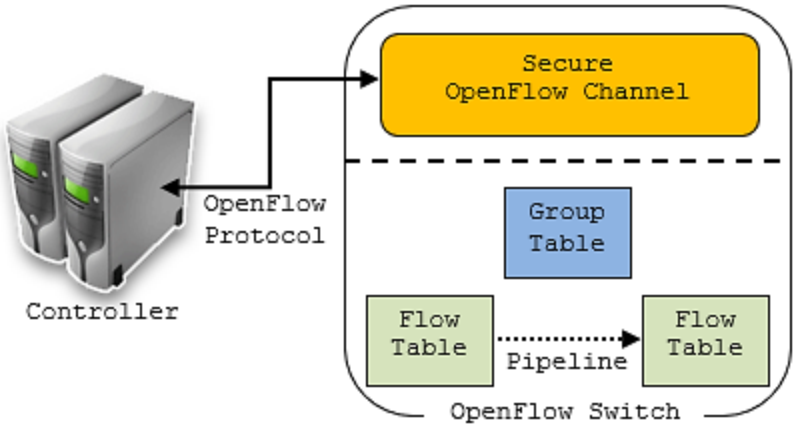


Figure 12. The components of the OpenFlow switch

Whenever a packet arrives at the switch, the pipeline packet matching process is initiated from the first flow table, and flow entries within each flow table are matched based on the priority order of the entry in the packet being processed. The corresponding instructions are performed whenever the packet matches an entry of the flow table. According to the configuration of the table-miss flow entry of the switch, the unmatched packet maybe dropped, forwarded to the controller, or sent to the next flow table for further processes.

A set of instructions that contains actions such as packet forwarding, packet modification, and group table processing, is used to associate with each flow table. Similar to flow entry table, the group table also has group entries, and a list of *action buckets* with specific semantics contained in each group entry depending on the group type.

Another essential concept of OpenFlow protocol is the employment of the *OpenFlow Ports*. These ports are used as the network interfaces for passing packets among OpenFlow processes and the rest of the network. The logical connection between the OpenFlow switches is also using port. In addition, the communication between OpenFlow switch and controller is managed by the OpenFlow Channel in terms of managing, configuring, receiving events from, and sending packets to the connected switch(s).

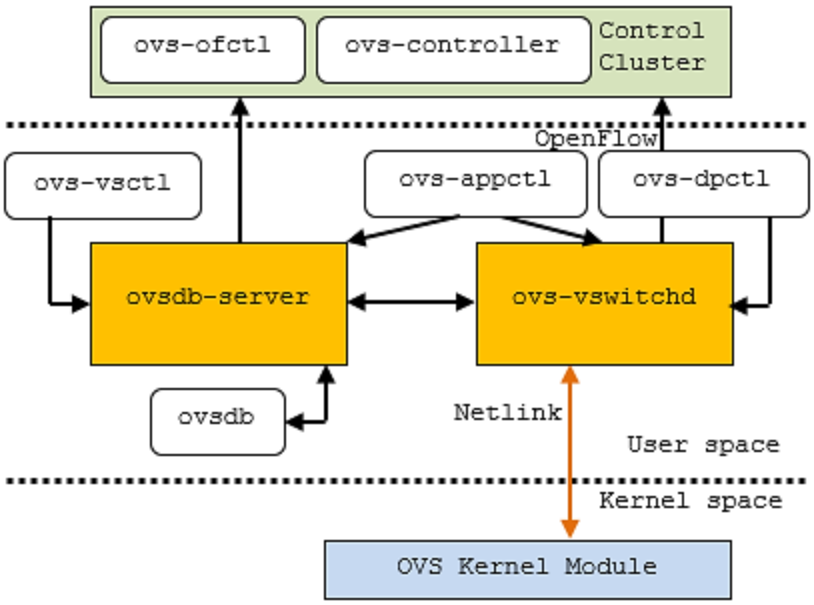


Figure 13. The architectural blocks of Open vSwitch and main components

Open vSwitch: Open vSwitch (OVS) is the virtualized software-based switch platform compatible with OpenFlow protocol. As presented in Figure 13, the OVS constitutes a *Control Cluster*, an *ovsdb-server*, an *ovs-vswitchd*, and the OVS *Kernel Module*. The definitions of switch level configurations, such as network bridges, interfaces, tunnels, addresses of the OVSDB, and the OpenFlow controller, are stored in the configuration database (ovsdb-server). The root table of the configuration database, namely "*Open_vSwitch*", is used to store the configuration of one Open vSwitch daemon (ovs-vswitchd), and other tables are utilized to store related records only when they can be accessed directly or indirectly from the "Open_vSwitch" table. The unreachable records from the root table will be automatically purged. To query and configure OVS daemon, the high-level utility program "*ovs-vsctl*" is employed to apply changes to the database, which then will be deployed to the connected OVS daemon.

The core component and actual implementation of OVS is the Open vSwitch daemon, and the utility "*ovs-appctl*" is utilized to configure the daemon and modify the switch behavior at run-time. During the booting process of the OVS, the OVS daemon fetches the configuration from the database and orchestrates the OVS datapaths and executes switching operations across each bridge. Additionally, the configuration is synched automatically between database and OVS daemon whenever the configuration is

updated. Multiple independent datapaths (bridges) are supported by OVS daemon using "*ovs-dpctl*" utility, which is used to communicate with the kernel module to perform CRUD (Create, Read, Update, and Delete) operations on OVS datapaths, and only one network device can be deployed with a newly created datapath. The remote controller utility "*ovs-ofctl*" enables OVS to manage multiple Open vSwitches with OpenFlow protocol.

Table 3. The OpenFlow Compatiable Open Source SDN Switches.

SDN Switch	Summary
OpenFlow Switch [8]	The C/C++ based software switch compatible with OpenFlow 1.4 with pipelined flow table processing, and support flow monitoring, extensible match, and optical port.
Open vSwitch [21]	The C/Python based open source switch for virtual machine environment, the current version supports communication monitoring, 802.1 ag link and BFD monitoring, OpenFlow protocol, LACP protocol, and etc.
NetFPGA [42]	The open source OpenFlow compatible hardware platform that supports hardware enhanced Ethernet switching capabilities.
Indigo [47]	A C based open source software switch compatible with OpenFlow protocol that operates over physical and hypervisor switches.

The basic switching and tunneling operations are handled by OVS kernel module for fast and simple packets processing. Whenever a packet arrives, the lookup and matching operations are performed, and the associated actions are executed with updated counters if a match is found; if there is no match, the packet is sent to the user space for further processing. The detailed supporting information can be found in [21]. The variety of OpenFlow compatible SDN switches are presented in Table 3.

2.7 SDN Deployment Environments

To construct a system, design, deployment, and verification are considered as the crucial steps involved in this process. In this section, the deployment of SDN/OpenFlow technology in the variety of environments such as in datacenters, testbeds, and high performance clusters are discussed.

Software emulation-based deployment: A software-based network emulator, Mininet [45], [46], is developed to emulate a collection of end-hosts, network devices (switches and routers), and links over a single Linux kernel. Mininet provides the capability to have entire OpenFlow network to be emulated on a single computer with

lightweight process-based virtualization. It simplifies the initial network deployment, and testing by using virtual Ethernet as the link between switches and endpoints. Mininet enables the experimental network orchestration to be first initiated and tested on the emulation environment before being physically deployed.

Hardware emulation-based deployment: NetFPGA [42], [44] is considered as an open source hardware and software platform utilized by academic researchers and industry experts for easy prototyping the networking devices. To achieve the configurability of an integrated circuit (IC), NetFPGA uses a *hardware description language* (HDL) to perform the configuration of the device. The programmable logic components (logic blocks) contained in FPGA enable developers to perform complex integrated functions and processes on the IC. In addition, as a hardware-based networking device platform, NetFPGA provides the ability to process network packets at a line-rate, which is considered as the compensation capability lacked in the software-based approaches. Currently, NetFPGA has two types of platforms: One is HetFPGA 1G (4 × 1G) that provides one Xilinx Vertex-II Pro 50 FPGA logic unit with four 10-Gigabit Ethernet interfaces. Another one is NetFPGA 10G (4 × 10G) that provides faster resources with one powerful Xilinx Virtex-5 FPGA logic unit and four 10-Gigabit Ethernet interfaces. The more detailed information of NetFPGA can be found in [42].

With the fast line-rate packet processing supported by NetFPGA, the OpenFlow protocol has been implemented on the 1G platform proposed by [44] that employs the entire NetFPGA SRAM (*Static Random Access Memory*) for storing flow entries, and the output queues are stored in DRAM (*Dynamic Random Access Memory*) for fast data retrieval. This approach allows more than 64,000 exact flow entries and more than 32 wildcard rules to be created in support of typical network demands. Another NetFPGA 1G approach [43] proposes the OpenFlow-enabled switch can be used in more network applications than traditional packet switching by utilizing only half of the SRAM capacity to handle up to 100,000 wildcard rules, with remaining other half to be utilized for storing output queues to mitigate the loss of performance.

Testbed-based deployment: The SDN technology has been deployed in applied network engineering research fields with the objective for orchestrating and developing fully functional OpenFlow compatible SDN testbeds. The testbeds constitute with both

virtual (Open vSwitch, ofsoftswitch13, etc.) and physical (OpenFlow-enabled hardware devices from variety of vendors, physical and virtual hosts, slices, etc.) SDN/OpenFlow-enabled networking elements. The individual slices [50] can be differentiated from each other by utilizing the networking hypervisor such as FlowVisor [75]. The experiments conducted on the testbed have the capability to generate and route traffic, which expands the horizon for more innovative researches. In addition, the researcher could deploy more realistic network traffic imported from real world scenario in the testbed environment. Moreover, there are several ongoing research efforts that focus on supporting the small scale of SDN testbeds such as in [114] for the Raspberry-pi based SDN testbed, which has been proposed to use Open vSwitch as the forwarding engine. A few examples of large-scale SDN testbeds are presented in Table 4.

Table 4. The List of Large Scale SDN-based Testbeds.

Testbed	Funding Agency	Geographical Spans	Example of Experiments
GENI [51]	NSF (National Science Foundation)	USA	Novel Routing [49], MobilityFirst Internet Architecture [106]
COTN [52]	NSF, GENI	USA-California	SDN/OpenFlow-based experiments [52]
OF@TEIN [53]	NIA (National Information Society) of Korea	Korea, South East Asia Country	L2 Generic Routing Encapsulation tunneling of Narinet/NF/OVS Capsulators for the linking of the OpenFlow-enabled switches [53]
TWAREN [54]	NCHC (National Centre for High performance Computing) of Taiwan	Taiwan	Topology discovery, virtual network integration, and multimedia streaming [56]
OCEAN [57]	NSF, DARPA, Industry	USA-UIUC	Verifying network-wide invariants in real time [110]
RISE [64]	NICT	Japan	MPLS-based tunneling (EoMPLS) over Wide area OpenFlow Networks (OFNs) [63], [64]
FEDERICA [70], [71]	European project of the 7th Framework	Europe	Optimal Resource Allocation for Disaster Recovery [113]
OFELIA [74]	European Commission's FP7 ICT Work Programme	Europe and Brazil	Software Transactional Networking [77]
Community Network Testbed [78]	CONFINE2 European project	Europe	Long-term running services (crowdsourcing) [78]

Commercial-based deployment: The realization of large-scale network research infrastructures [49], [54], [63], [64] made the integration of existing network with OpenFlow-enabled network possible. Specifically, a hybrid network architecture that incorporates the characteristics of SDN/OpenFlow with existing network. The commercial based deployments of the SDN technology is discussed in this subsection.

Google's SDN Solution for WAN: B4, a private commercial level deployment of SDN *Wide Area Network* (WAN) from Google, is one of Google's two backbone networks. Initially the B4 was implemented to serve as a low cost backend backbone, especially for transmitting high volume copy traffic. As discussed in [59], the detailed design of B4 involves a WAN connecting to Google's datacenters, and the capabilities of SDN/OpenFlow can provide a flexible administration of the network, improved network utilization, and cost effective datacenter-to-datacenter network [58], [59]. Google's WAN is structured as two backbones based on characteristics of traffic: I-scale and G-scale. With integrated SDN capabilities, the user traffic is carried by I-scale (an Internet facing network) while the traffic of datacenter-to-datacenter is handled by G-scale (an internal network) [58], [59]. The OpenFlow is utilized as the communication protocol between the controller and associated switches to ensure the reliability and scalability of the backbone [58], [59]. Furthermore, the real-time network utilization metrics and topology data are collected by the centralized Traffic Engineering (TE) service for calculating the efficient path assignments for the traffic, and programming the computed paths into the underlying switches [58], [59]. Those SDN supported capabilities fulfill the network managerial needs for deadline scheduler features, which is utilized in large file data transfer [58], [59]. More detailed design and implementation of Google's B4 can be referenced in [59].

Hadoop-OFE: With the exponential growth of data and the high demand for the analysis of large dataset, the Hadoop platform has been commonly utilized to process data in a timely and efficient manner. Recently, OpenFlow-enabled Hadoop cluster has become the popular research area in the *High Performance Computing* (HPC) community. OpenFlow is utilized to reduce limitations of Hadoop, and further improve the performance. For instance, during shuffling and sorting phases of MapReduce process, the switches in the Hadoop cluster suffer from saturation, specifically the

"hotspots" scenario resulting insufficient bandwidth offered to the computing resources. To address this issue, the authors of [68], [69] proposed an OpenFlow-based solution to provide dynamic network topology adjustment for local and wide area Hadoop clusters in order to improve performance by offering additional bandwidth to the computing resources as needed.

2.8 Standardizing SDN

Despite SDN promises the programmable network, the lack of standardized interfaces between the SDN controller and the forwarding devices is currently the major challenge of SDN. To address this, IETF 86, the first face-to-face meeting was held to focus on the dynamic management of the routing information and the I2RS WG (*Interface to Routing System Working Group*) was formed to fulfill the requirements. Additionally, the existing technologies are required to work with the logically centralized SDN management model to support additional capabilities. For instance, the existing standard PCE (*Path Computation Element*) is utilized to search for the optimal LSPs (*Label Switched Paths*) in IP/MPLS (*Multi-Protocol Label Switching Protocol*). However, due to the state of LSP cannot be managed by the controller, the stateful-PCE is proposed to mitigate this disadvantage.

To provide network topology transparency in SDN, ALTO (*Application-Layer Traffic Optimization*) is employed to collect the topology data and manage devices with new protocol extension. A newly implemented interface, BGP-LS (*BGP link state*) is developed for publishing the link-state and TE information by the network devices to the SDN controller.

Moreover, the *Internet Research Task Force* (IRTF) [35] is also making efforts on the field of SDN to explore more innovations to build future networking, which is advocated by the *Software-Defined Networking Research Group* (SDNRG) of IRTF. In the context of the management protocol, NETCONF (Network Configuration Protocol) was proposed by the IETF *Network Configuration* working group in 2006 [29], and the most recent proposed standard RFC 6241 [30] was published in June 2011.

The ONF [8] has been dedicated efforts on promoting standard architecture of SDN and the associated white paper was published in April 2012 on defining the demand

for the new network architecture, and the formal SDN definition. Correspondingly, the Internet-Draft [33] was published by the IETF Network Working Group in January 2014 in promoting a functional taxonomy of the approaches that can be utilized in the SDN domain. The efforts have been coordinated among those groups to convert existing standards with newly proposed standards to accommodate the dynamic and evolving nature of networks.

2.9 SDN Applications and Experiments

According to MIT Tech Review, Software-Defined Networking has been considered as one of the top ten evolving technologies of the year 2009 [85]. As the world is experiencing the benefits provided by the applications of this technology, the most promising SDN applications, variety of experiments, and research efforts are discussed in this section.

Cloud computing: In this IT-driven era, cloud computing has been widely adopted due to the rapid infiltration of IT in our daily life and the high demand for computing such as utility computing, reliable data storage, on-demand computing services, flexible infrastructure, and more. In addition to the supported capabilities of cloud computing, the privacy and security of the stored data are the major components that need to be addressed in the cloud computing infrastructure. It is crucial for the cloud services providers to take these two aspects into consideration, especially when the *Transborder data flow*, the governance of geographical border data flow, is involved in the scenario [81]. To properly address this concern, SDN/OpenFlow-enabled network is used for managing sovereignty and border control of the data based on its capability in controlling the data flow path [81]. According to the report of IDC (*International Data Corporation*), the SDN technology is growing rapidly via deployment in the datacenters, and the market is about to increase $3.7 billion by 2016 [82].

The cloud computing datacenter is designed to provide scalable and reliable computing resources for cloud based services. Although the datacenter can provide promising benefits to support cloud based services, the hunger for power of datacenter remains a major issue that needs to be improved. According to [96], the cost of electrical energy consumption is about 70% of its total power budget. In USA, the total power

consumption was around 3 billion kWh in 2006 [87], [96], which is about 10 ~ 20% of its total power consumed by the datacenters networking elements. Reducing the total energy consumption of cloud computing datacenters is important, and a SDN-based network-wide power manager [96] was proposed to analyze the traffic of the datacenters and dynamically adjust the set of active network devices in the datacenters. The power being conserved could range from 25% to 62%, depends on the conditions of the network load [87], [96].

Virtualization, a revolutionary technology, has been considered as the accelerator for improving cloud computing datacenters' advancement. The utilization of VMs provides multiple promising benefits such as isolated environment, workload consolidation and migration, improved manageability, increased fault tolerance, and higher performance [101]. To achieve VMs migration to a different set of physical resources, SDN is employed to carry the migration of VMs, underlying network, and management system [98]. The state of VMs and the networks are maintained and synchronized during the time of migration [97]. In the work of [99], a management framework LiveCloud has been proposed to manage the resources in the cloud computing datacenters for providing elastic cloud architecture in production applications.

Several academic and industrial research efforts have been emphasize on improving datacenters efficiency, power consumption, and carbon emission. Besides, fulfilling *Service Level Agreements* (SLAs) is considered as the major challenges for cloud service providers to their customers with specific requirements on the Quality of Service. The larger the size of the datacenter, the better performance customer can acquire. However, the energy consumption associated with the scale of the datacenter resulting a basic trade-offs with customer satisfaction. SDN technology has been used by several research works to mitigate these cloud computing challenges in cloud computing datacenters [86], [120], [122], [123]. These efforts employed SDN based techniques in support of virtualization, service consolidation, and optimization to allocate resources within the virtualized environment dynamically to minimize the energy cost while fulfilling the SLA.

Enterprise network: Enterprise networks, such as university's networks, Internet infrastructure, and cloud datacenters, all share the common characteristic of having the

massive facilities that are difficult to manage. In the context of management, the network administration, security, and dynamic resources allocation are laborious to control. To reduce the problem scope, the SDN technology can be utilized to provide some indisputable solutions such as hybrid network architecture [64], research testbeds, improved manageability, programmable network, higher security, and high quality network services.

The authors of [14] proposed the employment of OpenFlow protocol in university's network. In [26], this research effort proposed Ethane, a novel network architecture for enterprise networks. Additionally, enterprise WLANs (*Wireless Local Area Networks*) are normally leveraged by many network services such as authorization, load balancing, credential authentication, and mobility [88], [89]. There are some anomaly issues associated with WLANs and the work of [88], [89] proposed a programmable SDN framework to mitigate the issues by implementing WLANs services as the applications. Moreover, the security mechanism is generally controlled by the host security and middle boxes, which usually leads to a complicated synergy between system and the underlying protocols, further producing multiple vulnerabilities such as malfunctioning behavior, and tardy reaction to attacks [90]. The authors of [90] addressed this issue by proposing a design of a system that uses the programmable switches to manage the lower layers traffic for securing the enterprise network. Furthermore, the authors of [91] present a Software Defined MB (Middle Box) networking framework to manage middle boxes.

Wireless network: Currently, the innovation in the field of wireless network has been obscured by the closed and proprietary nature of the wireless infrastructure. To eliminate this obstacle, the work of [102] proposed a SDN-based solution to open the domain of wireless mobile network by providing the freedom for wireless service tenants to move freely between any wireless infrastructure owned by the service providers, and this is achieved by decoupling the network services from the lower level network infrastructure, which enables the innovative technology practitioners, developers, and vendors to dedicate their efforts in the network services area [103]. This work is currently deployed on the university's networks allowing researchers to experiment the new network services in the production network over the virtualization.

Multiple disadvantages are associated with legacy cellular networks such as non-programmable costly equipment, and complex control-plane protocols. A SDN-based solution, namely OpenRoads [104], is proposed to address these issues by emphasizing the simplified design and management framework of cellular networks. OpenRoads provides an open and innovative mobile network platform that enables the novel routing protocols, network services, and mobility functions to be tested [102]. It provides the researchers with full control on two aspects: datapath control using OpenFlow and device configuration control using SNMP [102].

In addition, the programmability of wireless data plane is demonstrated by the effort of OpenRadio [105]. The proposed approach enables the delivery of the declarative and modular programming interfaces through entire wireless stack. OpenRadio redefines the wireless protocols into processing and decisions planes [105]. Moreover, SoftCell [140] provides the capability to direct the traffic dynamically based on the subscriber's profile and executes detailed classification of the packets at the *access* switches, allowing the network state and the bandwidth demands to be processed easily by the software switches. In the context of *Heterogeneous Mobile Networks* (HMNs), SoftMobile [141] provides the efficient control of coordinating complex radio access in HMNs using the concept of SDN.

To build the next generation mobility-centric network, MobilityFirst Internet Architecture Project [106], funded by *Future Internet Architecture* (FIA) of NSF, is proposed to address problems and challenges in the areas of wireless access and mobility associated with large scale wireless network. The project has the goal to provide new multi-path, multicast, any-cast, and context-aware services [106]. Overall, the implementation of this mobility-centric network consists of multiple major protocol components such as identity and network location separation, *Name Certification Service* (NCS) de-centralization, Massively Scalable Global Name Resolution Service (GNRS), Generalized Storage Award Routing, Content and Context-aware Services, and Computing and Storage layer [106], [107]. Moreover, the GENI framework provides realistic environment for the MobilityFirst FIA project to be evaluated and validated with its underlying protocols and algorithms [49], [106]. Currently, the realization of MobilityFirst on GENI constitutes three edge networks with twelve routers [49], and

these edge networks support WiMAX capabilities with Wi-Fi access points for end-user access [49].

To integrate the concept of SDN and NFV in the field of wireless networking, an open experimental testbed EmPOWER [138], is proposed to utilize Open vSwitch and Click Modular Router to build the datapath, and employs Floodlight controller to implement wireless SDN applications. This project comprises 30 programmable access points and 30 power meters for monitoring the power consumption of those access points.

2.10 Conclusion

In this chapter, the architecture, infrastructure, recent deployments, and innovative applications of SDN were discussed. As addressed earlier, it is predicted the market of SDN will keep growing [82]. There are multiple indisputable advantages accompanied by the SDN technology including logically centralized control, opened development, simplified operations, flexible and scalable network management, enhanced network security, programmable interface, and easier to evolve. The existence of the testbeds play a major role in aiding the advancement of SDN technology with the capability to test and experiment the innovative ideas. Additionally, there are some other technologies such as *Network Function Virtualization* (NFV), *Network Virtualization* (NV), *Information Centric Networking* (ICN), and *Recursive InterNetwork Architecture* (RINA), are utilized jointly with SDN to provide more innovative functionalities and better network manageability. To conclude this literature review, we believe SDN framework will play a major role in shaping the future networks as the type of hybrid and integrated networks that fuses the existing network architecture with SDN/OpenFlow-enabled infrastructure.

CHAPTER 3

AUTOMATING THE CONFIGURATION OF MAPREDUCE: A REINFORCEMENT LEARNING SCHEME

3.1 Abstract

With the exponential growth of data and the high demand for the analysis of large datasets, the MapReduce framework has been widely utilized to process data in a timely, cost-effective manner. It is well-known that the performance of MapReduce is limited by its default configuration parameters, and there are a few research studies that have focused on finding the optimal configurations to improve the performance of the MapReduce framework. Recently, machine learning based approaches have been receiving more attention to be utilized to auto configure the MapReduce parameters to account for the dynamic nature of the applications. In this paper, we propose and develop a Reinforcement Learning (RL) based scheme, named RL-MRCONF, to automatically configure the MapReduce parameters. Specifically, we explore and experiment with two variations of RL-MRCONF; one variation is based on the traditional reinforcement learning algorithm and the second is based on the deep reinforcement learning algorithm. Results obtained from simulations show that the RL-MRCONF has the ability to successfully and effectively auto-configure the MapReduce parameters dynamically according to changes in job types and computing resources. Moreover, simulation results show our proposed RL-MRCONF scheme outperforms the traditional RL-based implementation. Using datasets provided by MR-Perf, simulation results show that our proposed scheme provides around 50% performance improvement in terms of execution time when compared with MapReduce using default settings.

3.2 Introduction

With the continuous growth in the volume, velocity, veracity and variability of data (i.e., four V's), big data processing started to play an ever-increasing role in today's datacenters. The Hadoop [141] framework and its MapReduce [143], [144] programming model have been leveraged to address large-scale experimentation, simulation, and computation, and more importantly to address big data challenges, allowing for large-scale data processing across the entire datacenter with superior level of parallelism,

performance, and efficiency. MapReduce is now being heavily used in Google, Facebook, Yahoo!, Amazon, and many other leading companies.

The performance of MapReduce is heavily impacted by its configuration parameters [145]. Therefore, despite the fact that MapReduce has been widely adopted in several fields, system administrators or application developers still face the challenge of having to identify the best configuration parameters to obtain the best performance for their applications running on Hadoop. Re-searchers have demonstrated that MapReduce configurations play a key role on delivering high performance execution. Fortunately, this role can be affected by a limited number of configuration parameters that make a remarkable difference in performance when processing the same MapReduce job [146]. Furthermore, due to the dynamic and heterogeneous nature of Hadoop clusters in terms of computing and networking resources and the variety of the submitted jobs, it is extremely difficult to formulate a mathematical model that captures the dynamics and behavior with all the configurations of Hadoop cluster related to a specific job.

To address this issue, we have implemented a reinforcement learning (RL) scheme, named as RL-MRCONF, to automatically tune and determine the optimal MapReduce configuration parameters based on the type of submitted MapReduce jobs and the cluster resources. Reinforcement learning is a machine learning technique inspired by behavioral psychology where an agent needs to take actions in order to maximize its cumulative reward. For the MapReduce framework in Hadoop cluster, all the possible combinations of the configurable parameters can be treated as a state space, and the action space is defined as the reconfiguration of the parameters in the state space. In particular, our pro-posed RL-MRCONF has the following key contributions:

- It can adapt to all types of jobs submitted to a MapReduce framework, and to the dynamic changes in computing and networking resources within the cluster.
- It has the ability to obtain near optimal configuration tailored to the specific job type, and to the specific hardware profile for the Hadoop cluster automatically.
- It can achieve around 50% of performance improvement in terms of execution time when com-pared with default MapReduce settings.

In this paper, we implement a simulation model of the MapReduce framework using the Network Simulator 3 (NS-3) [148] based on the original work of Wang *et al*. [149], [150], and add an extra layer of RL to automatically deter-mine the near optimal configuration parameters that can be utilized in the MapReduce framework to enhance the performance. The implementation contains two types of reinforcement learning algorithms: (1) Traditional reinforcement learning; (2) Deep reinforcement learning. Both are used to drive the simulation and tweak (learn) the configuration parameters with the intention to determine the proper actions needed to be carried out at each state to maximize the long term accumulated re-ward. Since the majority of jobs submitted to the MapReduce framework are repeated or having similar properties [162], [163], the experienced reinforcement learning agent is able to obtain the near optimal configuration that is suitable to the submitted jobs to obtain the best performance. Additionally, our proposed approach was carried out on original MapReduce framework and we expect the similar results in YARN [141], one of the key features in the 2nd generation of Hadoop, and this has been emphasized in our ongoing research to address usefulness of our proposed scheme in YARN model.

The rest of the paper is organized as follows. Section 3.3 discusses related work and compares them with our scheme; the motivation is briefly discussed in section 3.4; the design structure and algorithm overview are addressed in Section 3.5; the detailed design and implementation of RL-MRCONF are given in Section 3.6; and the performance evaluation is conducted in Section 3.7. Finally, the paper is concluded in Section 3.8.

3.3 Related Work

Research work on improving the performance of MapReduce started as early as 2008. These studies tack-led the MapReduce performance from different perspectives. However, most papers argue that Hadoop MapReduce configuration has to be adjusted based on the available resources, task scheduling, and the application-specific compute requirements within the cluster. The automation and self-configuration of the MapReduce framework have been discussed in several research studies based on the heterogeneous nature of the jobs submitted.

The automation of configuring the MapReduce parameters has been recently addressed using a variety of techniques. In [147], the author shows how MapReduce parameter configuration can play a significant contributing key role in determine performance, and presents different techniques to automate the optimization of MapReduce applications. However, it is unclear how effective the proposed techniques are and there was no clear plan on how to apply them. In addition, in order to ease the challenges in effectively managing the resources of MapReduce clusters, [152] proposed a framework called "*Profiling and Performance Analysis-based System*" (PPABS) for the automatic tuning of Hadoop configuration set-tings based on deduced performance requirements of the application. However, such an approach lacks a "definitive function" on how the "optimal solution" is obtained.

AROMA [153] and *Starfish* [154] are other research efforts that are related to our approach. AROMA uses two-phase machine learning techniques to automatically allocate the computing resources in the cluster and configures Hadoop parameters by adapting the cluster to the newly submitted jobs if their pattern in resource utilization matches the previously processed jobs. However, AROMA observes the resource utilization pattern of the job by statistically calculating the computing resources usage profile every second on each computing node within the cluster, which introduces more overheads. Besides, it uses supervised learning to learn the performance model of each set of jobs, which requires a large number of training samples in order to produce accurate results. Therefore, this approach has low scalability. Starfish provides a method to find the optimal cluster settings by collecting the executed job profile in terms of job-level tuning that selects efficient execution techniques automatically for MapReduce jobs, and work-flow-level tuning by using a "*What-If Engine*," that searches the parameter space for optimal solutions. Such an approach has the potential to fail in searching for optimal parameters due to the complexity of the MapReduce framework.

The research studies in [155], [156] are the most relevant to our research, these studies utilize reinforcement learning of the automatic tuning of parameterized systems and they are proven to be effective. The algorithm adapts well to different types of computing environments. However, the technique that the authors employed to automatically tweak the system considers the parameters in turn (i.e., one parameter at a

time), which leads to efficiency and scalability issues as the number of parameters increases.

The core functionality of RL-MRCONF is based on the reinforcement learning algorithm, a machine learning technique that does not required the correct in-put/output pairs at the starting point, with the clear objective used to establish the policy that maximizes the long term cumulative satisfaction from unknown environment with all its characteristics. Instead of tuning all the configurable parameters one-by-one, our scheme considers all the selected parameters as a whole and processes them jointly to reduce the time needed to search for the best configuration. This technique makes our scheme significantly different from the aforementioned techniques reported in the literature. Further-more, our proposed scheme offers better scalability compared to [155] and [156]. Finally, our scheme makes use of deep reinforcement learning, a combination of reinforcement learning and neural networks, which has the capability to handle more complex systems with superior performance [157], [158], [161].

3.4 Motivation

The motivation of using reinforcement learning in our scheme is inspired by modern robotic learning research where it allows a robot to acquire skills or capabilities to adapt to its environment and respond to different stimuli in the environment, and further use it to strengthen its future behavior through the use of reinforcement. In addition, the efficacy of utilizing reinforcement learning to tune the reconfigurable parametric systems has been proven in [155] and [156], and MapReduce framework shares the similar properties by having a set of reconfigurable parameters.

To realize an automatic self-reconfiguration mechanism, we believe the use of reinforcement learning algorithm is advantageous as it allows the agent to explore the space of configuration options through the feedback given by the environment based on the action performed by the agent in order to acquire the policy that maximizes the long term reward.

3.5 Overview of RL-MRCONF

The details of our proposed RL-MRCONF scheme are discussed in this section. Since we focus on the automatic configuration of MapReduce parameters within a

Hadoop cluster, a brief introduction to the Hadoop and MapReduce frameworks is necessary to review related background and make this research article self-contained.

Configurability of Hadoop MapReduce Framework: The configuration parameters of the Hadoop MapReduce framework can be granulated into 100 or more parameters. Based on the way they affect the performance of MapReduce applications, these parameters can be mainly classified into three categories:

- *Cluster parameters*: These are considered as the environmental parameters to Hadoop cluster, including the path to the temporary data, the maximum number of concurrent MapReduce tasks, buffer size, etc.
- *MR-associated parameters*: These parameters are relevant to the MapReduce operations, with some of them specifically target the Map and Reduce phases, such as the number of concurrent mappers and the number of concurrent reducers.
- *Job-specific parameters*: These parameters are specifically related to job configuration such as the size of each data block, the replication factor, and the amount of memory being utilized when processing the job.

Although the available simulator [149], [150] can be used to allow users to specify the values of the cluster parameters before running MapReduce jobs, in real-world scenarios, it is impractical and costly to change the hard-ware of each computing servers within the cluster. Therefore, the cluster parameters are typically not taken into consideration to further reduce the state space in the reinforcement learning algorithms.

Overview of RL-MRCONF Architecture: There are more than 100 configuration parameters for the Hadoop MapReduce framework, but some of them are more significant than others in impacting the performance. To increase the feasibility of automatic configuration using RL-based approach, we first consider the most performance-significant parameters suggested in [147], [152], [153], [154] to generate candidate configurations. The number of configurable parameters dramatically increases the states space, leading to the possibility of state explosion that would cause the significant overhead to produce reasonable results in a timely fashion. The initial value ranges of the selected parameters are also carefully determined based on the prior work done by [147], [152]; therefore, we reduce the learning time of our scheme since we only consider the configuration parameters and values that significantly impact the

performance. In our scheme, the chosen configuration parameters are initialized as shown in Table 5, where it contains the default value of those chosen parameters as suggested by Hadoop [141], and the range of values constructed by prior studies, along with the description of each parameter. Our scheme uses an RL-based agent that automatically configures the Hadoop MapReduce framework. The agent changes the configuration parameters and monitors the performance in terms of the execution time while making decisions based on what it learned from previous experiences.

Table 5. Overview of Candidate MapReduce Configuration Parameters.

Parameter Name	Default Value	Values Considered	Description
mapreduce.tasktracker.map.tasks.maximum	2	1 ~ 15	The maximum number of map tasks concurrently running on a node
mapreduce.tasktracker.reduce.tasks.maximum	2	1 ~ 15	The maximum number of reduce tasks concurrently running on a node
mapreduce.task.io.sort.factor	10	10 ~ 150	The number of streams used while sorting files
mapreduce.task.io.sort.mb (MB)	100	60 ~ 200	The amount of buffer memory used when sorting files
mapreduce.map.sort.spill.percent	0.8	0.66 ~ 0.8	The soft limit that determines whether to spill contents to disk
dfs.block.size (MB)	64	32 ~ 512	The size of the data block used in the file system

Reinforcement learning is a machine learning technique that is inspired by behaviorist psychology where an agent needs to take actions in order to maximize its cumulative reward. A finite *Markov Decision Process* (MDP) is typically used to formulate a reinforcement learning model. The model contains a set of states and several actions associated in each state. The learning agent receives a reward during the state transition process, defined by a reward function and the ultimate goal of the agent is to develop a policy $\pi : S \rightarrow A$ to maximize the obtained cumulative rewards on the longer term from the iterative trial-and-error learning processes [3]. In this paper, we utilize following reinforcement learning algorithms:

Traditional Reinforcement Learning: We transform the automatic parameter reconfiguration problem of the MapReduce framework into a MDP by implementing a

state space $S = \{s_1, s_2, \ldots, s_i\}$, a space of actions $A = \{a_1, a_2, \ldots, a_j\}$, and the immediate reward function $r(s,\ a)$. The notions of state space, action space, and reward function used in our model are defined as follows:

- *State Space*: The state space in our model represents all the possible combinations of framework configurations. For the chosen group of six parameters shown in Table 5, we denote a state as follows:

$$s_i = (Param_1, Param_2, Param_3\ \ldots, Param_6)$$

- *Action Space*: The action space in our model consists of the following: (1) *No action*; (2) *increase*; and (3) *decrease* action associated with each parameter. For example, we denote the decrease action on parameter *i* as follows:

$$a_i^{decrease} = (\ldots, Param_i^{decrease}, \ldots, Param_n)$$

- *Immediate Reward Function*: The immediate re-ward is determined based on the simulation execution time (which is the primar performance metric) using the given values for the configuration parameters. The immediate reward is represented as follows:

$$r_t = Compare(perf_{current\ best}, perf_t)$$

where $perf_{current\ best}$ is the current best performance obtained so far by the simulation. For a given comparison, a lower performance time returns a positive reward +1 to the agent; otherwise, the agent is given a negative value -1 as the reward. If $perf_t$ equals $perf_{current\ best}$ then a 0 reward is given.

The *temporal difference* (DT) **Q-Learning** algorithm is used in our simulation to update the Q-values at each time step based on the estimation of the algorithm (i.e., action-value function). The learning process is an iterative process in which the agent interacts with environment in discrete time steps. In each time step, the agent chooses an action $a_t \in A$ in the given state s_t. The state of the agent is changed from s_t to s_{t+1} when it takes action a_t and obtains the immediate reward r_{t+1}. The agent makes its own decisions to choose an action to be taken with the goal of maximizing the expected reward. The notation $Q(s, a)$ denotes the average Q-value of an action *a* at state *s*. Once the immediate reward is collected, the $Q(s, a)$ can be further refined as:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha * [r_{t+1} + \gamma * \max_a Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \quad (1)$$

where α $(0 < \alpha \leq 1)$ is the learning rate that determines to what degree the newly obtained information will override the old one and γ $(0 \leq \gamma \leq 1)$ is the discount factor that defines the importance of future reward and guarantees convergence of the accumulated reward. The action selection is based on the *ϵ-greedy* policy. A greedy policy always selects the action with highest Q-value, whereas the *ϵ-greedy* policy has the small probability, ϵ, for selecting another action at random. This allows the algorithm to occasionally explore the action space uniformly at random, in the hope of finding a better action to ensure proper state convergence. The algorithm represents the "*quality*" of a certain action in a given state; hence, it is also known as *Q-function*. The pseudocode of the Q-Learning algorithm utilized in our work is presented in Algorithm 1.

Algorithm 1: Q-Learning Algorithm

Pre-condition:
Initialize Q table with small random number
Initialize state s_t
Initialize goal μ
Procedure:
improvement = 0
repeat
  for (step = 0; step < learning_iteration; step++)
    Get action a_t from s_t using $\epsilon - greedy$ policy
    Get parameter $param_t$ from s_t using $\epsilon - greedy$ policy
    $\epsilon = \epsilon$ – (step / learning_iter) * ϵ
    Take action a_t on the $param_t$ and receive reward *r*, obtain $runtime_t$
    Sample new state s_{t+1} after applied action a_t
    Update $Q_t \leftarrow Q_t + \alpha * (r + \gamma * \max Q_{t+1} - Q_t)$
    Update the corresponding $param_t$ of Q_t
    improvement = get_improvement($best_t, worst_t$)
    $s_t = s_{t+1}, a_{t+1}$ = Get action from $s_t, a_t = a_{t+1}$
  end for
until *improvement* > μ

Deep Reinforcement Learning: The deep reinforcement learning algorithm is a variation of traditional RL implemented on top of a new type of learning agent, called deep Q-network (**DQN**), which is the combination of traditional reinforcement learning with a class of artificial neural network known as deep neural network [157].

Artificial Neural Networks (ANNs) are networks of primitive neurons inspired by the process of how the human brain works [156]. They can be used to approximate

functions by learning from large sets of input data. The feed-forward neural network is the only approach used in DQN and consists of the basic components depicted in Figure 14: (1) *Neurons* inter-connected using unidirectional links to form a network; (2) *Weights* associated with each connection; (3) *Layers* consisting of a number of neurons.

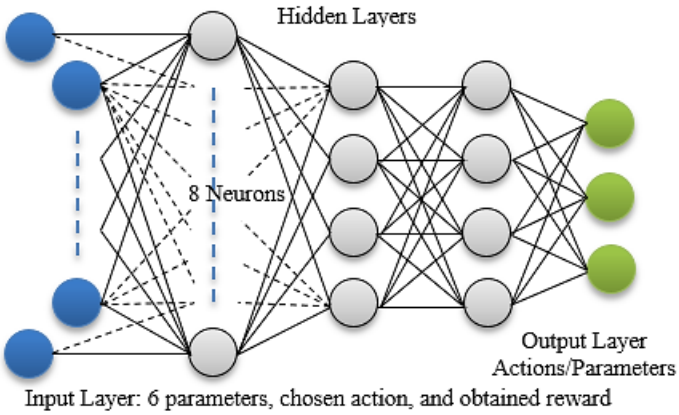


Figure 14. The feedforward neural network architecture of deep reinforcement learning

A neural network with multiple hidden layers is considered as a *deep neural network* (DNN) [5]. A notable property of DNNs is that the number hidden layers and the number of neurons in each layer are free parameters. This means that there is no predefined or default value for these parameters. The larger/deeper the neural network, the more complex application it can model. An activation function is used to calculate the activation value and propagate it between connected neurons. The numeric weight value associated with each connection between two neurons is used to determine the weight of the link. To compute the activation value a_j from neuron *i* to neuron *j*, the *j* neuron needs to compute a weighted sum of all of its inputs, and then applies an activation function *f* as follows:

$$a_j = f(\sum_{i=0}^{n} w_{i,j} * a_i) \quad (2)$$

There are several flavors of activation functions available based on the users' interest, including *Sigmoid*, *SoftMax*, *Rectified Liner Unit* (ReLU), etc. Different types of activation functions have different ways to calculate the activation value. Among all the available activation functions, ReLU is gaining the popularity in deep neural networks due to its faster and more efficient learning in terms of reducing the likelihood of

vanishing gradient problem, and sparsity in activation [5], which provides better degrees of freedom in learning from the underlying complex dataset. The ReLU has the following form:

$$f(x) = \begin{cases} 0, \text{for } x < 0 \\ x, \text{for } x \geq 0 \end{cases} \quad (3)$$

The feedforward DQN for deep reinforcement learning is used in our scheme. In our scheme, we assume that among all the states in our state space, some of them would never or rarely be chosen to run the simulation; otherwise, it would take long time for the Q-table to converge. Therefore, it is a good idea to guess (predict) the Q-values for those rarely chosen states instead of actually processing them in our simulation. The prediction is based on the states being processed previously (i.e., during the training phase). With this in mind, we can represent our original Q-function with a deep neural network that takes the state (i.e., MapReduce configuration parameters) and the selected action as the input and output the Q-value for all possible actions.

During the training phase, an array of experience memory is used to store all the experienced transitions (s, a, r, s'), where *s* denotes the current state, *a* denotes the action selected to be carried out, *r* denotes the reward obtained, and s' denotes the next state after the action is applied. Random mini-batches from the experience memory are chosen to train the network instead of using the most cur-rent transition in order to avoid the similarity of subsequent training samples. The Q-network update in the *i*th iteration is formulated by the following loss function:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')} \sim P(D) \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a, ; \theta_i) \right)^2 \right] \quad (4)$$

where $\mathbb{E}_{(s,a,r,s')}$ is the processed experience which is uniformly drawn using the probability $P(D)$ from the experience memory, γ is the discount factor, and θ_i is the state of the Q-network in the *i*th iteration, whereas θ_{i-1} is the state used to compute the target in the *i*th iteration.

Similar to the traditional RL, the ϵ-*greedy* policy is also used to select the action to perform based on the highest Q-value associated with that action. The initial ϵ value is set to 1 at the beginning so that the agent can choose random actions. This value is then

decreased over time in order to maintain a fixed exploration rate. The pseudocode of the Deep Q-Network algorithm utilized in our work is presented in Algorithm 2.

Algorithm 2: Deep Q-Network Algorithm

Pre-condition:
Initialize expereince memory M
Initialize action-value pair Q with random weights
Initialize state s_t
Initialize goal μ
Procedure:
01:*improvement* = 0
02:**repeat**
03: **for** (step = 0; step < learning_iteration; step++)
04: Get action a_t from s_t using $\epsilon - greedy$ policy
05: Get parameter $param_t$ from s_t using $\epsilon - greedy$ policy
06: $\epsilon = \epsilon$ – (step / learning_iter) * ϵ
07: Take action a_t on $param_t$ and receive reward *r*, obtain $runtime_t$
08: Observe new state s_{t+1}
09: Store experience memory (s, a, r, s_{t+1}) into M
10: Sample *n* random transitions (s', a', r', s'') from M
11: Update $transition\ tt \leftarrow r' + \gamma * \max(s'' - a'')$
12: Update the $param_t$ of θ_i
13: Train the Q network using *loss* = $(tt - Q(s', a'))^2$
14: *improvement* = get_improvement($best_t, worst_t$)
15: $s_t = s_{t+1}, a_{t+1}$ = Get action from $s_t, a_t = a_{t+1}$
16: **end for**
17:**until** *improvement* > μ

3.6 The Design and Implementation of RL-MRCONF

To demonstrate the usefulness of our proposed RL-based scheme to auto-configure the MapReduce parameters, we implemented a simulator for the MapReduce framework. The details of our simulator are presented in the following sub-sections.

Operation Phases of RL-MRCONF: Real-time MapReduce auto-reconfiguration has a time efficiency requirements, which makes conventional RL approaches impractical to the system that needs to process the information in a timely manner. To reduce the initial learning overhead, we setup our RL-MRCONF into two phase: orchestration phase and production phase. The cluster settings used in the orchestration phase are identical with the cluster used in the production phase.

From Figure 15 we can see the orchestration phase is where the learning agent spends time to obtain the near optimal configuration while acquiring the experience through the submitted jobs to the MapReduce framework. Once the best-effort near optimal configuration of each type of job is obtained, the configuration along with the

initialized policy will be deployed onto the production system for improving the performance while the learning agent continues to learn over time to adapt the dynamic nature of the cluster.

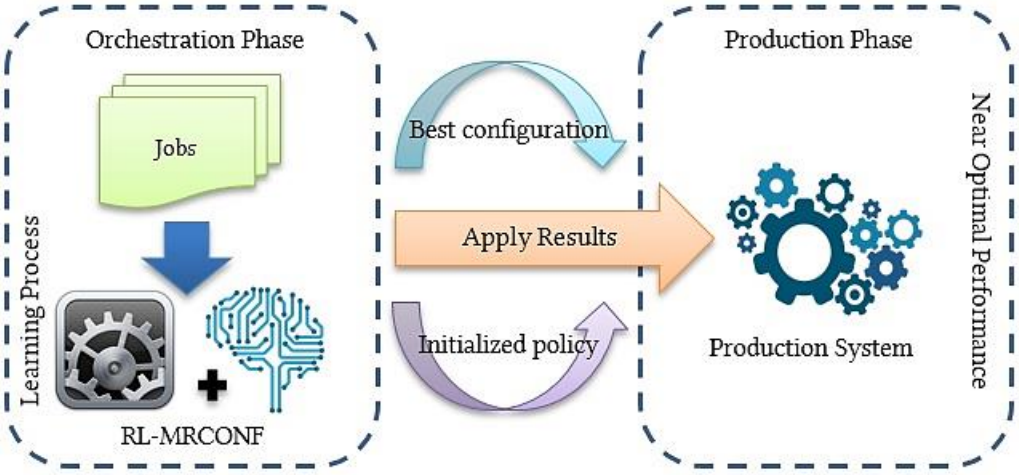


Figure 15. The operation phases of RL-MRCONF

MR-Perf Simulator: *MR-Perf* [149], [150] is the simulator that models Hadoop cluster and its MapReduce framework on top of the *NS-2* simulation engine. The goal of MR-Perf is to serve as a design and planning tool for MapReduce infrastructure and deployment. It provides an interface for easy configuration of a number of configurable parameters that currently need to be hand-tweaked using rules of thumb.

The simulator requires several files as input to specify hardware profile of participating nodes, cluster topology, data specification, and job description. The simulation output contains a detailed trace of the jobs executed by the simulator, including job execution time (i.e., runtime), the time stamped status of each task phase, and the amount of data being transferred. As illustrated in Figure 16, the Job Tracker is the main driver of the simulator responsible for generating map and reduce tasks, monitoring progresses of different phases, and producing the final results when job is completed. The simulated behavior is modeled in response to the messages received from other nodes. Besides, the functions and operations of the Job Tracker are initiated and captured using heartbeat trigger, which is implemented using timer mechanism that ticks every 1 second of the simulation time.

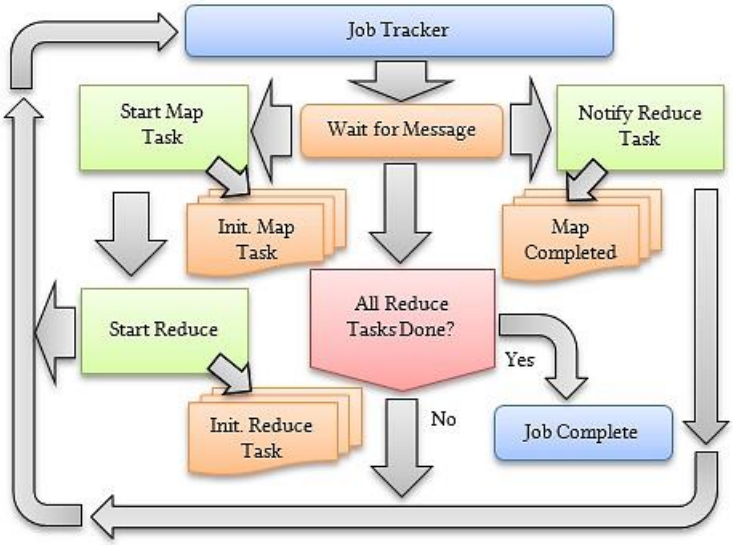


Figure 16. MR-Perf workflow

When simulation starts, it reads all the configurable parameters specified by the input files, and instantiates the participating nodes according to the topology specification file. The jobs are submitted based on the scheduled simulation time, and each node starts to process the scheduled job and communicates with other nodes using the NS-2 simulation engine.

MR-Perf provides key features and characteristics for the application within its simulated environment to take advantage of which include:

- ***TeraSort***: This application is inspired by the TeraSort benchmark [141] that measures the time required to sort one terabyte of data that is randomly distributed within the cluster.
- ***Search***: This application matches each input data with a set of criteria and outputs the data of the match results. The CPU load represents the complexity of the match criteria in the map tasks. The search application, characterized by match complexity, allows the simulator to observe the relationship between the map times given fixed input and output sizes.
- ***Index***: This application generates map or reduce the output for each unique word found in the input data. The number of unique words in the input data defines the size of output data. Characterized by the fraction of the unique input words, the

index application allows the simulator to observe the impact of the size of the intermediate data on the map time.

The characteristic of the above applications (jobs) are defined by using two parameters in the MR-Perf simulator: *cycle/byte* and *filter ratio*. The Cycle/byte parameter models the complexity of the computation by defining the number of compute cycles used per input byte. The filter ratio parameter is used to model the difference in size between the input and output data during the map phase.

Integration with NS-3: As was mentioned before, our implementation of the MapReduce framework simulation is based on the work of Wang *et al.* [149], [150], with the goal of making MR-Perf deployable on top of the NS-3 simulator. MR-Perf was implemented using multiple programming languages including C++, TCL, and Python. Due to the nature of the code base of NS-3, we use C++ as our primary programming language in order to maintain full compatibility. Fortunately, approximately 60% of code of MR-Perf is implemented using C++.

With this objectives in mind, we made no change to the core logic of MR-Perf. However, we converted the Python parts of the code to C++ for integration purposes. In addition, MR-Perf requires multiple XML files as input for the specifications of the participating nodes, network topology, and jobs property. We eliminated this requirement by writing additional code to incorporate the integration of these parameters.

Our simulator design made use of the NS-3 `Application` class extensively to create functionalities to respond to events (sending/receiving packets). We also extended the base class `Node` for adding more functionalities in computing resources such as CPUs, memory, and hard drive of each node instance so that the node level computing statistics can be collected by the simulator. Moreover, the extended application modules are *installed* on to each node that runs individually within the node domain, and the computing power is distributed equally among the assigned jobs for the node. The basics and foundations of the simulation workflow and processes can be found in [149], [150].

3.7 The Evaluation of RL-MRCONF

In this section, we evaluate the usefulness and effectiveness of the reinforcement learning based scheme in auto-configuring the MapReduce configurable parameters. We

also conduct systematic experiments to compare the traditional RL with deep RL. The performance metric considered in our study is the execution time of the MapReduce job.

To evaluate the effectiveness of our RL-MRCONF scheme, we setup a dedicated server machine to run the simulation experiments. The physical server is equipped with two Intel Xeon quad-core E5520 CPUs with 64GB of memory. The simulation is compiled using Microsoft Visual Studio with Visual C++ 11 compiler.

The simulation starts by generating all the possible combinations of the chosen 6 configurable parameters shown in Table 5. Then, the program starts creating simulation nodes, building the cluster topology, loading the initial MapReduce configuration, and starting the simulation. The detailed simulation parameters are presented in Table 6. These parameters are not considered configurable in our RL based scheme since they remain constant until the physical machines of the cluster are upgraded.

Table 6. The Configuration Parameters of NS-3 Simulator.

Simulation Parameter	Values
Number of nodes	30
Number of racks	2
Number of switch per rack	1
Number of routers	1
Node hardware profile	CPUs: 2 Cores per CPU: 2 Memory: 16GB Hard disk capacity: 500GB Hard disk read bandwidth: 280MB/sec. Hard disk write bandwidth: 75MB/sec. Hard disk read/write seek time: 0.2ms
Links profile between nodes	Intra rack link bandwidth: 1Gbps Intra rack link latency: 0.15ms Inter rack link bandwidth: 10Gbps Inter rack link latency: 0.05ms
Submitted jobs type	Search TeraSort Index
HDFS file size	2 GB

Before the MapReduce process starts, the HDFS data write process discussed in section 4 must take place to ensure the availability of the data for the MapReduce process. In our simulations, we assume the data is divided based on the default data

replication factor of 3 and the size of each replica (i.e., data block). We also assume that the replicas are well-distributed based on the default replica placement policy of Hadoop [141]. This policy ensures that two data block replicas are placed in the same rack to leverage the in-rack network bandwidth and fast failover. The third replica is placed onto different rack to maintain failover capability while ensuring data availability in the event that the rack becomes offline.

The performance of each MapReduce configuration is defined by the execution time of the submitted job given the current configuration used for the MapReduce framework. Each job has its own submission time. Once the job is processed successfully, the simulator computes the actual job processing time by subtracting the job submission time from the current simulation time when the job completes its execution.

Effectiveness of Traditional Reinforcement Learning (RL): In this section, we discuss the effectiveness of our proposed RL-MRCONF scheme to auto-configure the MapReduce framework. The RL-MRCONF agent automatically tweaks the system starting from an initial set of random configuration parameters and tunes these configurations based on the policy it learns. The agent does not tune these six parameters individually, but collectively. This repetitive process stops when the configuration parameters yield a job execution time that satisfies our objective, say at least 50% improvement on the execution time. Such a configuration will be considered optimal for the given job.

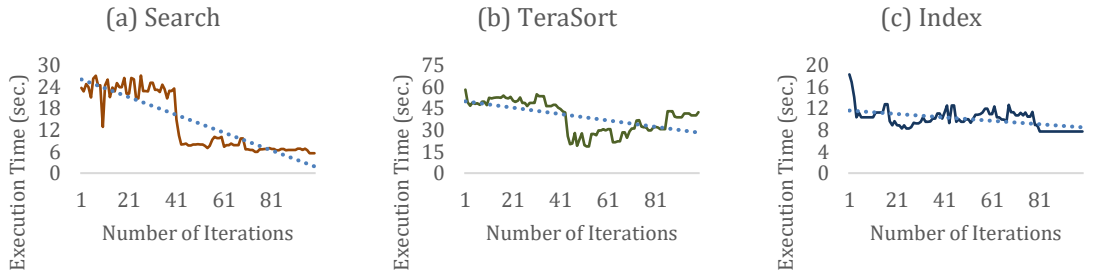


Figure 17. RL-MRCONF execution time vs. number of iterations for different types of jobs

Adaptability of RL-MRCONF: In our first experiment, we studied the performance of RL-MRCONF on three job types; namely, Search, Index, and TeraSort.

The goal of this experiment is to evaluate the adaptability of the RL-MRCONF agent to the different job types. The simulator processes each job for 100 iterations before it processes another type of job. The simulation results are depicted in Figure 17 where it shows a clear performance improvement in terms of execution time for the three job types.

In our scheme, the goal of the RL-MRCONF agent is to obtain the MapReduce configuration that can yield at least 50% improvement on the execution time compared to the initial configuration. The results shown in Figure 17 demonstrate the effectiveness and adaptability of our proposed scheme. The results show that the execution time of all the jobs processed by our scheme has be improved by at least 50% compared to the execution time obtained initially using the initial configuration. From Figure 17 (a) and 17 (b) we can see the execution time drops after 41st iteration, 25th iteration for Figure 17 (c), for the execution time is improved at least 50%. Based on the results presented in Figure 17, it can be seen that RL-MRCONF adapts to the different job types and that the agent can find a near optimal configuration within 50 learning iterations. In addition, the best configuration found resulted in 50% better performance in terms of execution time compared to the execution time using an initial random configuration.

It should be observed that in some cases the performance obtained by the RL policy deteriorates towards the end of the learning iterations. For instance, the execution time of the TeraSort job in Figure 17 starts to deteriorate after the 51st iteration. We have determined that this is due to the nature of the ϵ-greedy policy that has ϵ probability to pick some other action at random, regardless of its Q-value. Although the exploration rate ϵ decrease as number of iterations goes up, the learning agent still has the possibility to perform an action that deviates from the known best action. Another instance of this situation occurs on the index job after the 25th iteration.

Significance of each chosen parameter: In our second experiment, we analyzed the significance of each parameter on the performance of the MapReduce framework. Figure 18 presents a series of values of each parameter selected by our RL-MRCONF agent (rows 2 through 4 on Figure 18) and the corresponding execution time (row 1 on Figure 18). As expected, the parameters *map.tasks.maximum* and *reduce.tasks.maximum* have more significance to impact the performance compared to other parameters. From

Figure 18 (a), it can be seen that the execution time tends to go down for the search job as the maximum number of map tasks parameter increases while the maximum number of reduce tasks remains mostly unchanged for most of the iterations. This situation does not apply to all the jobs submitted to the MapReduce simulator. For example, as depicted in Figure 18 (b) for the TeraSort job, the execution time tends to decrease as the maximum number of reduce tasks parameter increases whereas the maximum number of map tasks remains between 1 and 3.

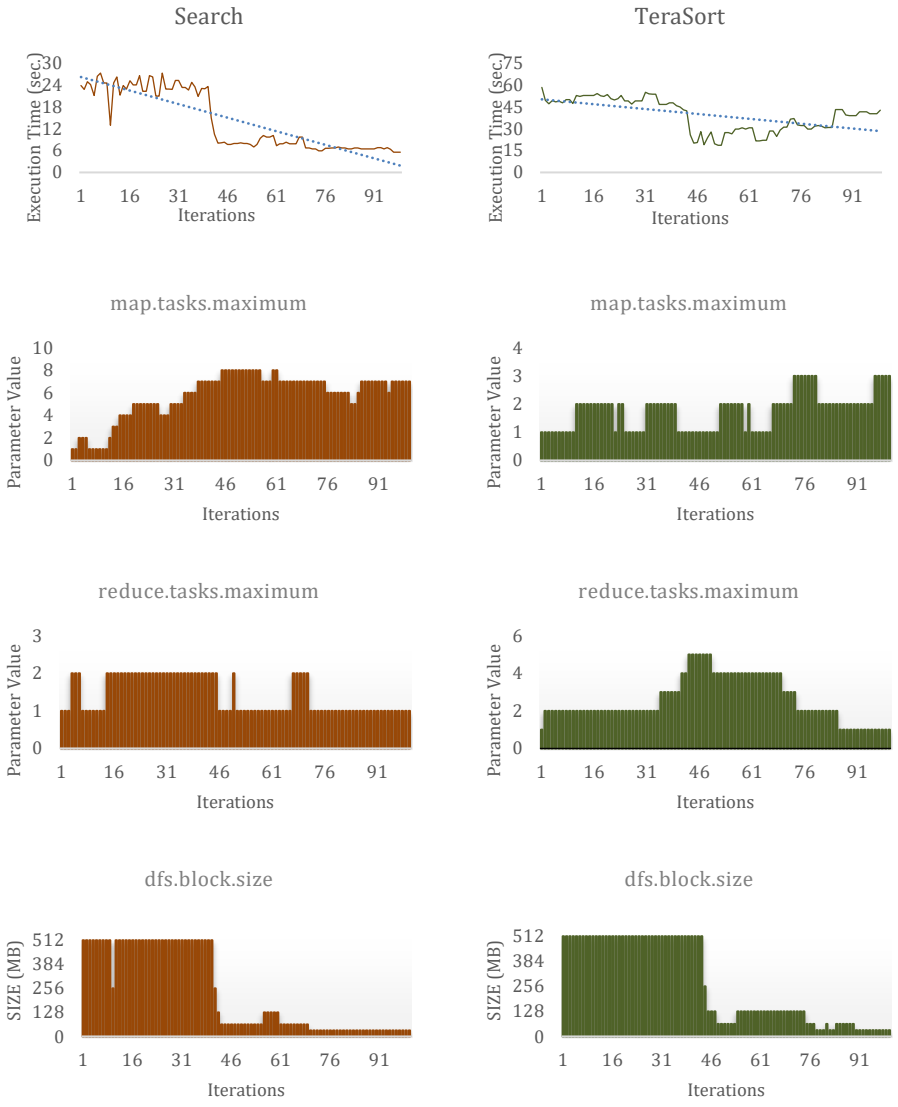


Figure 18. Relationship between execution time and the key input parameters

A common observation to these two jobs is impact of the *dfs.block.size* parameter on the execution time. The execution time tends to decrease as the size of the replicated data blocks decreases. This supports the fact that the size of the data block has a significant role in determining the performance of the MapReduce process.

These results indicate that our proposed RL-MRCONF leads to significant execution time improvements and adapts well to the job types submitted to the MapReduce framework. The *Search* job requires more mappers to match each input data with a set of criteria whereas the *TeraSort* job requires more reducers to perform the sorting process.

Effectiveness of initialized Q-table: The number of configurable parameters determine the size of the state space of the RL algorithm. This size grows exponentially as the number of parameters increases. Due to the nature of the RL algorithm, the RL agent performs a large number of explorations before it reaches the goal state. We believe this significantly degrades the performance of the MapReduce framework.

In our third experiment, we observed the impact of the initialized Q-table (i.e., initialized policy) on improving the performance of our proposed scheme. The motivation behind this experiment is that in a typical setting, the same jobs or jobs with similar properties are submitted to the MapReduce process. Hence, under this assumption, we argue that our RL-MRCONF agent can achieve the objective we set for the end state of the learning process faster, and minimize the time and computing resources needed in order to find a near optimal solution.

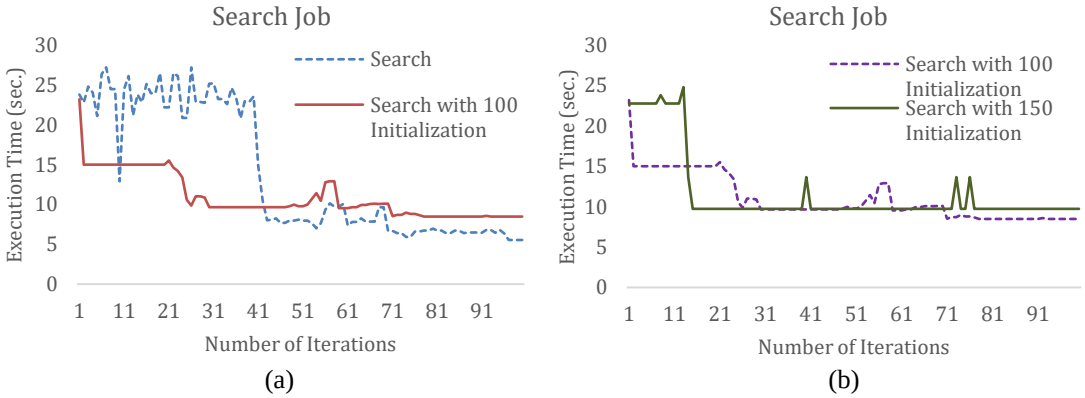


Figure 19. Performance with and without Q-table initialization while varying number of iteractions

Figure 19 presents the performance of the RL-MRCONF agent with and without initializing the Q-table for the search job. As depicted in the figures, the agent with an initialized Q-table (trained) yields a considerable performance improvement over a shorter time interval. Figure 19 (a) shows that the initialized policy can lead the agent to obtain at least 35% improvement from the 2nd iteration and finds a configuration with at least 57% improvement in the 27th iteration. Conversely, the agent without initialized Q-table finds the configuration that produces 66% performance improvement after the 44th iteration; thus, resulting in a longer time to search for the best configuration.

In this experiment, we further analyzed the impact of the number of Q-table learning iterations on the execution time. Figure 19 (b) shows the execution time when the Q-table is initialized after 100 and 150 learning iterations, respectively. We observe that agent with the Q-table that has been initialized over 150 learning iterations can achieve 57% performance improvement in less than 16 iterations. Again, this is faster compared to the one initialized after 100 learning iterations. This demonstrates that a larger number of learning iterations used to initialize the Q-table can significantly improve the convergence time to a goal state that achieves the target improvement.

From this experiment, it can be noticed that the RL-MRCONF agent without initialization can find the better MapReduce configuration compared to the two other initialized agents. This is due to the dynamic exploration rate used during the learning process, which is defined based on the ratio of the current iteration to the total number of learning iterations. The value of ϵ decreases as the iteration index increases. When we evaluate the initialized Q-table used by the agent, we use a static exploration rate as small as 0.1 so that the agent performs less exploration and more exploitation of its initialized Q-table to perform the proper actions based on its past experience in order to obtain better accumulative long-term reward.

The uninitialized agent achieves a 71% improvement on the execution time but the agent requires long time to find such configuration. This can be unacceptable for time sensitive systems that require fast or near real-real time processing of big data. Conversely, the figure shows that the two other initialized agents obtained a good configuration in less time. Therefore, the RL-MRCONF agent with initialized policy can obtain a near optimal configuration in a timely manner. The RL-MRCONF performs well

to fit applications that require fast or near real-time processing of big data. Moreover, the RL-based approach continues to learn over time which allows the MapReduce framework to adapt regardless of the type of job submitted to the cluster.

Table 7. The Best Obtained MapReduce Configuration Parameters for Job Types.

MapReduce Parameters	Search	Index	TeraSort
mapreduce.tasktracker.map.tasks.maximum	7	3	2
mapreduce.tasktracker.reduce.tasks.maximum	1	4	4
mapreduce.task.io.sort.factor	10	40	20
mapreduce.task.io.sort.mb (MB)	80	80	100
mapreduce.map.sort.spill.percent	0.7	0.67	0.67
dfs.block.size (MB)	32	64	64
Best Runtime (sec.)	7.0632	7.3198	18.574
Initial Runtime (sec.)	24.468	16.916	47.178
Performance Improvement	71.13%	56.73%	60.63%

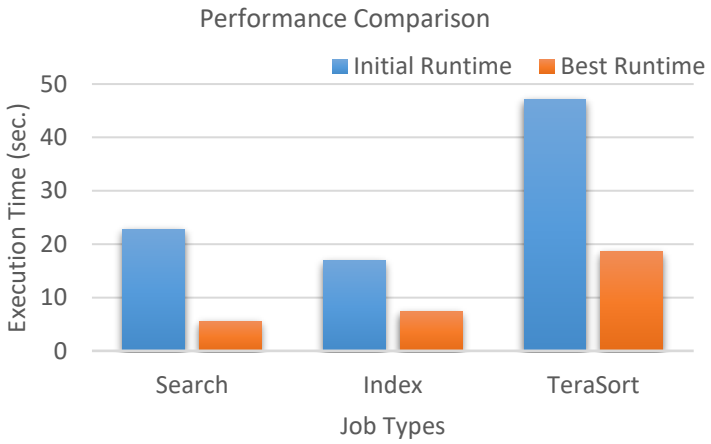


Figure 20. Performance comparison between the execution times using the initial and best configuration for different job types

Table 7 ranks in terms of performance the MapReduce configurations obtained by our uninitialized RL-MRCONF agent that correspond to the different job types and the resulting percentage of improvement. Figure 20 illustrates the effectiveness of our scheme in improving the execution time for the different job types. The initial configuration is defined as the worst case scenario when the parameters are poorly configured.

Effectiveness of Deep Reinforcement Learning (DQN): In this section, we conduct experiments to study the effectiveness of deep reinforcement learning to support auto-configuration of MapReduce. Our simulator was modified to use Caffe library to develop a deep Q-network. Similar to our RL-based scheme, the DQN-based RL-MRCONF agent automatically tunes the configurable MapReduce parameters based on what it learned during the training process. Instead of evaluating the chosen actions individually on a given configuration from the previous state as implemented in the traditional RL agent, the Deep Q-Network estimates the possible reward for all the actions for the given configuration produced by applying the chosen action on the configuration from the previous state.

Before the DQN-based RL-MRCONF agent starts to learn the environment though the simulation, the underlying neural network, as illustrated in Figure 14, is constructed with 8 neurons in the input layer (6 configurable parameters + the chosen action + the immediate reward), and 3 neurons in the output layer (3 actions) with Q-value associated with each action. The number of hidden layers can be freely determined based on user's preference. In this research work, we consider 3 hidden layers.

During the training phase, the *Stochastic Gradient Descent* (SGD) method is used to obtain the minimum loss value. A vector of memory is used to collect all the processed states associated with the action performed and the parameter selected by the deep Q-network learning agent. These processed states are referred to as transitions in our scheme. Each transition is a tuple that consists of the previous state, the chosen action, the reward obtained by performing the action, the selected parameter, and the next state produced by the action. When the size of the memory vector reaches a certain threshold, 50 transitions in our simulation, the algorithm updates the network by randomly selecting 4 transitions and uses them to calculate the weights in the network. The update process lasts until the agent reaches the goal state. The goal state requires 40% performance improvement on the initial configuration when using the DQN-based RL-MRCONF scheme.

In order to evaluate the effectiveness of the trained network, we considered and designed 3 training scenarios to evaluate the performance of the DQN that is trained with 10, 20, and 30 uniformly selected random configurations. In theory, the more samples get

used to train the network, the better the performance of the DQN in finding a near optimal configuration. Therefore we expect the network trained with 30 randomly selected configurations to perform better than the one with 10 or 20 configurations.

The evaluation of the DQN is conducted by feeding a configuration that excludes the randomly picked training configurations set to the network, as follows:

$$Config_{Evaluation} \notin \{Configs_{Training}\}$$

and evaluate its performance by obtaining the number of iterations for the DQN-based RL-MRCONF agent requires to find a configuration that satisfies our objective.

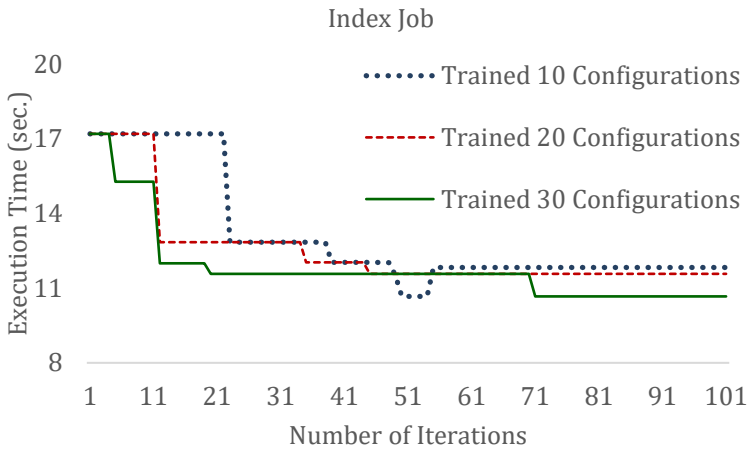


Figure 21. Deep reinforcement learning performance with different training scenarios

The performance of the deep Q-network that has been trained under different training scenarios is depicted in Figure 21. The Figure shows that the DQN-based RL-MRCONF agent trained with 30 randomly chosen configurations outperforms the two other scenarios. In this scenario, the goal configuration was reached after 18 iterations whereas the other scenarios required 35 or more iterations to obtain a configuration that satisfies the goal. This clearly supports our initial hypothesis that the more samples used to train the network, the better the performance of the DQN in finding a near optimal configuration. Similar to the traditional reinforcement learning algorithm, the exploration and learning rates of the DQN-based RL-MRCONF agent are set to a value of 0.1 when evaluating the performance of the agent.

Performance Comparison between Algorithms: The previous sections illustrate the effectiveness and usefulness of RL and DQN to support the automatic configuration of MapReduce. In this section, we conduct experiments to compare the performance and the effectiveness of RL with DQN. As discussed in [158], [159], [161], the DQN has been demonstrated to be effective in solving complex problems.

To compare the performance of RL and DQN, we designed a training/initialization scenario that include 10, 20, and 30 uniformly selected random configurations. The configurations used to evaluate both algorithms are excluded from the chosen sets of training configurations to fairly measure the performance of the trained/initialized Q-matrix and Q- network.

In our first experiment, we evaluated the performance of the learning agent using one evaluation configuration that is not part of the training set. In other words, this evaluation configuration is considered as the new configuration state by the agents, and how the agents react to this configuration completely relies on their experiences and the knowledge obtained from the training dataset. The objective of both agents is set to 40% for the performance improvement. The Index Job provided by the MR-Perf framework is used to evaluate the agents.

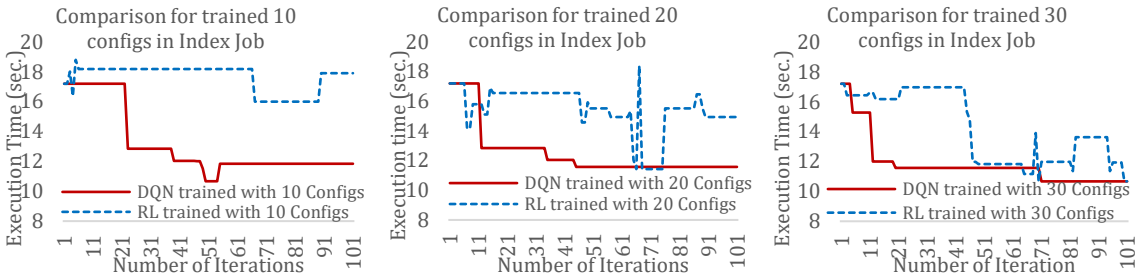


Figure 22. Performance comparison between RL and DQN under different training scenarios

As illustrated in Figure 22, it can be observed that the performance of the DQN agent is always better than the RL agent regardless of the number of configurations used during the training process. It can also be noticed that the DQN-based RL-MRCONF agent is capable of finding the better configuration in a shorter period of time in terms of the number of required iterations needed to converge to the goal state. Moreover, the

DQN-based RL-MRCONF agent outperforms the RL-MRCONF agent under the same training scenarios with the same type of job.

With the same settings as our previous experiment, we evaluate the performance of both learning agents using the Search Job and an objective of 50% performance improvement. Based on the results from our previous experiments, we can conclude that the more the agent is trained, the faster it can find the optimal solution that satisfies the objective. In contract to our first experiment, we focus on the relationship between training scenarios and the near-optimal configuration found by the agents. The evaluation process terminates when the agent satisfies the pre-defined goal.

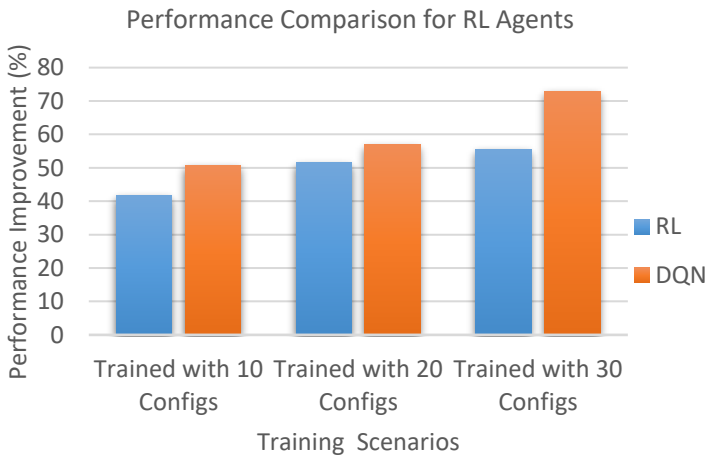


Figure 23. Performance evaluation of RL and DQN agents under the same training and objective

Our experimental results show that under the same training scenarios (i.e., 10, 20, and 30 sets of training configurations), the DQN-based agent always finds better configuration compared to the RL agent (c.f. Figure 23). From the results, it can be concluded that the RL agent trained with 10 randomly chosen configurations could not find the configuration that satisfies the predefined goal within the limited number of iterations (i.e., 100 iterations), whereas the DQN-based agent can satisfy the goal within the limited number of iterations. Moreover, for both agents being trained under 30 randomly chosen configurations, the DQN-based can even find the configuration that produces 72.8% performance improvement, while the RL agent finds the configuration that yields 55.4% improvement in performance. The results of our second experiment

negate and invalidate the naïve assumption that both agents produce similar results given similar training sets. Our results clearly show that the DQN-based agent performs better than the RL and the performance gap grows as the training set size increases.

By combining the results of those two experiments, we conclude that the effectiveness and the performance of the DQN-based RL-MRCONF learning agent are better than the Original RL-MRCONF learning agent. Our experimental results clearly demonstrate the superiority of the Deep Reinforcement Learning algorithm.

Validation Experiment: In this section, we conduct the experiment to verify the correctness of our simulation results by collecting data from a real cluster environment and compared it with that obtained through RL-MRCONF. The cluster was configured using Cloudera [166] CDH 5 with 30 nodes divided into 2 sever rack groups, and each node is equipped with 2 CPUs, 16GB of memory, and 500 GB of hard drive capacity that can be utilized by the unified interface of Cloudera Manager. The near optimal configuration obtained by the RL-MRCONF will be brought over to the Cloudera cluster for this validation experiment.

Table 8. Performance Comparison Based on Job Types between RL-MRCONF and Real Hadoop Cluster.

MapReduce Parameters	Job Types in RL-MRCONF			Job Types in CDH		
	Search	Index	TeraSort	Grep	WordCount	TeraSort
map.tasks.maximum	7	3	2	7	3	2
reduce.tasks.maximum	1	4	4	1	4	4
mapreduce.task.io.sort.factor	10	40	20	10	40	20
mapreduce.task.io.sort.mb	80	80	100	80	80	100
mapreduce.map.sort.spill.percent	0.7	0.67	0.67	0.7	0.7	0.67
dfs.block.size (MB)	32	64	64	32	64	64
Best Runtime (sec.)	7.0632	7.3198	18.574	6.13	6.21	17.28
Initial Runtime (sec.)	24.468	16.916	47.178	21.57	15.43	46.03
Performance Improvement	71.13%	56.73%	60.63%	71.5%	59.75%	62.45%

The optimal configuration of the maximum number of mappers/reducers obtained by RL-MRCONF are specified as part of command line parameters when the job is initiated, while the rest of 4 chosen parameters are reconfigured by using Cloudera Manager before the job is submitted to the cluster. The jobs used in our simulation are Search, Index, and TeraSort, and the corresponding jobs used in CDH are *grep* (i.e., counts the matches of a regex in the input), *WordCount* (i.e., counts the works in the

input file), and *TeraSort* respectively, and the input data size is 1GB as used in the simulation.

For the studied jobs, as summarized in Table 8, it is clearly observed that most map tasks are data-local, and the simulation results show similar performance for these jobs as observed through the real cluster. The simulation also produces similar job performance under difference configurations. Overall, even at this granularity, the simulated results are quite similar to the actual results.

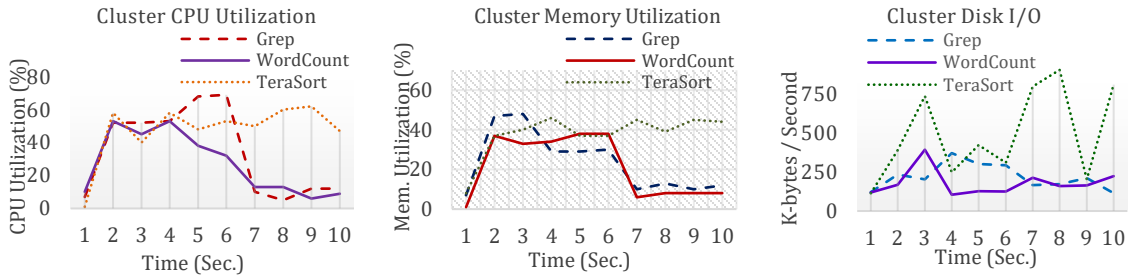


Figure 24. CHD cluster resources utilization matrix based on different jobs running on the cluster

The computing resources utilization characteristics of the studied jobs with the obtained optimal MapReduce configuration are depicted in Figure 24, where 24 (a), 11 (b), and 24 (c) represent the utilizations for CPU, Memory, and Disk I/O, respectively. We can observe that the computing resources were properly utilized during the execution of the jobs, and based on the results demonstrated in Figure 24, we can conclude that the WordCount job has the balanced utilization between CPU and memory, whereas the Terasort is I/O intensive and grep is CPU intensive.

Effectiveness of RL-MRCONF: In this section, we conduct more experiments and comparisons to verify the effectiveness of our proposed scheme RL-MRCONF. The verification process has two parts: one is to verify the generality of RL-MRCONF in terms of performance for similar jobs that differ in the input data size; another is the effectiveness comparison in terms of searching for the optimal configuration that yields the near optimal performance between RL-MRCONF and the previous studies.

To verify the effectiveness of RL-MRCONF adapts well to similar submitted jobs, we consider 3 different input data sizes as in 2GB, 4GB and 8GB respectively for

the submitted job types: Search, Index, and TeraSort. As depicted in Figure 25, we can see RL-MRCONF adapts well to similar submitted jobs with different in-put data sizes, the proposed scheme can achieve 40% ~ 45% on average of performance improvement in terms of job execution time when compared with the default configuration.

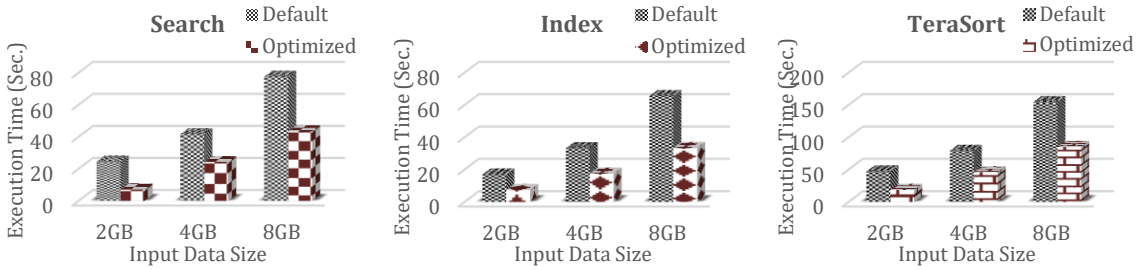


Figure 25. Adaptability of RL-MRCONF to similar submitted jobs that varies with input data size

Through this experiment, we notice the convergence time of our proposed heuristic increases with the size of the input data, along with the characteristics of the submitted jobs. This directly impacts the cost of our orchestration phase. However, when comparing with deploying the orchestration phase onto the real cluster and waiting for the results before initiating the production phase, the simulation-based approach can be deployed onto one or more independent machines that requires less computing resources to run the learning processes that are specifically targeted certain job profiles with various input data sizes. In addition, these learning processes can be executed in parallel with production phase and the learning agents keep collecting data over time. We believe this can reduce the time needed in orchestration phase and increase the viability of our proposed approach.

For a fair comparison with other schemes reported in the literature, we compare our RL-MRCONF against AROMA [153] and Starfish [154] to assess how much performance we can obtain with each approach. We performed the experiments using the studied jobs (i.e., Grep, WordCount, and TeraSort) with identical input data size as in 20GB. We compared the job execution time of our approach against the execution time achieved with the configuration recommended by AROMA and Starfish respectively. In addition, the absolute job execution time numbers are not directly comparable, and we

have normalized the job execution time with respect to the execution time obtained with the default settings in order to make fair comparison.

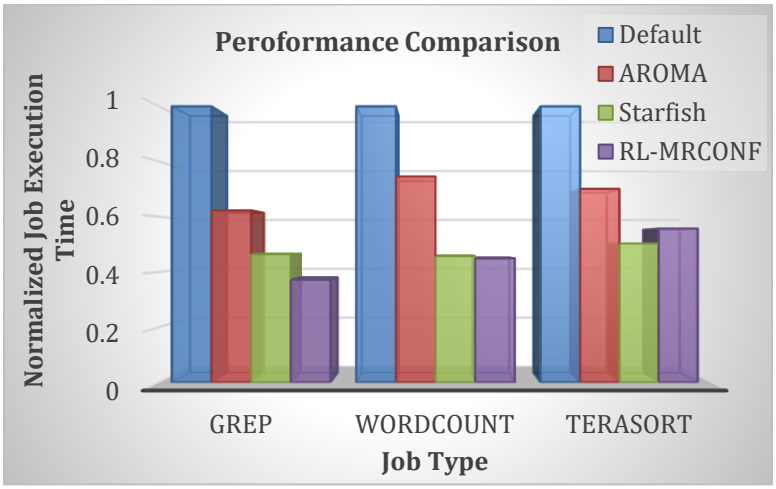


Figure 26. The normalized job execution time with respect to AROMA and Starfish for three studied jobs with input data size as 20GB

Figure 26 shows the results of our comparison. For the Grep, our RL-MRCONF has improved the job execution time by 26% and 15% compared with AROMA and Starfish, respectively. For the WordCount, our proposed scheme still performs better than AROMA and performs similarly with Starfish that improved the performance around 45%. Similarly, for the TeraSort, RL-MRCONF achieves better performance than AROMA, and exhibited relatively similar performance improvement with Starfish being slightly better than RL-MRCONF (around 7%). Since our proposed scheme has a smaller number (of 6 in total) of parameters than Starfish (which has 14), we expect that increasing the number of scheme input parameters will provide more flexibility for auto-tuning the parameters and further improve the scheme performance. Our results show that our RL-MRCONF achieves (when compared with other popular schemes) comparable or even better performance improvement in terms of job execution time in some cases.

3.8 Conclusion

In this chapter, we have presented a reinforcement learning scheme, named RL-MRCONF, to automatically configure the parameters of the MapReduce framework in a

Hadoop cluster. The majority of jobs submitted to a MapReduce process are repeated jobs or jobs with similar characteristics. Under such scenarios, an RL-MRCONF agent can be trained to automatically tune and adjust parameters to improve the overall system performance. Moreover, the training overhead can be reduced over time since this process is only performed during the initialization process of the agent. Our experiments demonstrate that the RL-MRCONF agent without initialization can be highly effective in finding configurations that yield better performance. We also showed the initialized RL-MRCONF agent is capable to find in a shorter time the near optimal. Moreover, we investigated the performance of the DQN-based agent. Experimental results of both trained/initialized DQN-based and RL-based agents show that the DQN-based learning agent outperforms the RL-based agent by 25% on average regardless of the job type or the configuration used to achieve optimal performance.

CHAPTER 4

SDN FLOW ENTRY MANAGEMENT USING REINFORCEMENT LEARNING

4.1 Abstract

The popularity of cloud computing services has evolve the way we implement modern information technology services. To facilitate cloud computing infrastructure, datacenter is built on top of high-speed Datacenter Networks (DCNs) to provide better flexibility and resiliency. Based on the dynamic nature of this network environment, the network traffic patterns can be further classified into long-lived (elephant) and short-lived (mice) flows with partitioned/aggregated traffic types. Despite the fact that SDN-based solution can dynamically allocate networking resources toward such flows, the network reconfiguration overhead can have the major impact on network performance. With this in mind, it is indisputable to decide which forwarding rules should remain in the flow table and which rules should be carried out by the SDN controller in the case of table-miss on the switch, especially when the limited capacity of Ternary Content Addressable Memory (TCAM) is installed in an OpenFlow enabled switch. To address this issue, we propose a machine learning scheme classified into two variations of Reinforcement Learning (RL) algorithms: one is traditional RL-based approach, and another one is deep reinforcement learning-based. With the fixed size of flow table of 4KB, the emulation results over RL algorithm demonstrate around 60% improvement in reducing long-term network reconfiguration overhead and improved the table-hit ratio around 14% when compared to the existing Multiple Bloom Filters (MBF) method.

4.2 Introduction

The demand for rapid big data analysis and fast computing power with large-scale datacenters has introduced the need for efficient network management systems. This trend inspired the current networking architecture to evolve toward Software-Defined Networking (SDN) [8], [26], a popular networking paradigm that enables the programmability of the network and decouples the packet forwarding plane from the control (decision making) plane. SDN advocates the openness of the programmable interface allowing network connectivity to be managed dynamically by network

applications with participating network elements, which enables the properties of "application-aware" and "network-aware" of the network.

At the time of writing this chapter, the dominating SDN protocol is OpenFlow [8], [14]. This protocol is utilized by controllers and switches of SDN environment to exchange information, which allows a switch to notify the controller of an incoming packet cannot be matched with the forwarding rules in the flow table. Analogously, the control message can be sent from the controller to a switch waiting for the instruction to add a new or modify an existing forwarding rule in the flow table to process incoming packets. This message exchange process is referred as *network control overhead* in this chapter. This overhead will gradually consumes networking resources and eventually decrease the performance of the packet forwarding process of the network.

To analyze the traffic pattern in datacenters, the flow of the traffic are further categorized into elephant flows (throughput-dependent), and mice flows (short-lived). Normally, web searches, social networking, etc. are considered as transactional applications executed by multiple end users that produces small flows need to be delivered in the timely manner. Alternatively, the applications like MapReduce, virtual machine migration, etc. that require bulk transfers with minimal packet delivery delays while consuming high network bandwidth. Based on the previous studies on datacenters traffic pattern [167], [168], the number of elephant flow is less than 10% among the traffic flows in the DCN, but the payloads contained by such flows consumes more than 80% of the entire traffic volume. Additionally, the short-lived mice flow might be harmless if considered individually, but mice flows account for 90% of the entire flows [167], [168], and they can have major impact on the performance of the network if they are not handled delicately.

The characteristics of the traffic pattern represented by such elephant and mice flows indicate a challenge in current DCNs as the large volume of mice flows can cause the forwarding rules of the elephant flow to be evicted prematurely from the flow tables, especially under the scenario of having a bust of mice flows arrive at a switch in between the ongoing elephant flow [169]. With the limited amount of memory capacity for storing forwarding rules (e.g., flow entries), the switch will not have sufficient space to accommodate additional flow entries for packets require new forwarding rules. This will

ultimately lead to packet drops and/or processing delays due to the increasing network control overhead.

The traffic patterns of DCNs are dynamic and heterogeneous depending on the services provided over their networking resources, it is extremely difficult to formulate a mathematical model to effectively and efficiently manage the flow entries with the capability to accommodate the pattern of elephant and mice flows. To mitigate this issue, we propose to utilize Reinforcement Learning (RL) algorithm to search for an optimized set of forwarding rules automatically to minimize the long-term network control overhead. The objective of this approach is to improve the long-term table miss event occurrences and packet processing latencies in considering the capacity of the Ternary Content-Addressable Memory (TCAM) equipped inside the SDN switches. The motivation of utilizing RL in our approach is based on the impressive capabilities of RL for exploring the unknown operating environments and improving its decision making using reinforcement through previous acquired experience in interacting with the environment. We believe that RL is well suited approach in achieving automatic self-adapting mechanism that enables the learning agent to explore the flow configurations space with the feedback provided by the environment based on the action performed by the agent. The objective of the learning agent is to construct the policy that maximizes the reward for the long-term. The RL is also the better fit than traditional optimization approaches (e.g., discrete optimization techniques like integer linear programming) since RL adapts in dynamic environments, whereas traditional optimization techniques only produce the optimum for a given snapshot of the problem scope.

Our proposed RL-based approach, as depicted in Figure 27, is deployed in the controller that utilizes two metrics to select the flow entries that should be installed on the switch: the flow match frequency and the flow recentness (duration). According to the OpenFlow specification [8], these two metrics are automatically recorded by the design of the protocol. The objective of the learning agent is to develop the policy that maximizes the long-term accumulative reward, which is taken to be inversely proportional to the network configuration overhead and the number of table-miss events. Based on this, the RL agent divides the pool of flow entries into two sets: the local switch entries and the remote controller entries. The purpose of this division is to reduce the

control plane overhead in considering the TCAM capacity of the SDN switches. Besides, our proposed RL approach is capable of adapting the aforementioned traffic pattern of the network.

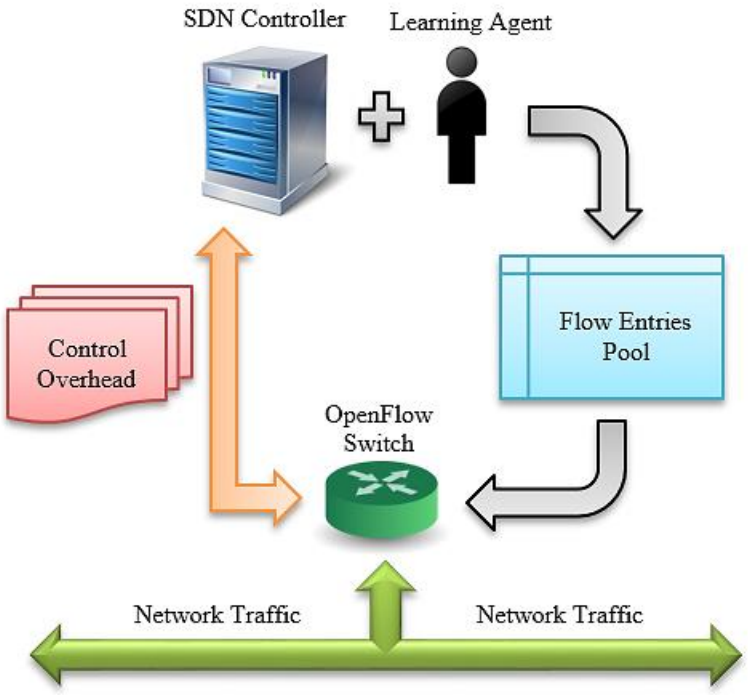


Figure 27. The overview of proposed RL-based approach for managing SDN flow entry

To determine the efficacy of our proposed RL-based approach in terms of reducing the overall control overhead via appropriate parameter selection for flow entries placement in TCAM, we have conducted a series of systematic effectiveness evaluation using Mininet [45], [46], a network emulator software that emulates the set of network elements. Our evaluation experiments are performed on two types of RL algorithms: traditional RL and deep RL. Both algorithms are utilized to drive the tuning of the parameter in the emulations to determine the proper needed actions in each state with the goal to maximize the accumulated log-term rewards. The emulation results demonstrate an improvement of around 60% in reducing the long-term control plane overhead and around 14% improvement in table-hit ratio compared with the existing Multiple Bloom Filters (MBF) method [170] given 4KB as the fixed flow table size. *To the best of our*

knowledge, this is the first research effort that has utilized RL for managing flow entries in SDNs.

The rest of the chapter is structured as follows. The research background and related works are discussed in section 4.3. In section 4.4, the design and implementation details are presented. The experiments, analysis, and evaluations are performed in section 4.5. Finally, we conclude this chapter in section 4.6.

4.3 Background and Related Work

In this section, the research background and related works are discussed. Particularly, we provide a brief discussion on the OpenFlow protocol, the characteristics and limitations of OpenFlow-enabled switch, TCAM utilization optimization methods, and RL applications in networking in the following subsections. We cover a discussion on these topics due to the essential understanding of the problem scope and its background. This work aims to divide the forwarding rules properly between the OpenFlow-enabled switches and the controller to further reduce the network configuration control overhead in SDNs.

OpenFlow Protocol: As stated in the chapter 2, OpenFlow is a communication protocol utilized between an SDN controller and OpenFlow-enabled switches. OpenFlow enables the direct access to the forwarding plane of network elements and their packets routing path programmatically. According to the specification of the OpenFlow switch supplied by ONF [8], one or more flow tables and a group table are involved when the packet forwarding process is initiated.

While OpenFlow is being utilized between the SDN controller and the OpenFlow-enabled switches as the communication protocol, the controller can add, update, and delete flow entries of the switch in two manners: 1). Reactive control through the processing of the incoming packets. 2). Proactive control via the built-in flow entry timeout mechanism [8]. A flow entry comprises of multiple components (match fields, priority, counters, instructions, timeouts, cookies, and flag) in performing the packet matching process. The uniqueness of the flow entry in each flow table is defined by the combination of match fields and priority.

When multiple flow tables are defined in the OpenFlow switch, the packet matching pipeline process may be initiated. When a packet arrives at the OpenFlow switch, the packet is processed by matching the flow entries with the defined priority, and then the instructions are applied to update the match fields, the action set, and the metadata (information used between tables) of the packet, and lastly by forwarding the matching data and the action set to the next table if required. In addition, when there is no flow entries can be matched with the packet, a table-miss event is triggered. The action associated with the table-miss event is defined by the table configuration describing how the unmatched packets can be processed, which includes the operations such as forwarding the packet to the controller, dropping the packet, or sending it to another table. Table-miss flow entries can be added in the flow table of the switch for executing appropriate table-miss actions in the future. The unmatched packets will be dropped if the table-miss flow entry does not exist in the flow table [8].

Characteristics and Limitations of OpenFlow Switch: Network policies are typically deployed in DCNs to provide quality network and infrastructure services to end users and cloud applications. These policies are implemented through a collection of forwarding rules, and the rule number can easily reach hundreds or thousands depending on the size of the SDN-based datacenter. As the result, it is important to design a better way to store those rules due to the limited memory capacity equipped inside the switch.

The SDN switch is generally using the embedded memory module in an *Application Specific Integrated Circuit* (ASIC) that executes a line-rate hardware-based packet processing. Typically, the memory module used by most common SDN switch is TCAM [171], which is considered as the extension of content-addressable memory (CAM). The primary difference between the most commonly used random access memory (RAM) and TCAM is that a data memory address is required for accessing data in TCAM. To access data in CAM, a query is employed to the data content itself for the entire memory in one clock cycle. The performance in data access process of CAM is better than RAM due to the nature of its data access parallelism and retrieval mechanism. Additionally, the term “ternary” in TCAM is described by the capability of using three different inputs (i.e., 0, 1, and X). This mechanism broadened the search patters in the entire memory, and this search pattern process is considered as the better fit in dealing

with networking applications based on its similarity of the process on how net-masks work, whereas the binary CAM module only operates exact pattern searches using only 0s and 1s.

With the capabilities provided by TCAM, it is commonly used in packet switching devices in order to achieve superior performance and perform additional wildcard matching process. However, high cost and high energy consumption are considered as the disadvantages of TCAM when compared with RAM (around $350 for 1 Mbits chip [172]), and the cost of a TCAM module is around 400 times more than a RAM module with the same capacity, the energy consumption is around 100 times of RAM [172]. Based on the current limitations of technological advancement, the size of TCAM is usually around 1 ~ 2 Mbits [173] and the TCAM capacity utilized by commercial grade networking devices is typically about 18 Mbits [172]. According to the information provided by OpenFlow specification 1.4 [8], 40 tuples are employed to form a single flow entry and 13 of them are essential tuples with 27 optional. In most cases, a 15 tuples flow entry occupies 356 bits [173] of memory. With a device installed with a TCAM memory module of 2 Mbits size, the device is anticipated to store around 6,200 flow entries. Depends on the scale of the network, the average number of flows arriving at a switch is estimated to be 75,000 to 1.3 million flows per minutes.

Under this scenario, an SDN controller can exceed the TCAM capacity limitation of the managed switches easily by installing a large volume of flow entries into their flow table(s). Additionally, the switches also needs to maintain policy/security associated rules such as firewalling, Access Control List (ACL), or rules defining routing mechanisms and traffic monitoring [179]. The OpenFlow switches will eventually stop accepting control messages from the controller once their limited TCAM capacity is reached. This issue can have major impact on packet switching performance of the switch since all new arriving packets need to be forwarded to the controller for further process if the packets cannot be matched locally by the switch. This posts a situation where the controller-to-switch communication volume is largely increased and further consumes unnecessary network resources on the control plane and delays the packet processing speed.

TCAM Capacity Utilization Optimization Approaches: Since the number of nodes connected to the network can have direct influence on the number of forwarding

rules required to provide the service, the current TCAM capacity cannot accommodate all entries for the large-scale networks with lots of network participants, and this has been proven in the work of [179] when OpenFlow-enabled switches are deployed in the production environments. To address this issue, several research efforts have been conducted and they are organized as follows:

Flow Table Compression: The flow table compression techniques are proposed by several research efforts [173], [174], [175]. In the work of [174], the authors proposed to utilize zero compaction method to reduce the routing table size for storing IPv6 addresses in two steps: 1) A *don't care* statement (x) is used to replace the number of zeros in the IPv6 address in the IP lookup table to reduce the size of IP address occupied within the table. 2) For the IP addresses having the same next hop IP addresses, the prefix overlapping approach is employed to store only one of those overlapping prefixes in the memory. The combination of zero compaction with prefix overlapping and prefix minimization techniques in Veeramani *et al.* [173] can reduce the table size by 50 ~ 60%. In Zhu *et al*. [175], the authors proposed a Multidimensional Table Compression (MDTC) method to represent a flow entry in the table by using the concept of a box, and a set of boxes comprise a multidimensional table. The small boxes are obtained by iteratively dividing this multidimensional space, and those boxes can be merged if they have the same action. The fundamental idea behind this approach is to use a larger box to contain multiple smaller boxes performing the same action. According to the work of [175], the MDTC can shrink entries of ACLs by 23% and the size of the generated OpenFlow tables by 2.4%. A two-level tagging approach, namely Tag-in-Tag, is proposed by the work of [173] to substitute flow entries with two layers of simpler and smaller tags to further reduce the size of flow entries. This approach consists of two types of tags: 1) The PATH TAG (PT) is used to associate a tag with a given path to route the packet. 2) The FLOW TAG (FT) is used to associate packets with a flow. Both PT and FT are utilize jointly to uniquely identify the flow without using the original flow entry tuples. The Tag-in-Tag method can store 15 times more entries in a fixed-size TCAM based on the work of [173].

Flow Entries Aggregation: The authors of [176] proposed to merge multiple flow entries by using two different schemes to minimize the flow table usage. One is the

offline scheme called Fast Flow Table Aggregation (FFTA) to reduce the flow table size; another one is online scheme called improved-FFTA (iFFTA) used to update the flow entries efficiently. For the design of FFTA, the fundamental concept is similar to the 3-step aggregation framework of bit weaving. First, the framework converts the forwarding rule list into the Binary Search Tree (BST)-based prefix-permutable partitions and then executes the Optimal Routing Table Constructor (ORTC) to aggregate flow table. Second, it performs the bit merging process that merges those rules that differ by a single bit or having the same action into one entry on each partition together iteratively to shrink the number of entries stored in the flow table. With the original flow table being partitioned and each partition is aggregated, the iFFTA can then perform flow entry updates efficiently by updating the affected partition in the modified-BST first, and then the bit merging process is re-executed for the associated rules.

Efficient Flow Table Management: Based on the specification of OpenFlow protocol, the flow entries can be evicted from the flow table proactively or reactively. The flow entry expiry mechanism is utilized in the proactive mode, which is defined by two attributes: *idle_timeout* and *hard_timeout* [177]. The *idle_timeout* is used to enforce the flow entry to be removed when the flow entry is not being matched for a given time frame. The *hard_timeout* is utilized to have flow entry evicted from the flow table regardless of its status. The removing of the flow entries by the switch itself is defined as the reactive mode. The works of Challa *et al*. [170] and Lu *et al*. [178] employ this default flow expiry mechanism to efficiently manage the flow table. In [170], the authors proposed an autonomous and smart flow eviction mechanism to remove the flows from the flow table using smart data logging methods with MBFs. The design of MBFs utilize the logical shift operations in column-major order to deduct an importance value of each flow, which is defined by two attributes of the flow: locality and recentness. The higher the importance, the higher the weight value of the flow, and the less important flow entries will be automatically purged from the switch to reduce switch-controller communication overhead. In [178], the authors proposed a method called TF-IdleTimeout to adjust the value of flow entry *idle_timeout* attribute dynamically based on real-time network traffic. This real-time network traffic evaluation is conducted from analyzing the

packet arrival intervals and further modifying flow entry *idle_timeout* to effectively improve the management of the flow table and improve the utilization of TCAM.

Flow Cache Mechanism: The authors of work [172] proposed a flow cache architecture that combines both of hardware and software switches as if they were a single switch with unlimited flow entry capacity. This approach employs one CacheMaster module to get the OpenFlow commands from the controller and distribute the flow entries to the managed switches. The delicacy of the forwarding rules placement lies in the algorithm used in [172] that utilizes the priority of the flow to arrange the order of the entries. In addition, each forwarding rule is associated with a match, action, and a weight value used to compute the traffic matching the rule. A list of prioritized *n* rules is compiled by the algorithm in order to maximize the weights sum of the rules placed in the TCAM. In the work of [169], Lee *et al*. proposed a shared cache architecture that uses a separate cache memory accessible by multiple switches, which is placed between the controller and the managed switches. If the scenario requires, the switch can first consult the flow cache memory instead of consulting the controller directly. The SHA hashing algorithm is used to make sure the flows are mapped to the proper shared cache of each switch in order to maintain the fairness between mice and elephant flows and also to protect elephant flows from premature eviction. To replace the cache flow in the shared cache memory, the authors design a localized Least Recently Used (LRU) algorithm to remove flows from the cache memory with specific constrain that only a mice flow can evict mice flow and an elephant flow can only evict an elephant flow.

Despite the dynamic management of the utilization of networking resources is promised by SDN-enabled environment, the efficient placement of the flow entries in the switch to reduce the overall information exchange between the controller and the switches still remains as the challenge. Besides, it is difficult to protect the TCAM capacity from being overused without the constant proactive monitoring performed by the controller, which indicates more control message exchange overhead is required by this process. On the other hand, our proposed approach can mitigate aforementioned issues using machine learning technique to automatically determine which forwarding rules should be placed inside the switch memory to reduce overall network configuration

overhead regardless of the traffic pattern in DCN. The technique does not require proactive inspection or monitoring of the incoming packets.

RL Network Applications: In this artificial intelligence era, the machine learning based approaches have motivated researchers to utilize similar techniques in innovating current networks. Particularly, the RL-based technique is considered as part of machine learning paradigm that allows the learning agent know how to choose the action it should perform in interacting with an unknown system to maximize the cumulative reward. The RL-based approach is getting more attention and it has been applied on multiple network applications [3], [4] (e.g., network routing [181]). In the RL-based technique, the repetitive learning process is performed by the agent who receives the feedback as the form of the reward from the dynamic system through the interaction with current system state, and then carry out the associated action based on its past experience to establish a policy to maximize the received cumulative rewards via transitioning of the system state in the long-term.

In the work of [182], Desai and Patil proposed a novel machine learning technique to determine the optimized route in ad hoc networks by modeling the swarm intelligence from the models of social insect behavior using cooperative RL. To evaluate the effectiveness of the proposed approach, the comparison with the existing routing protocols were conducted, and the results demonstrate the packet delivery ratio for RL-based approach is significantly improved.

Moreover, the authors of [183] presented the RL-based extensions of the existing Ad hoc On-Demand Distance Vector (AODV) routing protocol. The approach emphasized on improving route error tolerance by performing route reconstruction from the closest neighboring node instead of the source node in the case of a link failure. The simulation results imply the proposed approach is superior to the traditional AODV in terms of transmission delay and packet delivery ratio. Additionally, the results also indicate the improvement in reducing the route discovery frequency and further minimize the routing overhead of the wireless network.

The RL-based technique is also employed for improving the efficiency of the network routing in the work of [184] in the SDN domain. The Lin *et al*. utilizes a QoS-aware Adaptive Routing (QAR) in the three-level, multilayer, and hierarchical large-scale

SDNs involving different types of controllers. The design distribute the control loads among the controllers and is capable of reducing signaling delay remarkably. In combining with the characteristics of RL-based algorithm, their simulation results demonstrate that QAR performs better than the existing learning solutions and provides a faster convergence rate in QoS provisioning.

Additionally, the authors of [185] presented a deep RL algorithm, a Deep Q-Network (DQN) approach, to process large number of correlated channels in a Wireless Sensor Network (WSN). This work is similar to our proposed approach and the goal of this design is to maximize the long-term reward in obtaining the near optimal channel utilized for delivering the packets without failing. The simulation results of [185] show that DQN-based approach can accomplish the near-optimal performance on real complex scenarios that do not necessarily behave Markovian dynamics.

Table 9. The TCAM Capacity Utilization Optimization Approaches.

Approach Category	Summary
Flow Table Compression	• The authors of [173] proposed the use of two-level tagging approach to replace flow entries in order to reduce the size of the flow entries. • Veeramani *et al*. [174] proposed to use zero compaction method to reduce the size of the routing table in storing IPv6 addresses. • Zhu *et al*. [175] presented a "Multidimensional Table Compression" technique.
Flow Entries Aggregation	• The work of [176] designed a set of offline scheme (Fast Flow Table Aggregation) and an online scheme (improved-FFTA) to shrink the flow table and efficient update the flow entries respectively.
Flow Table Management	• Challa *et al*. [170] proposed an automatic and intelligent flow eviction technique to purge the entries using smart data logging with Multiple Bloom Filters (MBF). • The authors of [178] proposed the TF-IdleTimeout mechanism to adjust the value of the flow entry *idle_timeout* attribute dynamically based on the real-time network traffic.
Flow Cache Mechanism	• The authors in [172] presented a differential flow cache framework that achieves fairness and efficient cache utilization with fast lookup. • The work of [169] proposed an integrated hardware-software architecture namely "CacheFlow" with infinite rule capacity.

There are several research works that deployed RL over the network applications. However, none of these works apply RL in flow entry management in SDNs. To the best of our knowledge, our proposed approach is the first work that utilizes RL algorithms, traditional RL and deep RL (DQN), in managing SDN flow entries. In our scheme, DQN

is the fusion of traditional RL with neural networks, which has been proven to be capable of dealing with more complex systems with superior performance [157], [158], [161]. The summarization of aforementioned approaches and the existing research work that emphasizes the efficient utilization of TCAM capacity of OpenFlow-enabled switches is organized in Table 9.

Based on the discussed related works, Challa *et al*. [170] is similar to our proposed approach since both efforts utilize flow entry frequency and recentness in the algorithms, with the similar objective to reduce overall network configuration overhead while improving the table-hit ratio. The detailed study between our proposed RL-based scheme and the MBD-based approach [170] is discussed in section 4.5.

4.4 Design and Implementation

To properly construct and emulate the networking environment that is SDN-enabled, we utilize a software-based network emulator called "Mininet" [45], [46] that provides the capability to emulate a collection of endpoints (hosts), network intermediate devices (switches, routers, etc.), and links on top of a single Linux kernel. Mininet enables the entire OpenFlow-enabled network to be emulated using the lightweight process-based virtualization over a single computer. In this section, we introduce the learning framework firstly, then provide a detailed discussion on the design of our technique, and lastly discuss the details of the proposed algorithms.

The Design of the Learning Framework: For our specified approach, the property/configuration of the flow entries can be described by two deterministic parameters: *flow match frequency* and *flow recentness*. With this in mind, the RL algorithm can be built to acquire the configuration of the network that reduces the overall network control overhead. The *Markov Decision Process* (MDP) [3] is used to model our proposed algorithms that can be represented as the tuple <*S, A, P, R*>, where $S = \{s_1, s_2, \dots, s_i\}$ refers as the state space, $A = \{a_1, a_2, \dots, a_j\}$ refers as the action space, P represents the probability of the state transitions from state *i* to state *n*, and R refers the reward received associated with actions set $a \in A$. The goal of this MDP is to establish a policy $\pi : S \rightarrow A$ that maximizes the long-term cumulative rewards. Given the capabilities provided by SDN, we are able to obtain all the forwarding rules needed for

all traffic flows traversed on the network in order to mitigate the problem we address in this work. For the scope of this research, the state space, action space, and immediate reward function are defined as follows:

- ***State Space***: The state space in our model represents all the possible combinations of finite values of flow match frequency and the flow recentness parameters. For the state of the chosen two decisive parameters, we denote a state as follows:

  $$s_i = (flow\_freq_i, flow\_recentness_i)$$

  where $flow\_freq_i$ is denoted as the matched frequency of the flow, and $flow\_recentness_i$ represents the duration of the given flow entry resides in the switch's memory.

- ***Action Space***: The action space in our model consists of the following: (1) *No action*; (2) *increase*; and (3) *decrease* action associated with each chosen parameter. For example, we denote the decrease action on the given flow frequency parameter as follows:

  $$a^{increase}_{flow\_freq_i} = (flow_{freq}{}^{increase}_{i}, flow\_recentness_i)$$

- ***Immediate Reward Function***: The network control/configuration overhead is defined as the number of communications initiated between the controller and the switches in order to process the incoming packets properly. The immediate reward is determined based on the measured configuration overhead given the selected flow entries are placed in the switch. The immediate reward is represented as follows:

  $$r_t = Compare(overhead_{current_{best}}, overhead_t)$$

  where $overhead_{current\_best}$ is the current best network control overhead obtained so far by the emulation. For a given comparison, a less overhead returns a positive reward +1 to the agent; otherwise, the agent is given a negative value -1 as the reward. If $overhead_t$ equals $overhead_{current\_best}$ then a 0 reward is given.

  In our proposed technique, we deploy the following RL algorithms:

Traditional RL: The *temporal difference* (TD) Q-Learning algorithm is used in our design for the emulation experiments. The reason behind this decision for utilizing Q-Learning algorithm is inspired by the characteristic that it does not need a model for the given environment, and the algorithm keeps updating the Q-values in each learning step based on the estimation of the calculated action-values function. The iterative nature of this learning process is performed in discrete time steps where the learning agent can interact with the environment. In each step, the agent choose an action $a_t \in A$ in the given state s_t. Specifically, the agent makes its own decisions to select an action to perform based on the action selection policy in order to achieve the goal of maximizing the obtained reward. The *Q(s, a)* represents the average Q-value of an action *a* at state *t*, whereas the immediate reward is received after the chosen action has been performed to alter the state. The formal notation of this Q-Learning algorithm is given in the Equation (1) of the chapter 3. In this research work, we use a learning rate of 0.1 and a discount factor of 0.95 as they are commonly used in practice as suggested by [3].

Algorithm 3: Q-Learning Algorithm

Pre-condition:
Initialize Q table with small random number
Initialize state s_t
Initialize goal μ
Procedure:
improvement = 0
repeat
  for (step = 0; step < learning_iteration; step++)
    Get action a_t from s_t using $\epsilon - greedy$ policy
    Get parameter $param_t$ from s_t using $\epsilon - greedy$ policy
    $\epsilon = \epsilon$ – (step / learning_iter) * ϵ
    Take action a_t on the $param_t$ and receive reward *r*, obtain control overhead *c*
    Sample new state s_{t+1} after applied action a_t
    Update $Q_t \leftarrow Q_t + \alpha * (r + \gamma * \max Q_{t+1} - Q_t)$
    Update the corresponding $param_t$ of Q_t
    improvement = get_improvement($best_t, worst_t$)
    $s_t = s_{t+1}, a_{t+1}$ = Get action from $s_t, a_t = a_{t+1}$
  end for
until *improvement* > μ

Similar to our previous work in chapter 3, the *ϵ-greedy* policy is also utilized as the action selection policy where the action with the highest Q-value is always picked with a small probability that agent will pick other action at random. This indicates that the learning agent is provided with the opportunity to explore the action space for

obtaining the better action that generates the better reward. Moreover, the algorithm also records the selected parameter associated Q-value of each action. If the action is selected by the agent based on its Q-value, the corresponding parameter value will also be updated. The complete Q-Learning algorithm utilized in our approach is presented in Algorithm 3.

Deep RL (DQN): The Deep Q-Network is a deep RL algorithm that integrates the capabilities of Artificial Neural Networks (ANN) known as Deep Neural Networks (DNNs) [157]. Neural networks are used to approximate functions by learning a large collection of input data, the term "*deep neural network*" is represented by a neural network with *n* number of hidden layers [5]. Similar to aforementioned algorithm in chapter 3, the feedforward neural network is employed by DQN that comprises three components: 1) the interconnected *neurons* using directed links to constitute the network, 2) the connection associated *weights* value, and 3) *layers* contain a number of neurons (including hidden layers).

As we have discussed in chapter 3, a notable property of DNNs is that the number hidden layers and the number of neurons in each layer are free parameters. This means that there is no predefined or default value for these parameters. The larger/deeper the neural network, the more complex application it can model. Besides, an activation function is used to calculate the activation value and propagate it between connected neurons. The numeric weight value associated with each connection between two neurons is used to determine the weight of the link. To calculate the activation value a_j from neuron *i* to neuron *j*, the *j* neuron needs to compute a weighted sum of all of its inputs. This activation function is presented in the Equation (2) in chapter 3.

Additionally, there are several flavors of activation functions available based on the users' preference. Different types of activation functions have different ways to calculate the activation value. Among all the available activation functions, ReLU is gaining the popularity in deep neural networks due to its faster and more efficient learning in terms of reducing the likelihood of vanishing gradient problem, and sparsity in activation [5]. The ReLU activation function is denoted in the Equation (3) of chapter 3.

In our research, it is inefficient to process all the possible state in the state space since some of them are rarely selected and further cause longer convergence times for the Q-table. Thus, the approximation (prediction) of the Q-values for those extreme states is considered as the better way to handle those states instead of actually processing them. The prediction process is performed based on previous processed states (i.e., during the training phase), and this is done by combining our original Q-function with a DNN that takes the state, which consists of flow match frequency, recentness, chosen action-parameter pair, and the received reward as the input, and produce the Q-values for all possible actions and its associated chosen parameter. The complete DQN update process can be formulated in the Equation (4) in chapter 3.

Algorithm 4: Deep Q-Network Algorithm

Pre-condition:
Initialize expereince memory *M*
Initialize action-value pair *Q* with random weights
Initialize state s_t
Initialize goal μ
Procedure:
improvement = 0
repeat
  for (step = 0; step < learning_iteration; step++)
    Get action a_t from s_t using $\epsilon - greedy$ policy
    Get parameter $param_t$ from s_t using $\epsilon - greedy$ policy
    $\epsilon = \epsilon$ – (step / learning_iter) * ϵ
    Take action a_t on $param_t$ and receive reward *r*, obtain control overhead *c*
    Observe new state s_{t+1}
    Store experience memory (s, a, r, s_{t+1}) into *M*
    Sample *n* random transitions (s', a', r', s'') from *M*
    Update $transition\ tt \leftarrow r' + \gamma * \max(s'' - a'')$
    Update the $param_t$ of θ_i
    Train the Q network using *loss* = $(tt - Q(s', a'))^2$
    improvement = get_improvement($best_t, worst_t$)
  end for
until *improvement* > μ

Similar to the traditional RL, the ϵ-*greedy* policy is also used to select the action to perform based on the highest Q-value associated with that action. The initial ϵ value is set to 1 at the beginning so that the agent can choose random actions. This value is then decreased over time in order to maintain a fixed exploration rate. In DQN, the algorithm will also keep track the Q-value of each action and its associated chosen parameter. The detail of the DQN algorithm is represented in Algorithm 4, and the terminal objective of the algorithm is denoted as μ.

Problem Formulation: We can describe the process of finding the best parameter set that reduces the overall control overhead in our approach is the *set partition problem*, and this problem is known to be NP-Hard. The problem emphasized in this research can be formally defined as an Integer Liner Programing (ILP) problem. However, due to the scale of the problem space, the ILP problems are considered troublesome for large-sized problems, we will use an RL-based solution as the alternative, which provides better efficiency for dealing with larger networks.

Let *T* denoted as a multiset where $T = \{t_1, t_2, \dots, t_n\}$, and the element t_i is the rate of overhead traffic generated by the control plane of the network when the *i*th flow entry is installed on the SDN controller. Alternatively, when the flow entry is installed on the switch, the flow traffic rate generated on the control plane is zero. In addition, let *S* denoted as the multiset where $S = \{s_1, s_2, \dots, s_n\}$, and the element s_i represents the size of the *i*th flow entry. Lastly, we denote a_i^c and a_i^s as the selectors where the value 1 will be given if the *i*th flow entry is assigned to the SDN switch or controller, and the value of 0 will be given otherwise. Therefore, the problem can be formulated as follows:

$$Minimize \sum_{i=1}^{N} t_i * a_i^c \tag{5}$$

Such that

$$\sum_{i=1}^{N} s_i * a_i^s \leq C_{TCAM} \tag{6}$$

$$a_i^c + a_i^s = 1 \; \forall_i \; 1 \; \leq i \leq N \tag{7}$$

$$a_i^c \in \{0, 1\} \; \forall_i \leq i \leq N \tag{8}$$

$$a_i^s \in \{0, 1\} \; \forall_i \leq i \leq N \tag{9}$$

where C_{TCAM} refers the TCAM capacity of the SDN switch and *N* refers the number of flow entries that require to be placed on the switch or controller. The objective of Equation (5) focuses on reducing the rate of overhead traffic generated on the control plane of the network. Equation (6) defines the constraint to ensure the total size of the

flow entries placed on the switch does not overwhelm the capacity of the TCAM installed on the switch. In Equation (7), the constraint is used to enforce that each flow entry is placed on the switch or the controller, and further ensure that no flow entry is left unused or duplicated. Finally, the rule for decision variables can only use integer values of a 0 or 1 is enforced by Equations (8) and (9).

Caffe – The Deep Learning Framework: In our design, the open source deep learning framework namely "Caffe" [159] is employed. Caffe is an open source framework developed and maintained by *Berkeley Vision and Learning Center* (BVLC) using the C++ programming language. The framework provides a toolset with key functionality of deep learning algorithms to users. Caffe typically runs with CUDA technology so that the execution and computation of the algorithm can be further boosted by using GPU(s) and it also supports Python and MATLAB bindings. The main components of the Caffe framework are as follows:

- ***Blobs***: A 4-dimentional array used by Caffe as a unified memory interface for holding data such as batches of images, model parameters, and so on. Blobs conceal the underlying operations in CPU/GPU by synchronizing the information between them as needed. For instance, the user can use the CPU to load the data into blob from the file system, and invoke the CUDA kernel to perform GPU calculation without knowing the low-level details.
- ***Layers***: Similar to a neural network layer, a Caffe layer plays an important role in the deep learning algorithm that takes one or more blobs as input, and produces one or more blobs as output. There are two major functions that need to be performed within the Caffe layer: (1) the *forward pass* that produces output based on the input fed into the layer; (2) the *backward pass* that takes the gradients of the output to calculate the parameters along with the gradients of the inputs and propagates them back to the previous layer using the backpropagation process. Caffe provides several types of layers including convolutional, pooling, and more. More details about Caffe can be found in [159].
- ***Network***: Caffe network represents a set of connected layers in the form of a directed acyclic graph that ensures the correctness of the network being constructed in terms of forward and backward pass operations. The network starts

by loading the data from the disk into the data layer and generates the results in the loss layer that can be used in classification or pattern extraction applications.

Caffe library is utilized for our deep reinforcement learning algorithm. The deep Q-network algorithm development is based on the work of Mnih *et al*. [157], [158] with some modifications tailored for our research focus.

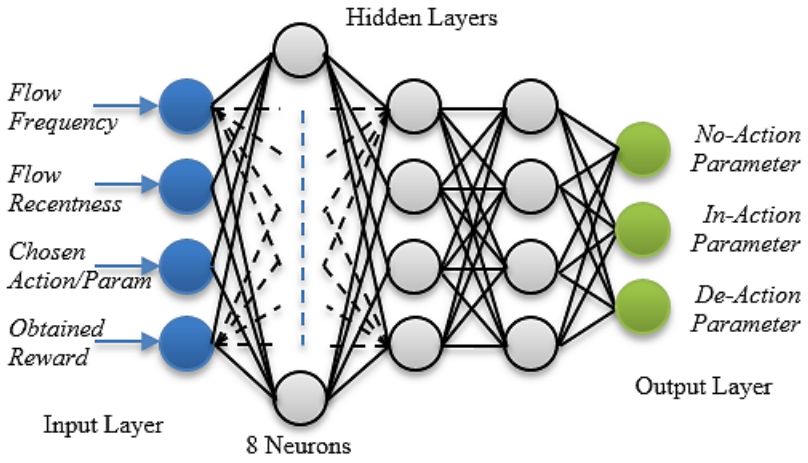


Figure 28. The deep RL neural network achitecture

The architecture of our deep learning network, as illustrated in Figure 28, consists of one input layer that takes as input high-dimensional data (i.e., the state space), three fully connected hidden layers, and a loss layer that produces output. The output in our case maps each configuration state to one of the three actions: no action, increase, and decrease operation, with its corresponding parameters (i.e., flow match frequency or flow recentness) with its associated Q-value. The training of the network starts by loading all the states into the input layer, passing it through the layers of the network, and feeding the final prediction into the loss layer that yields the loss and gradients. These are further used to train the entire network through the backpropagation process.

Generation of the Traffic Flow: One of the essential element of our emulation experiments is the flows. According to the previously discussed flow patterns, the flows can be further divided into two categories: the elephant flows that convey high volumes of data and the mice flows that carry small amounts of data generated from several endpoints/applications. Typically, these two types of flows running on top of the network has the ratio 1:9, where elephant flows account 10% and mice flows are 90%. Based on

the work of [181], the size of a mice flow is around 256 KB in average, whereas the size of an elephant flow is considered as tens of megabytes. In our emulation experiments, the values of 25.6 MB and 256 KB were employed to represent elephant and mice flows, respectively, and all the flows are conveyed over the 1 Gb/s links using TCP/IP protocol.

Additionally, the Mininet emulator supports the capability to execute Python codes within each emulated endpoint, which can run the script discretely or simultaneously to generate TCP/IP traffic to the predefined destination with the specified port number programmatically. The flow destination used by each endpoint is defined based on the passed-in arguments for the purpose of the script reusability. Moreover, Mininet is equipped with the built-in "*iperf*" tool that is also utilized collectively with our Python script to create randomly generated traffic patterns to simulate the real traffic in DCN. The traffic model used in our experiments assumes a greedy source producing Poisson traffic at an average rate of 310 Mbits/sec.

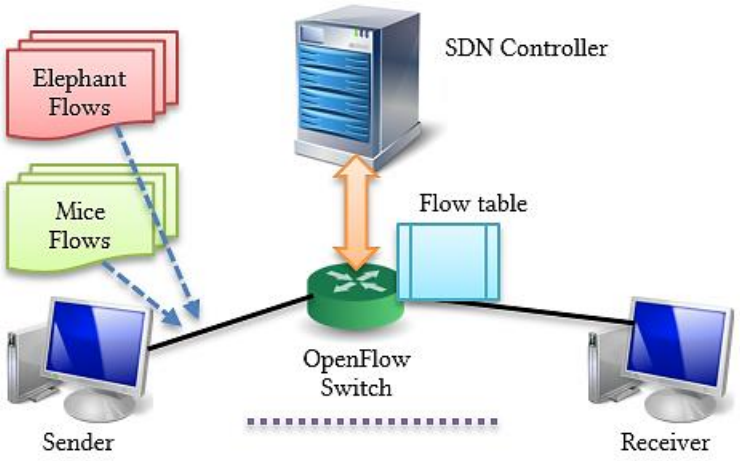


Figure 29. The Mininet emulation topology

Design of Mininet Emulation: In order to simplify the problem scope of our research, we designed a simplified SDN topology that utilizes one SDN controller and one OpenFlow switch with 20 connected endpoints. This network topology is capable of generating more than 4,000 flow entries to be resided in the OpenFlow switch and this is no doubt that those entries can exceed the 1 Mbits TCAM module. For the role assignment of those emulated 20 network participants, some of them will send traffic while others act as receivers, as illustrated in Figure 29. The network configuration

overhead is defined by number of times the controller exchanges messages with a switch in order to properly process the incoming packets to its destination.

The POX [37] SDN controller is employed in our emulation experiments. As we have discussed in chapter 2, POX is am open source, Python-based networking software platform that supports OpenFlow protocol version 1.0 and provides abundant APIs that reveal the capabilities of POX. To utilize POX in our emulation and ensure the full compatibility, we developed a customized POX component that can be used by the POX framework and further extended the following built-in event handlers of POX:

- ***ConnectionUp***: This event is triggered by the establishment of a new control connection with a switch. We extend the functionality of this event handler by implementing additional Python code to install the selected flow entries generated by learning agent before the processing of any incoming traffic.
- ***ConnectionDown***: Similar to the *ConnectionUp* event handler, this event is triggered when a channel to a switch is dropped, either explicitly from controller or the reboot of the switch. We extend this handler to output the final value of the measured control overhead during the emulation session, and the value can then be used later to calculate the immediate reward received by the learning agent for that specific session.
- ***PacketIn***: This event is fired whenever an OpenFlow packet-in message (sent by the switch) is received by the controller. This event refers to a packet has arrived at the switch port and can represent two scenarios: 1) the switch does not know how to process this packet (i.e., it failed to match all the flow entries in the table), or 2) the matched flow entry contains the instruction to send the packet to the controller. This event is where the controller calculates the overhead by incrementing the overhead value when the switch forwards a packet to the controller.

During the initial phase of each emulation experiment, the traffic is injected onto the network, and the state of the flow table is logged and monitored. To reduce the learning time of the agent, the deployment of our proposed technique can be divided into two phases: learning engine orchestration phase and production phase. We initiate the state space construction process in the orchestration phase to collect all the forwarding

rules needed to properly handle all the incoming traffic on the switch. During this flow entries collection process, the pool of flows is built on the controller side to observe all the unique flow entries for the given time interval after the traffic has been placed onto the network. We notice that the orchestration phase has direct impact on the efficiency and performance of the production phase due to the fact that network traffic patterns vary with time of the day and with the applications consuming the network resources. The implementation of the intelligent state space construction process that adapts to the dynamic nature of the network during the orchestration phase will be carefully studied in our ongoing research. To set the time limit of the data collection process in our experiments, we define this constraint to be 5 minutes.

When the agent is deployed on the controller during the production phase, the agent is expected to explore the state space and automatically search for the near optimal network flow entries from the flow pool by using two decisive criteria: flow match frequency and flow recentness. The ultimate goal of the agent is to determine the flow entry configurations that reduce the overall control plane overhead, and further yields a configuration that can accommodate the traffic patterns monitored on the network.

The detail workflow of our proposed approach, as illustrated in Figure 30, can be organized into series of procedures as follows:

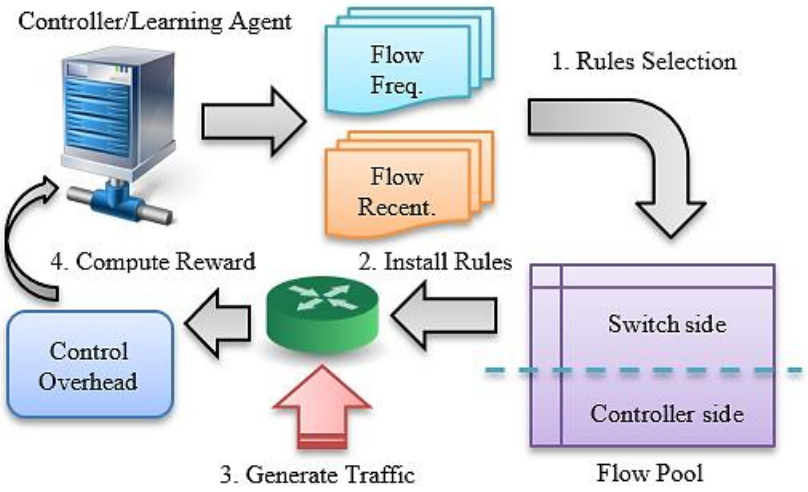


Figure 30. The workflow of the proposed technique

1) Depends on the decision made by the learning agent, the value of *flow_freq* and *flow_recentness* are used to filter all the flow entries from the flow pool and select all the candidate entries that satisfy those two criteria.
2) The selected flow entries are placed in the flow table of the switch before the traffic is pushed into the network.
3) The Mininet emulation session is initiated and the control plane overhead is obtained at the end of the emulation process.
4) The reward is calculated by the algorithm for the current flow entry configuration and the Q-values with its corresponding parameters are updated.
5) The aforementioned steps are repeated until the goal is met.

Whenever the near optimal parameters for choosing the flow entries that need to reside on the switch memory is obtained by the agent, the overall network control plane overhead can be reduced in the long-term while maintaining sufficient capacity for the TCAM module.

4.5 Experiment Analysis and Performance Evaluation

To present the effectiveness and performance of our proposed RL-based technique, the systematic evaluation experiments are conducted to compare the traditional RL and deep RL algorithms against a baseline that deployed the MBF method proposed in the work of [170]. Our proposed RL-based learning algorithm is added to the SDN controller as an extra layer of intelligence and the performance matric utilized in our study is the control plane overhead generated by each emulation experiment.

To setup the proper experiment environment, we use two Mininet-enabled Virtual Machines (VMs) operating over a *VirtualBox* hypervisor. One VM is implemented with traditional RL agent while another is DQN agent. The environment is equipped with Python version 2.7 with Ubuntu Linux LTS 14.04, and the Mininet is in version 2.2.1.

Performance of Traditional RL: The effectiveness of the RL-based approach is discussed in this section. The goal of the learning agent is to build a policy that minimizes the overall control plane overhead while improving the switch memory utilization efficiency.

In our first experiment, the performance of the traditional RL on two different sets of decision parameters on the light weight traffic network is studied. The first set, as depicted in Figure 31(a), contains the flow entries that satisfy flow match frequency as 90, and flow recentness as 30. In other words, this indicates any flow entries with the match frequency are more than or equal to 90 or recentness are less than or equal to 30 seconds will be selected to be installed into the flow table of the switch. The second set, as illustrated in Figure 31(b), has the flow entries the satisfy 40 as the match frequency and 50 as the flow recentness. The purpose of this experiment is to evaluate the adaptability of the traditional RL agent to different sets of parameter values. As we can observe in Figure 31, the emulation results show the control plane overhead is gradually reduced over time for those two different sets of parameters.

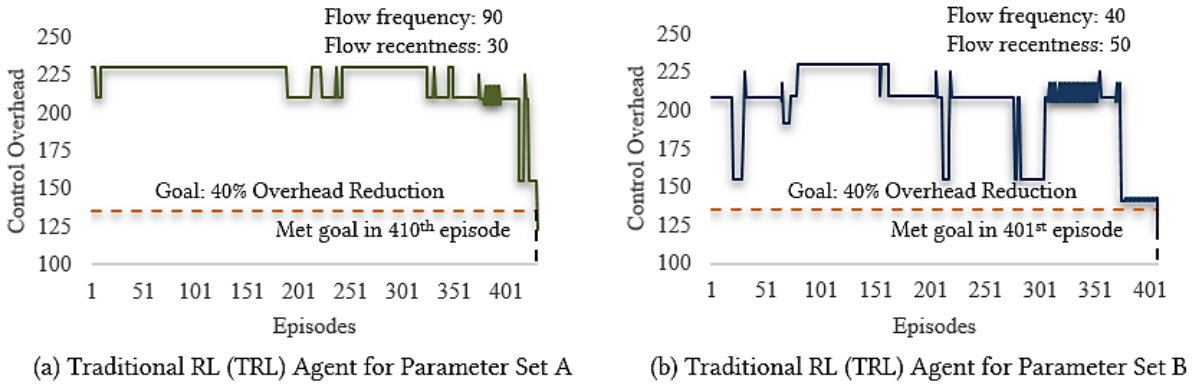


Figure 31. The traditional RL agent with different parameters set

The goal of this experiment for the traditional RL agent is to obtain the set of parameter that can be used to select the flow entries that can at least reduce the 40% of the network control plane overhead. The results indicate the control plane overhead has been reduced at least 40% when compared with the initial state of the configuration. The 40% here is the predefined objective set for this experiment, and other goals can be chosen as well depending on the requirement of the application. If the predefined goal is unreasonable or infeasible, the learning agent will run indefinitely as it will continue to strive to accomplish the given goal.

In observing the experiment results, we have noticed that the obtained control overhead increases toward the end of learning episode. As illustrated in Figure 31(b), the

obtained control overhead starts to deteriorate after 300th episode for the learning agent. It is due to the property of the $\epsilon - greedy$ policy utilized in the learning process that allows the agent to pick some other action at random regardless of its Q-value. Based on the design of the algorithm, the exploration rate ϵ decreases as learning episode increases, but the learning agent still has the possibility to make the decision that deviates from the known best course.

The Effectiveness of the Initialized Q-table: The size of the state space of the RL algorithm is defined by the number of configurable parameters and the range of values in each parameter. The state space grows exponentially as number of parameters and the range of values increase. Due to the nature of the RL algorithm, the RL agent performs a large number of explorations before it reaches the goal state. We believe this significantly degrades the performance of the RL algorithm.

With this in mind, we observed the impact of the initialized Q-table (i.e., initialized policy) on improving the performance of our proposed approach. The motivation behind this experiment is that in a typical setting, the networks experience similar traffic patterns repeatedly. Hence, under this assumption, we argue that our RL agent can achieve the objective faster, and minimize the time and computing resources needed in order to satisfy the goal. We again randomly selected two different parameters sets that do not belong to the set used to initialize the Q-table in order to evaluate the impact of the initialized policy in our second experiment, and the predefined goal of this experiment is 40% reduction in the control plane overhead.

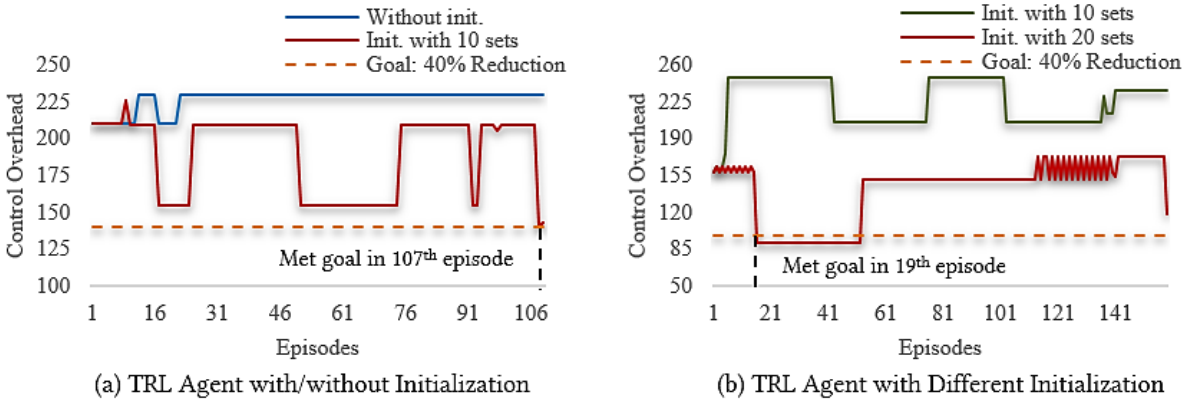


Figure 32. The traditional RL agent with and without initialized Q-table

The performance of the traditional RL agent with and without initialized Q-table is represented in Figure 32. As illustrated in Figure 32(a), the results show how the initialized policy can guide the agent to achieve at least 31% reduction in control overhead on the 17th learning episode. Besides, the agent can also achieve the predefined goal, which is 40% reduction on the control plane overhead, on the 106th episode. Alternatively, the agent without the initialized policy could not obtain the configuration that meets the goal within a finite 150 learning episodes, and further prolonging the convergence time of the agent.

In addition, the influence of the quality of the initialized Q-table on searching for the configuration that would produce the optimal network control overhead in terms of the number of the episodes needed is also studied in this experiment. Figure 32(b) represent the results when the Q-table is initialized with 10 and 20 different parameters sets. We observe that the agent with Q-table initialized with 20 different sets can achieve around 44% control plane overhead reduction on as early as 18th episode. This implies the more training set used to initialize the Q-table, the better performance that the agent can have in terms of convergence time. In this experiment, we used a static exploration rate as 0.1 to make the agent perform less exploration, but more exploitation on the knowledge enclosed in the Q-table.

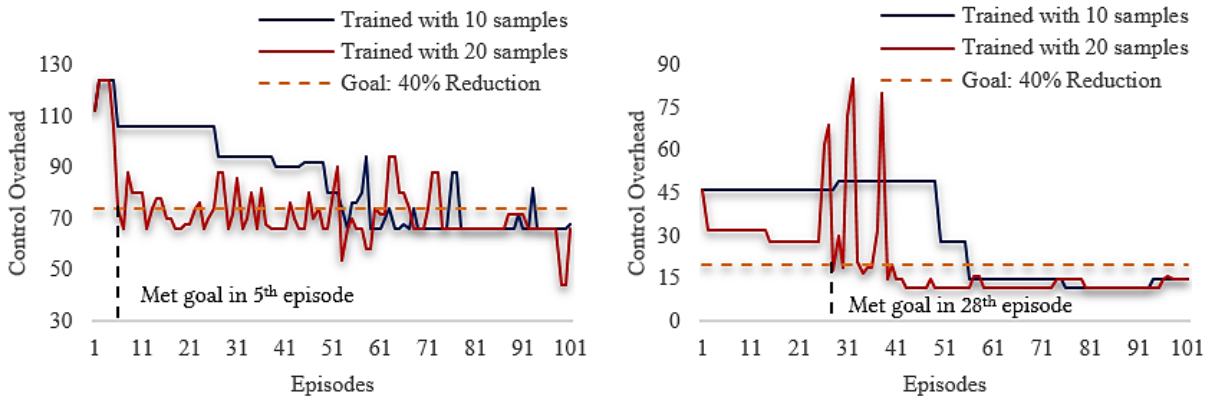
Performance of Deep RL (DQN): In this section, we conduct experiments to study the effectiveness of deep reinforcement learning in support of searching a configuration set that minimizes the overall network control plane overhead. Our emulations utilize the Caffe library to develop a deep Q-network. Similar to our RL-based scheme, the DQN-based RL agent explores the parameters sets based on what it has learned during the training process. Instead of evaluating all the actions individually on a given configuration from the previous state as implemented in the traditional RL agent, the DQN estimates the Q-values for all the actions for the given configuration produced by applying the chosen action on the configuration from the previous state.

Before the DQN learning process is initiated, the fundamental neural network is built with four neurons in the input layer (two decisive parameters, the selected action, and the immediate reward) and three neurons in the output layer used to represent each

action with associated Q-value and parameter. Moreover, three hidden layers are utilized in our approach as depicted in Figure 28.

During the training phase, the *Stochastic Gradient Descent* (SGD) method is used to obtain the minimum loss value. A vector of memory is used to collect all the processed states associated with the action performed and the parameter selected by the DQN learning agent. These processed states are referred to as the *state transitions* in our approach. When the size of the memory vector reaches a certain threshold, the algorithm updates the Q-network by randomly selecting four transitions and uses them to calculate the weights in the network. The update process lasts until the agent reaches the goal state. In this experiment, the goal state requires 40% control plane overhead reduction compared to the initial state.

To assess the performance of our proposed DQN technique, we designed two training scenarios to evaluate the performance of the DQN agent trained with 10 and 20 uniformly selected random parameters sets. The same light weigh traffic used to evaluate the performance of the traditional RL agent is also utilized in this experiment.



(a) DQN-based RL Agent for Set A with 40% Goal

(b) DQN-based RL Agent for Set B with 40% Goal

Figure 33. The DQN-based RL agent with different training scenarios

Based on Figure 33, we can learn that the agent trained with 20 randomly chosen sets is superior to the one trained with 10 sets. By observing the results, the goal state was achieved within 5 learning episodes while another scenario needs 28 or more learning episodes to reach the objective as represented in Figure 33(b). This implies the

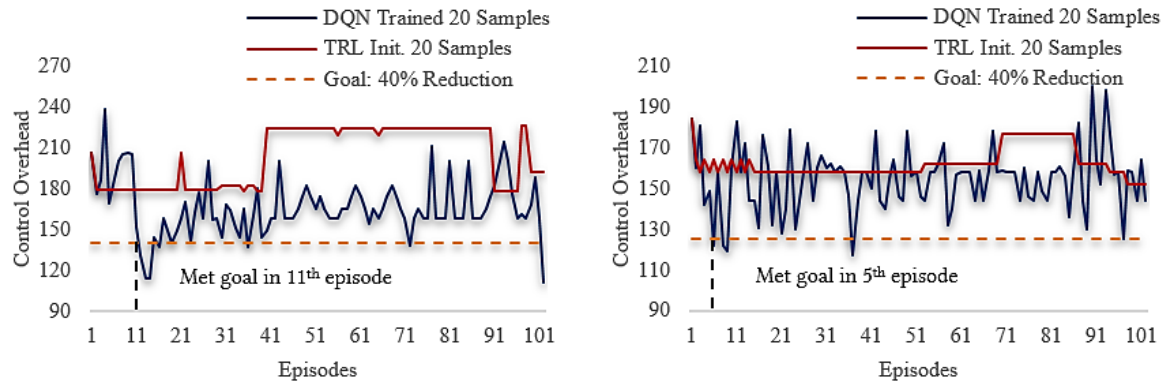
performance of the DQN agent improves with number of training samples utilized to train the network. During this evaluation process, the similar setting of the traditional RL experiments is also employed where a static exploration rate as 0.1 is used to let the agent perform less exploration and more exploitation of its learned experience.

Performance Comparison between the RL agents: In this section, we conducted multiple experiments to compare the performance of the traditional RL and DQN agent. To ensure the fairness of the comparison, we implemented a training/initialization scenario with 20 uniformly randomly chosen parameters sets. The selected candidate sets used to evaluate the learning agent are not part of the sets used to train the agents. Different from out previous experiments, the light weight traffic pattern is used to train/initialize the Q-matrix, and we utilize another heavier traffic pattern to initiate the evaluation process. In addition, we have predefined two objectives, 40% and 60%, to evaluate the effectiveness for reducing the control plane overhead of the two learning agents.

Moreover, we are interested in how the experienced agents interact with the unknown traffic patterns and the experiments started with the initial parameters set that are not part of the training sets. The decision making process of the agents in dealing with this unknown pattern completely relies on their past experience and the knowledge gained during the training phase. The initial state used to begin the learning emulation consists of randomly chosen parameters sets and they are enforced to be independent from the training sets. The objectives of this evaluation experiments are 40% and 60% reduction on the control plane overhead, respectively.

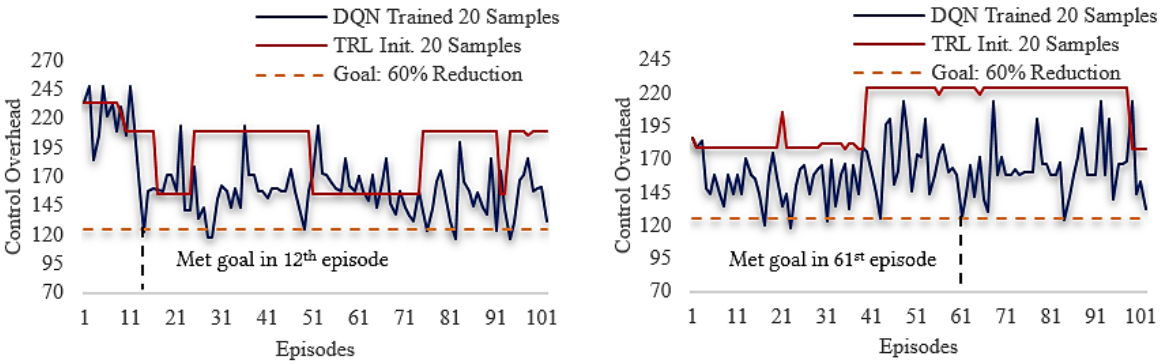
As depicted in Figure 34, we observe that the performance of the DQN learning agent is always superior to the traditional RL agent when the goal state is set to 40% reduction on control plane overhead. Based on Figure 34(a) and 34(b), we can also learn that DQN agent is able to obtain the better parameters set using shorter learning episode. Additionally, as illustrated in Figure 35(a) and 35(b), the DQN learning agent continued outperform the traditional RL agent when the goal state is set to 60% overhead reduction. Therefore, based on the results of the experiments, we can conclude that DQN learning agent has better efficiency and performance than the traditional RL agent when both

agents were trained using the same parameters sets, initialized similarly, and has the identical predefined goal state.



(a) Comparison for Set A with 20 Samples, 40% Goal (b) Comparison for Set B with 20 Samples, 40% Goal

Figure 34. The comparison between learning agents trained with the same 20 samples with goal state in 40% control plane overhead reduction



(a) Comparison for Set A with 20 Samples, 60% Goal (b) Comparison for Set B with 20 Samples, 60% Goal

Figure 35. The comparison between learning agents trained with the same 20 samples with goal state in 60% control plane overhead reduction

From the results shown in Figures 34 and 35, it can also be noticed that the traditional RL agent could not converge to yield the parameters set to satisfy the goal state within the finite learning episode (i.e., 100 episodes), whereas the DQN agent is capable to achieve the goal in shorter episodes. To clarify the purpose of the experiments, the agent can stop the learning process whenever it satisfies the objective, and we make the agents run for 100 episodes for demonstrating the concept of the experiments. Statistically speaking, the average control plane overhead obtained by the DQN agent is

close to where the goal was defined. In addition, the DQN learning agent can even obtain the parameters set that produce more reductions than the predefined objective.

Performance Comparison between RL-based Approach and MBF: In this section, we conducted the experiment to compare the performance of our proposed RL agents with MBF-based approach presented in the work of [170] due to the fact that MBF utilizes the similar decisive parameters in our technique. Also, this work is the first research effort to employ MBF on top of the built-in flow entry eviction mechanism in OpenFlow switches. The flow table-hit ratio is used as the measurement to compare the performance, and the statistics event handler "*FlowStatsReceived*" provided by POX API is utilized for collecting the flow statistics. From the experiment, we generate a traffic pattern with around 8K flows and 48 million packets for this emulation process. In addition, a fixed flow table size of 4K is used in this experiment, which is employed to ensure the agent does not exceed the flow table limit when selecting the flow entries.

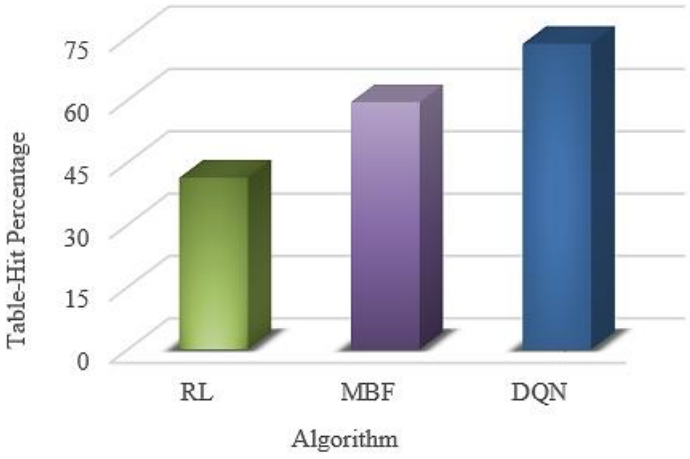


Figure 36. The comparison between our proposed RL agents and the MBF-based approach in terms of table-hit ratio

From Figure 36, we can observe that up to 41.3% table-hit ratio is achieved by the traditional RL agent, whereas MBF-based approach can achieve up to 59.5% [170] and DQN agent is up to 73.5% table-hit ratios. Thus, based on this results, our DQN learning agent outperforms the MBF-based approach by 14% in table-hit ratio, while the

traditional RL agent is 16.4% lower than the MBF-based approach in terms of table-hot ratio.

To this end, we can conclude on the basis of our experiments that performance of the DQN-based learning agent is better than both traditional RL agent and MBF-based approach presented by Challa *et al*. [170].

The significance of the Decisive Parameters: In our final experiment, we are interested in knowing the significance of each decisive parameter for satisfying the predefined objective for the emulation experiment. As shown in Figure 37, when the DQN agent is trained using 20 parameters sets with the objective of achieving the 60% reduction in overall control plane overhead, the total overhead reduction rate was only 12.3% when flow match frequency is the solo parameter considered in the flow entries selection process. The overall control plane overhead reduction is improved to 23.2% when the flow recentness parameter is considered alone. This results imply that the flow recentness parameter is more significant than the flow match frequency parameter on impacting the performance of the learning agent. As illustrated in Figure 37, the total overall control plane overhead can be reduced to 58.5% when considering both parameters jointly. Hence, this experiment provides valuable insight that both parameters should be utilized in the flow entries filtering and selection process.

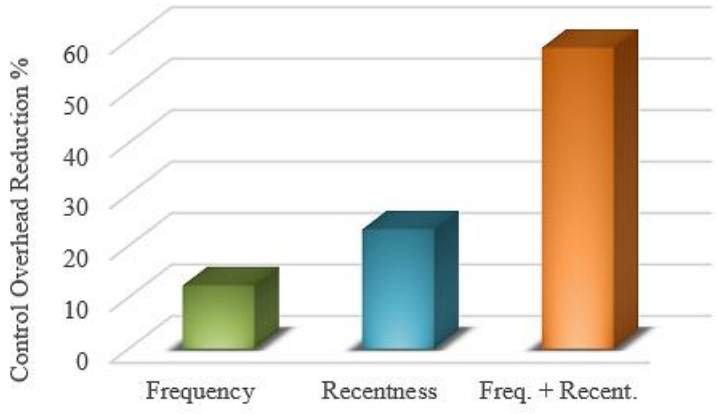


Figure 37. The significance of each decisive parameter, and the impact when they are considered jointly

4.6 Conclusion

In this chapter, we have demonstrated a novel RL-based approach that automatically search for the values of the two decisive parameters used to choose the candidate flow entries to be installed inside the switch's memory with the goal to minimize the overall control plane overhead while improving the table-hit ratio over the SDN-enabled environment. The results generated by our experiments indicate the proposed RL approach is more effective in minimizing the control plane overhead in comparing with the MBF-based approach proposed in the recent research efforts. The performance evaluations prove that the DQN-based learning agent outperforms the traditional RL-based agent and results in better efficiency in the aspect of convergence time. From statistical perspective, the emulation results demonstrate the DQN method has the ability to reduce the control plane overhead to 60%, with 13.9% more table-hit rate compared to the MBF-based method. Our novel technique promotes an online and flow pattern-aware forwarding rules selection mechanism that improves the utilization of the capacity of the switch's TCAM module while minimizing the overall control plane overhead effectively.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

The presence of novel parametric systems focusing on providing more efficient manner in dealing with complicated processes. To facilitate the flexibility and resilience of the system, the framework of the system provides a set of tunable and reconfigurable parameters that can be adjusted depending on the underlying hardware the system operates with. To achieve the maximum performance, those parameters must be studied and tuned carefully. Due to the dynamic nature of the available computing resources, the parameters tuning process of the system is considered as NP-Hard since there is no known effective mathematical formula can be utilized to locate the best solution.

Given that fact that it is impractical to locate the best solution mathematically, the most suitable approach is to utilize approximation to obtain the near optimal solution. The results of this work advance the understanding of the employment of the machine learning technique, particularly the reinforcement learning (RL), to automatically and adaptively tune those reconfigurable parameters to acquire the maximum performance of the system. To evaluate the effectiveness of the RL-based approach, the comparison among traditional RL algorithm, DQN-based algorithm, and other research efforts, has been conducted in this work, and it was observed that the DQN-based approach has better performance/efficiency in finding the parameters set that significantly improve the performance of the system automatically and adaptively given the dynamics of the system. Additionally, the significance of each reconfigurable parameter was further examined. Depending the characteristics of the system and the applications operating on the system, each tunable parameter possesses different level of significance. Although the parameter with the most significance can have the major influence on the system performance, the overall performance improvement is superior when all the configurable parameters were considered jointly, whereas the improvement scale is limited when the parameters were considered individually.

By providing a greater understanding of the efficacy for employing reinforcement learning toward self-reconfigurable systems and implementation of novel techniques to locate the near optimal solution adaptively, this work has contributed to the advancement

of machine learning technology in searching for the optimal configuration that fulfills the predefined objectives of the system.

Despite the fact that the RL and DQN based agents perform well in our RL-MRCONF approach, further development is needed to expand the capabilities of our simulator and the underlying learning algorithms. Currently, our simulator only takes into account six MapReduce configuration parameters. With the capability of deep reinforcement learning, more parameters can be considered and the MapReduce process can be studied and analyzed more precisely. In addition, the current version of the simulator does not include the implementation of Hadoop YARN resources management module. The RL-MRCONF was implemented based on the work of Wang *et al*. [149], [150] with the goal of making MR-Perf deployable on top of NS-3 simulator. With this objective in mind, we have no intention to change the core implementation of MR-Perf and we leave it as is since MR-Perf has modeled the performance of most Hadoop setups effectively. Nonetheless, since such support will enhance the value of RL-MRCONF and enable us to explore the configurations of MapReduce thoroughly, implementing the support of Hadoop YARN module is the focus of our ongoing research.

Furthermore, the value of the exploration rate in our scheme still needs to be studied carefully in order to ensure the learning process of the agent does not deviate from its intended predefined objective. Such deviation will increase the time required to find the MapReduce configuration that satisfies the goal conditions. Lastly, we plan to deploy the proposed system into real-world cloud environment, such as amazon AWS MapReduce service.

In the application of using RL-based approach in managing flow entries in the SDN environment, although the machine learning-based approach performs well in our experiments, further implementation is required to expand the capabilities of the emulation scope. Currently, our experiments were built on top of the simplified network topology, and the real-world deployments are considered as the essential step to prove the value of the proposed approach. Moreover, the flow entry collection process initiated in this research needs to be studied carefully in order to build more intelligent state space construction process that adapts to the dynamic nature of the network. Since such support will promote the significance of the research and allows us to have deeper comprehension

in SDN flow management, development of such framework in support of the real-world SDN topology is the center of our ongoing research.

CHAPTER 6

PUBLISHED WORKS

Publications

1. M. Anan, A. Al-Fuqaha, N. Nasser, T. Mu, and H. Bustam, “Empowering Networking Research and Experimentation through Software-Defined Networking,” *Journal of Network and Computer Applications*, vol. 70, pp. 140-155, July 2016.

2. T. Mu, A. Al-Fuqaha, K. Shuaib, F. M. Sallabi, and J. Qadir, “SDN Flow Entry Management Using Reinforcement Learning,” *ACM Transactions on Autonomous and Adaptive Systems*, vol. 13, no. 2, article 11, Nov. 2018

3. T. Mu, A. Al-Fuqaha, and K. Salah, “Automating the Configuration of MapReduce: A Reinforcement Learning Scheme,” accepted on *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, Dec. 2019

REFERENCES

[1] Mitchell, T. M. *The Discipline of Machine Learning*, Machine Learning Department technical report CMU-ML-06-108. Pittsburgh, PA: Carnegie Mellon University, July 2006

[2] Mitchell, T. M. *Machine Learning*. New York: McGraw-Hill, 1997

[3] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998

[4] S. Marsland. *Machine Learning*. CRC Press, 2nd Edition, 2014

[5] J. Schmidhuber, "Deep Learning in Neural Networks: An Overview," *Neural Networks*, vol. 61, pages 85-117, January 2015

[6] D. Meyer, V. Fuller, D. Lewis, E. Lear, S. Brim, D. Oran, N. Chiappa, J. Curran, and D. Farinacci, "Locator/ID Separation Protocol (LISP) Tutorial," Dec. 2007. [Online]. Available: http://www.ietf.org/proceedings/70/slides/RRG-7.pdf

[7] [30] F. d. O. Silva, J. H. d. S. Pereira, P. F. Rosa, and S. T. Kofuji, "Enabling Future Internet Architecture Research and Experimentation by Using Software Defined Networking," in *European Workshop on Software Defined Networking (EWSDN)*, pp. 73-78, October 2012

[8] Open Networking Foundation. [Online]. Available: https://www.opennetworking.org, July 2014

[9] H. Kim and N. Feamster. "Improving Network Management with Software Defined Networking," *IEEE Communication Magazine*, vol. 51, pp. 114-119, Feburary 2013

[10] R. Jain and S. Paul. "Network Virtualization and Software Defined Networking for Cloud Computing: A Survey," *IEEE Communication Magazine*, vol. 51, pp. 24-31, November 2013

[11] N. Feamster, J. Rexford, and E. Zegura. "The road to SDN: an intellectual history of programmable networks." *ACM SIGCOMM Computer Communication Review* 44.2, pp. 87-98, 2014

[12] Nadeau, Thomas D., and Ken Gray. "*SDN: Software Defined Networks.*" O'Reilly Media, Inc., 2013.

[13] Tail-f Systems. "Tutorial – SDN, NETCONF and YANG," 2013. [Online]. Available: https://www.comsoc.org/form/tutorial-registration-sdn-netconf-and-yang-programmable-networks-managed

[14] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," in *ACM SIGCOMM Computer Communications Review*, vol. 38, no.2, pp. 73-78, April 2008

[15] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis, "The Locator/ID Separation Protocol (LISP)," *Internet Engineering Task Force RFCs, ISSN 2070-1721*, RFC 6830, January 2013

[16] L. Yang, R. Dantu, T. Anderson, and R. Gopal, "Forwarding and Control Element Separation (ForCES) Framework," *Internet Engineering Task Force RCFs*, RFC 3746, April 2004

[17] A. Doria, J. Hadi Salim, R. Haas, W. Wang, R. Gopal, and J. Halpern, "Forwarding and Control Element Separation (ForCES) Protocol Specification," *Internet Engineering Task Force RFCs, ISSN: 2070-1721*, RFC 5810, March 2010

[18] D. Allan and T. Nadeau, “A Framework for Multi-Protocol Label Switching (MPLS) Operations and Management (OAM),” *Internet Engineering Task Force RFCs*, RFC 4378, February 2006

[19] E. Rosen, A. Viswanathan, and R. Callon, “Multiprotocol Label Switching Architecture,” *Internet Engineering Task Force RFCs*, RFC 3031, January 2001

[20] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow, “RSVP-TE: Extensions to RSVP for LSP Tunnels,” *Internet Engineering Task Force RFCs*, RFC 3209, December 2001

[21] Open vSwitch: An Open Virtual Switch. [Online]. Available: http://www.openvswitch.org, July 2014

[22] J. Pettit, J. Gross, B. Pfaff, M. Casado, and S. Crosby, “Virtual Switching in an Era of Advanced Edges,” in *2nd Workshop on Data Center – Converged and Virtual Ethernet Switching (DC-CAVES)*, September 06, 2010

[23] B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, and S. Shenker, “Extending Networking into the Virtualization Layer,” in *ACM Workshop on Hot Topics in Networks (HotNets-VIII)*, October 22-23, 2009

[24] A. T. Campbell, I. Katzela, K. Miki, and J. Vicente, “Open Signaling for ATM, Internet and Mobile Networks (OPENSIG’98),” in *Workshop of Open Signaling Working Group (OPENSIG)*, October 1998

[25] Ethane: A Security Management Architecture, [Online]. Available: http://yuba.stanford.edu/ethane/index.html

[26] M. Casado, M. J. Freedman, J. Pettit, J. Lou, N. McKeown, and S. Shenker. "Ethane: Taking control of the enterprise." *ACM SIGCOMM* Computer Communication Review 37.4, pp. 1-12, October 2007

[27] The 4D Project. [Online]. Available: http://www.cs.cmu.edu/~4D/, July 2014

[28] The Internet Engineering Task Force (IETF). [Online]. Available: http://www.ietf.org, July 2014

[29] R. Enns, “NETCONF Configuration Protocol,” *Internet Engineering Task Force RFCs*, RFC 4741, December 2006

[30] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, “Network Configuration Protocol (NETCONF),” *Internet Engineering Task Force RFCs*, ISSN: 2070-1721, RFC 6241, June 2011

[31] K. Ogawa, W. Wang, E. Haleplidis, and J. Hadi Salim, “High Availability within a Forwarding and Control Element Separation (ForCES) Network Element,” *Internet Engineering Task Force RFCs, ISSN: 2070-1721*, RFC 7121, February 2014

[32] A. Rodriguez-Natal, A. Cabellos-Aparicio, S. Barkai, V. Ermagan, D. Lewis, F. Maino, and D. Farinacci, “Software Defined Networking extensions for the Locator/ID Separation Protocol,” *IETF LISP Working Group Internet-Draft*, February 7, 2014

[33] M. Boucadair and C. Jacquenet, “Software-Defined Networking: A Perspective From Within A Service Provider,” *IETF Network Working Group Internet-Draft*, January 6, 2014

[34] M. Bjorklund, “YANG – A Data Modeling Language for the Network Configuration Protocol (NETCONF),” *Internet Engineering Task Force RCFs*, RFC 6020, October 2010

[35] Internet Research Task Force (IRTF). [Online]. Available: https://irtf.org, July 2014
[36] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "NOX: Towards an Operating System for Networks," *ACM SIGCOMM* Computer Communication Review, 38(3), pp. 105-110, July 2008
[37] NOX/POX. [Online]. Available: http://www.noxrepo.org, July 2014
[38] Trema OpenFlow Controller. [Online]. Available: https://github.com/trema/trema, July 2014
[39] Floodlight Controller. [Online]. Available: http://www.projectfloodlight.org/floodlight/, July 2014
[40] Ryu SDN Framework. [Online]. Available: http://osrg.github.io/ryu/, July 2014
[41] OpenDaylight – A Linux Foundation Collaborative Project. [Online]. Available: http://www.opendaylight.org, July 2014
[42] NetFPGA. http://www.netfpga.org, July 2014
[43] G. Antichi, A. D. Pietro, S. Giordano, G. Procissi, and D. Ficara, "Design and Development of an OpenFlow Compliant Smart Gigabit Switch," in *IEEE Global Telecommunication Conference (GLOBECOM 2011)*, pp. 1-5, December 2011
[44] J. Naous, D. Erickson, G. A. Covington, G. Appenzeller, and N. McKeown, "Implementing an OpenFlow Switch on the NetFPGA platform," in *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS'08)*, pp. 1-9, 2008
[45] B. Lantz, B. Heller, and N. McKeown, "A Network in a Laptop: Rapid Prototyping for Software-Defined Networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, October 2010
[46] Mininet – An Instant Virtual Network on your Laptop (or other PC). [Online]. Available: http://www.mininet.org/, July 2014
[47] Big Switch, Indigo. [Online]. Available: http://www.projectfloodlight.org/indigo/
[48] R. Giladi and N. Yemini. "A Programmable, Generic Forwarding Element Approach for Dynamic Network Functionality." in *Proceedings of the 2nd ACM SIGCOMM workshop on Programmable Routers for Extensible Services of Tomorrow*. *PRESTO*, pp. 19-24, August 21 2009
[49] M. Berman, J. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, and I. Seskar, "GENI: A federated testbed for innovative network experiments." *Computer Networks*, vol. 61, pp. 5-23, March 14, 2014
[50] H. Kim, J. Kim, and Y. Ko, "Developing a cost-effective OpenFlow testbed for small-scale Software Defined Networking." *Advanced Communication Technology (ICACT)*, 2014 16th International Conference on. IEEE, 2014
[51] GENI, [Online]. Available: www.geni.net, June 2014
[52] COTN, [Online]. Available: http://www.cenic.org/page_id=143/, June 2014
[53] J. Kim, B. Cha, J. Kim, N. Kim, G. Noh, Y. Jang, H. An, H. Park, J. Hong, D. Jang, T. Ko, W, Song, S. Min, J. Lee, B. Kim, I. Cho, H. Kim, and S. Kang, "OF@TEIN: An OpenFlow-enabled SDN Testbed over International SmartX Rack Sites." *Proceedings of the Asia-Pacific Advanced Network* 36, pp. 17-22, 2013
[54] TWAREN, [Online]. Available: http://www.twaren.net/ , June 2014
[55] National Center for High-performance Computing (NCHC), [Online]. Available: http://www.nchc.org.tw/en/, February 2014

[56] J. Hu, W. Huang, H. Tseng, H. Lee, L. Ku, S. Lin, T. Liu, and C. Yang, "Future Internet in Taiwan: Design and Research Activities over TWAREN Network." *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on. IEEE,* 2012

[57] OCEAN, [Online]. Available: http://ocean.cs.illinois.edu/, June 2014

[58] Inter-Datacenter WAN with centralized TE using SDN and OpenFlow, [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/customer-case-studies/cs-googlesdn.pdf, June 2014

[59] S. Jain, A. Kumar, S. Mandal, J. Ong, L, Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Xhu, J. Zolla, U. Holzle, S. Stuart, and A. Vahdat, "B4: Experience with a globally-deployed software defined WAN." *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*. ACM, 2013

[60] U. Hölzle, "OpenFlow@Google," *Open Networking Summit 2012*, Apr. 2012.

[61] E. Kawai, "Can SDN Help HPC?," *Applications and the Internet (SAINT), 2012 IEEE/IPSJ 12th International Symposium on* , vol., no., pp. 210, 16-20 July 2012

[62] National Institute of Information and Communications Technology. [Online]. Available: http://www.nict.go.jp/en/ , February 2014

[63] Y. Kanaumi, S. Saito, E. Kawai, S. Ishii, K. Kobayashi, and S. Shimojo, "Deployment and operation of wide-area hybrid OpenFlow networks." *Network Operations and Management Symposium (NOMS)*, 2012 IEEE. IEEE, 2012

[64] Y. Kanaumi, S. Saito, E. Kawai, S. Ishii, K. Kobayashi, and S. Shimojo, "RISE: A Wide-Area Hybrid OpenFlow Network Testbed." *Ieice Transactions on Communications* 96.1, pp. 108-118, 2013

[65] V. Kotronis, D. Schatzmann, and B. Ager, "On bringing private traffic into public SDN testbeds." *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013

[66] Enterprise GENI, [Online]. Available: https://www.geni.net/?p=1463, June 2014

[67] TangoGENI, [Online]. Available: http://groups.geni.net/geni/wiki/TangoGENI, June 2014

[68] S. Narayan, S. Bailey, M. Greenway, R. Grossman, A. Heath, R. Powell, and A. Daga, "OpenFlow Enabled Hadoop over Local and Wide Area Clusters," *High Performance Computing, Networking, Storage and Analysis (SCC)*, 2012 SC Companion: , vol., no., pp.1625-1628, 10-16 November 2012

[69] S. Narayan, S. Bailey, and A. Daga, "Hadoop Acceleration in an OpenFlow-Based Cluster," *High Performance Computing, Networking, Storage and Analysis (SCC)*, 2012 SC Companion: , vol., no., pp.535-538, 10-16 November 2012

[70] M. Campanella, "The FEDERICA Project: creating cloud infrastructures." *Proceedings of Cloudcomp*, pp. 19-21, 2009

[71] M. Campanella, and F. Farina, "The FEDERICA infrastructure and experience." *Computer Networks* (2014)

[72] R. Sherwood, G. Gibb, K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Can the production network be the testbed?" in *Proceedings of USENIX OSDI*, Canada, 4-6 October 2010

[73] R. DoriguzziCorin, M. Gerola, R. Roggio, F. De Pellegrini, and E. Salvadori, "VeRTIGO: Network virtualization and beyond," in *Proceedings of EWSDN*, Germany, October 2012

[74] M. Suñé, L. Bergesio, H. Woesner, T. Rothe, A. Kopsel, D. Colle, B. Puype, D. Simeonidou, R. Nejabati, M. Channegowda, M. Kind, T. Dietz, A. Autenrieth, V. Kotronis, E. Salvadori, S. Salsano, M. Korner, and S. Sharma, "Design and implementation of the OFELIA FP7 facility: The European OpenFlow testbed." *Computer Networks*, vol. 61, pp. 132-150, March 14 2014

[75] J. Matias, B. Tornero, A. Mendiola, E. Jacob, and N. Toledo, "Implementing Layer 2 Network Virtualization using OpenFlow: Challenges and Solutions." *Software Defined Networking (EWSDN)*, 2012 European Workshop on. pp. 30-35, IEEE, 2012

[76] M. Gerola, R. Doriguzzi Corin, R. Riggio, F. De Pellegrini, E. Salvadori, H. Woesner, T. Rothe, M. Sune, and L. Bergesio, "Demonstrating inter-testbed network virtualization in OFELIA SDN experimental facility," *INFOCOM*, April, 2013

[77] A. Köpsel, and H. Woesner. "OFELIA–pan-european test facility for openflow experimentation." *Towards a Service-Based Internet*. Springer Berlin Heidelberg, pp. 311-312, 2011

[78] E. Dimogerontakis, I. Vilata, and L. Navarro, "Software Defined Networking for community network testbeds," *Wireless and Mobile Computing, Networking and Communications (WiMob)*, 2013 IEEE 9th International Conference on , pp. 111-118, 7-9, October 2013

[79] CONFINE Project – Community Networks Testbed for the Future Internet. [Online]. Available: http://confine-project.eu/

[80] B. Godfrey, I. Ganichev, S. Shenker, and I. Stoica, "Pathlet routing." *ACM SIGCOMM Computer Communication Review*. Vol. 39. No. 4. ACM, 2009

[81] D. Pitt, "Trust in the cloud: the role of SDN." *Network Security*, pp. 5-6, March 2013

[82] IDC Predictions 2013: Competing on the 3rd Platform, [Online]. Available: http://www.idc.com/research/Predictions13/downloadable/238044.pdf, April 2014

[83] A. Sydney, D. Ochs, C. Scoglio, D. Gruenbacher, and R. Miller, "Using GENI for Experimental Evaluation of Software Defined Networking in Smart Grids." *Computer Networks*, vol. 63, pp. 5-16 April 22, 2014

[84] S. Das, G. Parulkar, N. McKeown, P. Singh, D. Getachew, and L. Ong, "Packet and circuit network convergence with OpenFlow." *Optical Fiber Communication Conference*. Optical Society of America, 2010.

[85] MIT Tech Review, 2009; [Online]. Available: http://www2.technologyreview.com/specialreports/specialreport.aspx?id=37

[86] H. Goudarzi, M. Ghasemazar, and M. Pedram. "Sla-based optimization of power and migration cost in cloud computing." *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium* on. IEEE, 2012

[87] B. Nunes, M. Mendonca, X. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks." *Communications Surveys & Tutorials, IEEE* vol. 16, pp. 1617-1634, February 13 2014

[88] L. Suresh, J. Schulz-Zander, R. Merz, A. Feldmann, and T. Vazao, "Towards programmable enterprise WLANs with Odin." *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012

[89] L. Suresh, J. Schulz-Zander, R. Merz, A. Feldmann, and T. Vazao, "Demo: programming enterprise WLANs with Odin." *ACM SIGCOMM Computer Communication Review* 42.4, pp. 279-280, 13-17 August 2012

[90] A. Nayak, A. Reimers, N. Feamster, and R. Clark, "Resonance: Dynamic Access Control for Enterprise Networks." *Proceedings of the 1st ACM workshop on Research on enterprise networking*. ACM, 2009

[91] A. Gember, P. Prabhu, Z. Ghadiyali, and A. Akella, "Toward Software-defined Middlebox Networking." *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*. ACM, 2012

[92] SDN FOR HOME NETWORKS, [Online]. Available: http://onrc.stanford.edu/research_sdn_home_networks.html, March 2014

[93] K. Calvert, W. Edwards, N. Feamster, R. Grinter, Y. Deng, and X. Zhou, "Instrumenting Home Networks." *ACM SIGCOMM Computer Communication Review* 41.1, pp. 84-89, 2011

[94] R. Mortier, T. Rodden, T. Lodge, D. McAuley, C. Rotsos, A. Moore, A. Koliousis, and J. Sventek, "Control and Understanding: Owning Your Home Network." *Communication Systems and Networks (COMSNETS), 2012 Fourth International Conference* on. IEEE, 2012

[95] F. Hsu, M. Malik, and S. Ghorbani. "OpenFlow as a Service."

[96] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, "ElasticTree: Saving Energy in Data Center Networks." *NSDI*. vol. 10. 2010

[97] E. Keller, S. Ghorbani, M. Caesar, and J. Rexford, "Live Migration of an Entire Network (and its Hosts)." *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*. ACM, 2012

[98] H. Shirayanagi, H. Yamada, and K. Kono, "Honeyguide: A VM Migration-aware Network Topology for Saving Energy Consumption in Data center Networks." *IEICE TRANSACTIONS on Information and Systems* 96.9, pp. 2055-2064, 2013

[99] X. Wang, Z. Liu, Y. Qi, and J. Li, "LiveCloud: A Lucid Orchestrator for Cloud Datacenters." Cloud Computing Technology and Science (CloudCom), *2012 IEEE 4th International Conference* on. IEEE, 2012

[100] R. Raghavendra, J. Lobo, and K. Lee, "Dynamic Graph Query Primitives for SDN-based Cloud Network Management." *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012

[101] W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya, "Cost of Virtual Machine Live Migration in Clouds: A Performance Evaluation." *Cloud Computing. Springer Berlin Heidelberg*, pp. 254-265, 2009

[102] K. Yap, R. Sherwood, M. Kobayashi, T. Huang, M. Chan, N. Handigol, N. McKeown, and G. Parulkar, "Blueprint for Introducing Innovation into Wireless Mobile Networks." *Proceedings of the second ACM SIGCOMM workshop on Virtualized infrastructure systems and architectures*. ACM, 2010

[103] L. Li, Erran, Z. Mao, and J. Rexford. "Toward Software-Defined Cellular Networks." Software Defined Networking (EWSDN), *2012 European Workshop* on. IEEE, 2012

[104] K. Yap, M. Kobayashi, R. Sherwood, N. Handigol, T. Huang, M. Chan, and N. McKeown, "OpenRoads: Empowering Research in Mobile Networks." *ACM SIGCOMM Computer Communication Review* 40.1, pp. 125-126, 2010

[105] M. Bansal, J. Mehlman, S. Katti, and P. Levis. "OpenRadio: A Programmable Wireless Dataplane." *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012

[106] I. Seskar, K. Nagaraja, S. Nelson, and D. Raychaudhuri. "MobilityFirst Future Internet Architecture Project." *Proceedings of the 7th Asian Internet Engineering Conference*. ACM, 2011

[107] MobilityFirst Future Internet Architecture Project [Online]. Available: http://mobilityfirst.winlab.rutgers.edu/ , April 2014

[108] H. Owens, and A. Durresi, "Video over Software-Defined Networking (VSDN)," Network-Based Information Systems (NBiS), *2013 16th International Conference* on , vol., no., pp.44,51, 4-6 September 2013

[109] H.E. Egilmez, B. Gorkemli, A.M. Tekalp, and S. Civanlar, "Scalable Video Streaming Over OpenFlow Networks: An Optimization Framework for QoS Routing." *Image Processing (ICIP), 2011 18th IEEE International Conference* on. IEEE, 2011

[110] A. Khurshid, X. Zou, W. Zhou, M. Caesar, and P.B. Godfrey, "VeriFlow: Verifying Network-Wide Invariants in Real Time." *ACM SIGCOMM Computer Communication Review* 42.4, pp. 467-472, 2012

[111] A. Singla, C. Hong, L. Popa, and P.B. Godfrey, "Jellyfish: Networking Data Centers Randomly." *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 2012.

[112] M. Canini, P. Kuznetsov, D. Levin, and S. Schmid, "Software Transactional Networking: Concurrent and Consistent Policy Composition." *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013

[113] A. Bianco, L. Giraudo, and D. Hay. "Optimal Resource Allocation for Disaster Recovery." *Global Telecommunications Conference (GLOBECOM 2010),* 2010 IEEE. IEEE, 2010

[114] Raspberry Pi. [Online]. Available: http://www.raspberrypi.org/, June 9, 2014

[115] J. Zhang, B. Seet, T. Lie, and C. Foh, "Opportunities for Software-Defined Networking in Smart Grid." *Information, Communications and Signal Processing (ICICS) 2013 9th International Conference* on. IEEE, 2013

[116] L. Ilyes, M. Anan, A. Al-fuqaha, and M. Ayyash, "Cloud-based Autonomic Service Monitoring In The Future Internet," *IEEE International Wireless Communications and Mobile Computing Conference (IWCMC),* pp.63-68, 2014

[117] IEEE & Smart Grid, [Online]. Available: http://smartgrid.ieee.org/ieee-smart-grid, October, 2014

[118] V.C. Gungor, B. Lu, and G.P. Hancke. "Opportunities and Challenges of Wireless Sensor Networks in Smart Grid." *Industrial Electronics, IEEE Transactions* on, no. 10, pp. 3557-3564, 2010

[119] [C. Tang, M. Steinder, M. Spreitzer and G. Pacifici, "A scalable application placement controller for enterprise data centers," *The 16th International World Wide Web Conference (WWW'07),* pp. 331-340, 2007.

[120] N. Bobroff, A. Kochut, K. Beaty, "Dynamic Placement of Virtual Machines for Managing SLA Violations," *IEEE International Symposium on Integrated Network Management*, pp. 119-128, 2007.

[121] H. Goudarzi, M. Ghasemazar and M. Pedram, "SLA-based Optimization of Power and Migration Cost in Cloud Computing," *IEEE/ACM International Symposium on Cluster Computing and the Grid*, pp. 172-179, 2012

[122] D. Ardagna, B. Panicucci, M. Trubian, L. Zhang, "Energy-Aware Autonomic Resource Allocation in Multi-Tier Virtualized Environments," *IEEE Transactions on Services Computing*, vol. 5, no. 1, pp. 2-19, 2010

[123] H. Goudarzi and M. Pedram, "Energy-Efficient Virtual Machine Replication and Placement in a Cloud Computing System", *IEEE 5th International Conference on Cloud Computing*, pp. 750-757, 2012

[124] ETSI, "Network Functions Virtualisation: An Introduction, Benefits, Enablers, Challenges & Call for Action." [Online]. Available: https://portal.etsi.org/NFV/NFV_White_Paper.pdf. October 2012

[125] ETSI, "Network Functions Virtualisation: Architectural Framework." [Online]. Available: https://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.01.01_60/gs_NFV002v010101p.pdf. October 2013

[126] H. Hawilo, A. Shami, M. Mirahmadi, and A. Asal, "NFV: State of the Art, Challenges and Implementation in Next Generation Mobile Networks (vEPC)." *IEEE Network*, vol. 28, no. 6, pp. 18-26, Nov.-Dec. 2014

[127] M. Jammal, T. Singh, A. Shami, R. Asal, and Y. Li, "Software-Defined Networking: State of the Art and Research Challenges." *Computer Networks Journal*, vol. 72, 29, pp. 74-98, October 2014

[128] L. Liao, A. Shami, V. C.M. Leung, "DFVisor: A Distributed FlowVisor Platform for QoS awared Cloud Network Virtualization." *IET Networks Journal*, vol. 4, no. 5, pp. 270-277, September 2015

[129] W. Xia, Y. Wen, C. Foh, D. Niyato, and H. Xie, "A Survey on Software-Defined Networking." *IEEE Communication Surveys & Tutorials*, vol. 17 no. 1 2015

[130] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: A Network Programming Language." *Proceedings of the 16th ACM SIGPLAN international conference on Functional programming. ICFP*, pp. 279-291, 2011

[131] N. Foster, M. J. Freedman, A. Guha, R. Harrison, N. Katta, C. Monsanto, J. Reich, M. Reitblatt, J. Rexford, C. Schlesinger, A. Story, and D. Walker, "Language for Software-Defined Networks." *IEEE Communications Magazine*, vol. 51, no. 2, pp. 128-134, February 2013

[132] J. Reich, C. Monsanto, N. Foster, J. Rexford, and D. Walker, "Modular SDN Programming with Pyretic." *The Usenix Magazine*, vol. 38, no. 5, pp. 40-47, October 2013

[133] A. Voellmy and P. Hudak, "Nettle: Functional Reactive Programming of OpenFlow Networks." *Proceedings of the 13th international conference on Practical Aspects of Declarative Languages. PADL*, 2011

[134] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click Modular Router." *ACM Transactions on Computer Systems (TOCS)*, vol 18, no. 3, pp. 263-297, August, 2000
[135] Click Modular Router. Click Elements. http://read.cs.ucla.edu/click/click, September 2015
[136] M. Dobrescu, N. Egi, K. Argyraki, B. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy, "RouteBricks: Exploiting Parallelism to Scale Software Routers." *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles. SOSP*, pp. 15-28, 2009
[137] J. Martins, M. Ahmed, C. Raiciu, and F. Huici, "Enabling Fast, Dynamic Network Processing with ClickOS." *Proceedings of the 2nd workshop on Hot topics in software defined networking. ACM SIGCOMM*, pp. 67-72, 2013
[138] R. Riggio, T. Rasheed, and F. Granelli, "EmPOWER: A Testbed for Network Function Virtualization Research and Experimentation." *IEEE SDN conference for Future Networks and Services. SDN4FNS*, pp. 1-5, November 2013
[139] A. Gudipati, D. Perry, L. E. Li, and S. Katti, "SoftRAN: Software Defined Radio Access Network." *Proceedings of the 2nd workshop on Hot topics in software defined networking. ACM SIGCOMM*, pp. 25-30, 2013
[140] X. Jin, L. E. Li, L. Vanbever, and J. Rexford, "SoftCell: Scalable and Flexible Cellular Core Network Architecture." *Proceedings of the 9th conference on Emerging networking experiments and technologies. ACM CoNEXT*, pp. 163-174, 2013
[141] T. Chen, H. Zhang, X. Chen, and O. Tirkkonen, "SoftMobile: Control Evolution for Future Heterogeneous Mobile Networks." *IEEE Wireless Communications*, vol. 21, no. 6, pp. 70-78, December 2014
[142] "Apache Hadoop." http://hadoop.apache.org
[143] J. Dean and S. Ghemawat, "Mapreduce: Simplified Data Processing on Large Clusters," *Communications of ACM*, vol. 51, pages 107-113, January 2008
[144] J. Dean and S. Ghemawat, "MapReduce: A Flexible Data Processing Tool," *Communications of ACM*, vol. 53, no 1, pages 72-77, 2010
[145] K. Wang, X. Lin, and W. Tang, "Predator – An Experience Guided Configuration Optimizer for Hadoop MapReduce," in *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference*, pages 419-426, 2012
[146] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving MapReduce Performance in Heterogeneous Environments," in *Proceedings of the 8th USENIX conference on Operating systems design and implementation*, pages 29-42, 2008
[147] S. Babu, "Toward Automatic Optimization of MapReduce Programs," in *Proceedings of the 1st ACM symposium on Cloud computing*, SoCC'10, pages 137-142, 2010
[148] Network Simulator 3 (NS-3). [Online]. Available: https://www.nsnam.org/
[149] G. Wang, A. R. Butt, P. Pandey, and K. Gupta, "A Simulation Approach to Evaluating Design Decisions in MapReduce Setups," in *IEEE International Symposium on MASCOTS 2009*, pages 1 – 11, September 2009
[150] G. Wang, A. R. Butt, P. Pandey, and K. Gupta, "Using Realistic Simulation for Performance Analysis of MapReduce Setups," in *Proceedings of the 1st ACM*

workshop on Large-Scale System and Application Performance 2009, pages 19 – 26, 2009

[151] C. Tain, H. Zhou, Y. He, and L. Zha, "A Dynamic MapReduce Scheduler for Heterogeneous Workloads," in *Grid and Cooperative Computing, 2009. GCC'09, 8th International Conference,* pages, 218-224, 2009

[152] D. Wu and A. Gokhale, "A Self-Tuning System based on Application Profiling and Performance Analysis for Optimizing Hadoop MapReduce Cluster Configuration," in *20th Annual International Conference in High Performance Computing*, pages 89- 98, December 2013

[153] P. Lama and X. Zhou, "AROMA: Automated Resource allo-cation and Configuration of MapReduce Environment in the Cloud," in *Proceedings of the 9th International Conference on Automatic Computing 2012*, pages 63 – 72, 2012

[154] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, and S. Babu, "Starfish: A Self-tuning System for Big Data Analytics," in *5th Biennial Conference on Innovative Data Systems Research (CIDR)*, pages 261 – 272, January 2011

[155] J. Rao, X. Bu, C. Xu, L. Wang, and G. Yin, "VCONF: A Reinforcement Learning Approach to Virtual Machine Auto-configuration," in *Proceedings of the 6th International Conference on Autonomic Computing*, ICAC '09, pages 137-146, 2009

[156] X. Bu, J. Rao, and C. Xu, "A Reinforcement Learning Approach to Online Web Systems Auto-configuration," in *Proceedings of the 29th IEEE International Conference on Distributed Computing Systems (ICDCS '09)*, pages: 2-11, 2009

[157] 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. December 19, 2013

[158] V. Mnih, K. Kavukcuoglu, D. Silver, AA. Rusu, J. Veness, MG. Bellemare, A. Graves, M. Riedmiller, AK. Fidjeland, G. Ostrovski, and S. Petersen, "Human-level Control through Deep Reinforcement Learning," *Nature*, 518(7540), pages 529-533. February 26, 2015

[159] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional Architecture for Fast Feature Embedding," in *Proceedings of the 22nd ACM International Conference on Multimedia*, pages 675-678. November 03, 2014

[160] Caffe Tutorial. [Online]. Available: http://caffe.berkeleyvision.org/tutorial/

[161] Google DeepMind. [Online]. Available: https://deepmind.com/

[162] Z. Bei, Z. Yu, H. Zhang, W. Xiong, C.Xu, L. Eeckhout, and S. Feng, "RFHOC: A Random-Forest Approach to Auto-Tuning Hadoop's Configuration," *IEEE Trans. On Parallel and Distributed Systems*, vol. 27, no. 5, pages, 1470-1483, May 2016

[163] L.Cai, Y. Qi, and J. Li, "A recommendation-based parameter tuning approach for Hadoop," in *2017 IEEE International Symposium on Cloud and Service Computing (IEEE SC2-2017)*, pages 223-230. November 22-25, 2017

[164] T. White, *Hadoop: The Definite Guide*, 3rd ed. O' Reilly Media/Yahoo Press, Sebastopol, California, 2012

[165] Cloudera, "Configuration Parameters," 2012. [Online]. Available: http://blog.cloudera.com/blog/author/aaron

[166] "Cloudera." http://www.cloudera.com
[167] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: Measurement & Analysis," in *Proceedings of the 9th ACM SIGCOMM conference on International measurement*, ser. IMC '09. New York, NY, USA: ACM, 2009, pp. 202-208
[168] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, ACM, Melbourne, 2010, pp. 267-280
[169] B. S. Lee, R. Kanagavelu and K. M. M. Aung, "An efficient flow cache algorithm with improved fairness in Software-Defined Data Center Networks," *2013 IEEE 2nd International Conference on Cloud Networking (CloudNet)*, IEEE, San Francisco, CA, 2013, pp. 18-24
[170] R. Challa, Y. Lee and H. Choo, "Intelligent eviction strategy for efficient flow table management in OpenFlow Switches," *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, IEEE, Seoul, 2016, pp. 312-318
[171] "TCAMs and OpenFlow – What Every Practitioner Must Know," [Online]. Available: https://www.sdxcentral.com/articles/contributed/sdn-openflow-tcam-need-to-know/2012/07/, 2012
[172] N. Katta, O. Alipourfard, J. Rexford, and D. Walker, "Inifinite CacheFlow in Software-Defined Networks," in *Proceedings of the 3rd workshop on Hot topics in software defined networking*. ACM, Chicago, pp. 175-180, 2014
[173] S. Banerjee and K. Kannan, "Tag-In-Tag: Efficient flow table management in SDN switches," *10th International Conference on Network and Service Management (CNSM) and Workshop*, IEEE, Rio de Janeiro, 2014, pp. 109-117
[174] S. Veeramani, M. Kumar and S. N. Mahammad, "Minimization of flow table for TCAM based openflow switches by virtual compression approach," *2013 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, IEEE, Kattankulathur, 2013, pp. 1-4
[175] H. Zhu, M. Xu, Q. Li, J. Li, Y. Yang and S. Li, "MDTC: An efficient approach to TCAM-based multidimensional table compression," *2015 IFIP Networking Conference (IFIP Networking)*, IEEE, Toulouse, 2015, pp. 1-9
[176] S. Luo, H. Yu and L. M. Li, "Fast incremental flow table aggregation in SDN," *2014 23rd International Conference on Computer Communication and Networks (ICCCN)*, IEEE, Shanghai, 2014, pp. 1-8
[177] M. Anan, A. Al-Fuqaha, N. Nasser, T. Mu, and H. Bustam, "Empowering Networking Research and Experimentation through Software-Defined Networking," *Journal of Network and Computer Applications*, vol. 70, pp. 140-155, July 2016
[178] M. Lu, W. Deng and Y. Shi, "TF-IdleTimeout: Improving efficiency of TCAM in SDN by dynamically adjusting flow entry lifecycle," *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, IEEE, Budapest, Hungary, 2016, pp. 002681-002686
[179] X. N. Nguyen, D. Saucez, C. Barakat and T. Turletti, "Rules Placement Problem in OpenFlow Networks: A Survey," in *IEEE Communications Surveys & Tutorials*, vol. 18, no. 2, pp. 1273-1286, May 20 2016

[180] C. Lee, Y. Nakagawa, K. Hyoudou, S. Kobayashi, O. Shiraki and T. Shimizu, "Flow-Aware Congestion Control to Improve Throughput under TCP Incast in Datacenter Networks." In *Proceedings of the IEEE 39th Annual Computer Software and Applications Conference*, IEEE, Taiwan, Taichung, 2015, pp. 155-162

[181] H. A. A. Al-Rawi, M. A. Ng, K.-L. A. Yau, "Application of reinforcement learning to routing in distributed wireless networks: a review", *Artificial Intelligence Review*, vol. 43, no. 3, pp. 381-416, 2015

[182] R. Desai, B.P. Patil, "Cooperative reinforcement learning approach for routing in ad hoc networks", in *Proceedings of the 2015 International Conference on Pervasive Computing (IEEE ICPC 2015)*, IEEE, Pune, pp. 1-5, January 8–10, 2015

[183] J. Solanki, A. Chauhan, "A Reinforcement Learning Network based Novel Adaptive Routing Algorithm for Wireless Ad-Hoc Network", *International Journal of Science Technology & Engineering*., vol. 1, no. 12, pp. 135-142, 2015

[184] 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," *2016 IEEE International Conference on Services Computing* (SCC), IEEE, San Francisco, CA, 2016, pp. 25-33

[185] S. Wang, H. Liu, P. H. Gomes, and B. Krishnamachari, "Deep reinforcement learning for dynamic multichannel access", in *International Conference on Computing, Networking and Communications (ICNC)*, IEEE, Silicon Valley, 257-265, 2017