OPTIMIZING ROUTING STRATEGY IN SOFTWARE DEFINED NETWORKING

Lingzhuo Tu

2304720

Supervisor: Dr. Nitheesh Kaliyamurthy

Project submitted as part of the requirements for the award of MSc: Software Engineer and Artificial Intelligence

September 2024

Declaration of Originality

I, Lingzhuo Tu declare that I am the sole author of this Project; that all references cited have been consulted; that I have conducted all work of which this is a record, and that the finished work lies within the prescribed word limits.

This work has not previously been accepted as part of any other degree submission.

Signed :..

Date :07/01/2024 ....

FORM OF CONSENT

I, Lingzhuo Tu, hereby consent that my Project, submitted in candidature for the MSC12 Software Engineering and Artificial Intelligence , if successful, may be made available for inter-library loan or photocopying (subject to the law of copyright), and that the title and abstract may be made available to outside organisations.

Signed : ....

Date : 07/01/2024...

Copyright Acknowledgement

ABSTRACT

Traditional Network Architecture (TNA) is becoming inadequate due to its rigid, hardware-centric configurations, especially in environments where network conditions are highly variable. This has led to increased latency, congestion, and packet loss rates. This project aims to develop an optimized routing strategy for Software Defined Networking (SDN) that leverages machine learning techniques to enhance network traffic management's adaptability and efficiency. The project employs a combination of Dueling Deep Q-Networks (Dueling DQN) and real-time traffic state predictions to create a dynamic routing strategy. The methodology includes extensive simulation using SDN environments to evaluate the performance improvements over traditional routing methods. Preliminary results indicate that the proposed SDN-based routing strategy not only responds more efficiently to dynamic network conditions but also significantly optimizes performance metrics such as bandwidth utilization, latency reduction, and packet loss. The integration of Dueling DQN and real-time traffic predictions within SDN frameworks could potentially redefine network performance standards, offering a more adaptive, efficient, and robust network management system. This study contributes to the field by providing a scalable solution to the complexities of modern network environments, supporting the ongoing evolution of network infrastructure management.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I, Lingzhuo Tu, sincerely express my gratitude to Dr. Nitheesh Kaliyamurthy from the University of Wales Trinity Saint David for his invaluable support and guidance throughout this project. His expert advice on academic writing, critical thinking, and research methodologies has significantly contributed to my research work. His consistent feedback and encouragement have helped me refine my ideas and navigate the complexities of this research, particularly in model development and result analysis.

I am also thankful to the Network Information Laboratory of Wuhan University of Technology for allowing me to collect and utilize data for this project. Their support was crucial in obtaining a high-quality research dataset. Additionally, I appreciate the provision of remote servers and high-performance graphics cards by the same laboratory, which were essential for meeting the computational demands of this research.

Furthermore, I extend my heartfelt thanks to the University of Wales Trinity Saint David for providing me access to its online library, which offered a wealth of academic resources and literature foundational to my research. The availability of these resources ensured that I could conduct a comprehensive literature review and establish a solid theoretical base for my work.

CHAPTER 1 INTRODUCTION

The evolution of network infrastructure management has increasingly required the development of new technologies and methods that can keep pace with the growing complexity and dynamism of modern networks. Traditional Network Architecture (TNA) relies on rigid, hardware-centric configurations, making it less adaptable to the changing conditions and demands of network traffic. TNA typically employs static routing protocols which decide paths based predominantly on initial configurations and infrequently updated network state information. This method becomes inadequate in environments where network conditions fluctuate, leading to inefficiencies such as increased latency, congestion, and higher packet loss rates.

In contrast, Software Defined Networking (SDN) offers a transformative approach to network management. SDN separates the network's control plane from the data plane, centralizing control in a software-based controller. This architectural change introduces a level of flexibility and dynamism that TNA[1] cannot match. The central controller in an SDN can view and manage the network holistically, making real-time, informed decisions that adapt to changes in network traffic patterns and conditions. This model not only simplifies network design and operation but also enhances scalability and agility in deploying new services.

Comparing TNA and SDN, the latter's centralized control mechanism allows for a more nuanced and responsive management approach. SDN's ability to programmatically direct traffic and dynamically adjust to network conditions can significantly optimize the performance metrics of the network. This includes better utilization of bandwidth, minimized latency, and decreased packet loss, particularly in dynamic and complex network environments.

Moreover, the predictive capabilities that can be integrated into SDN, as explored in this project through the use of Dueling Deep Q-Networks (Dueling DQN) and real-time traffic predictions, enable preemptive adjustments to routing decisions. Such anticipatory measures are crucial in maintaining optimal network performance and can

[1] TRADITIONAL NETWORK ARCHITECTURE

greatly enhance the adaptability of SDN compared to the more reactive and less flexible TNA. This project proposes not just a shift from static to dynamic routing but a move towards intelligent, learning-driven network management that stands to redefine the standards of network performance and reliability.

1.1 RESEARCH PROBLEM STATEMENT

The conventional routing algorithms employed in SDN environments are often rigid and unable to adapt dynamically to changing network conditions. While SDN offers the potential for more flexible and efficient network management, the existing routing methods do not fully adapt this flexibility, leading to suboptimal network performance and inefficiencies. Furthermore, the integration of advanced machine learning techniques with SDN routing strategies remains underexplored. This presents a significant gap in the research, as there is a pressing need for innovative routing algorithms that can leverage the capabilities of SDN to improve the adaptability and efficiency of network traffic management. Thus, the research problem addressed in this project is the development of an optimized routing strategy within SDN that effectively combines machine learning techniques to enhance network performance and management.

1.2 AIM

The aim of this project is to develop an optimized routing strategy specifically for SDN, aimed at enhancing the efficiency of network traffic management. The project seeks to implement and refine a routing optimization solution within a SDN framework, utilising the capabilities of Dueling DQN deep reinforcement learning and network traffic state prediction. This approach is designed to enhance network management by leveraging real-time global network topology and link status information, enabling a more responsive, adaptive, and efficient network routing process that can dynamically optimize network performance and service quality.

1.3 OBJECTIVES

The objectives of the project are detailed as follows:

- To conduct extensive research into existing literature concerning SDN and routing optimization techniques to provide a theoretical foundation for the proposed solution.
- To adapt a suitable research methodology by identifying the appropriate simulation tools and statistical techniques for analyzing network performance, as well as outlining the criteria for model validation and evaluation, to ensure rigorous and reproducible results within the project.
- To design and implement a routing optimization model that leverages Dueling DQN reinforcement learning combined with network traffic state predictions.
- To evaluate the proposed model through experimental setups, comparing its performance against traditional routing methods and other SDN-based solutions to demonstrate its efficacy in real-world scenarios.
- Analyze different network states and their impact on routing strategies, utilizing simulations to gauge performance improvements over traditional methods.

1.4 SIGNIFICANCE/CONTRIBUTION OF THIS RESEARCH

This project substantially enhances the field of SDN by pioneering the innovative integration of Dueling DQN reinforcement learning with real-time network traffic predictions. Utilizing a blend of SDN capabilities, including centralized control, flexible management, and the integration of heterogeneous network data, the project develops a cutting-edge and comprehensive computational model for routing optimization. This holistic approach significantly boosts the accuracy and efficiency of routing decisions within SDNs, addressing critical inefficiencies in traditional routing methods. As a result, the proposed model offers a quicker and more effective method for managing network traffic flow, ensuring optimal network performance and reliability. Such advancements are crucial for supporting the growing demands on network infrastructure, thereby enabling significant progress in information technology and network management applications.

1.5 STRUCTURE OF THE PROJECT

The structure of this thesis is designed to systematically address the research objectives and provide a coherent flow of information and analysis:

Chapter One – Introduction This chapter provides a broad overview of the project, including the research problem statement, aims, and objectives. It sets the context for the research by discussing the need for optimized routing strategies in Software Defined Networking (SDN) and outlines the significance and potential contributions of the study.

Chapter Two – Review of Literature This chapter presents a thorough analysis of existing literature on SDN, covering fundamental concepts, architecture, and various controllers. It identifies gaps in current research and demonstrates how the project's approach can address these deficiencies.

Chapter Three – Research Methodology This chapter details the research methods used to achieve the objectives of the study. It describes the experimental design, data collection techniques, and analytical methods employed, ensuring that the research is reproducible and valid.

Chapter Four – Architectural Design and Modeling: This newly added chapter provides a detailed exposition of the architectural design and modeling processes for Software Defined Networking (SDN). It includes discussions on key technologies and design choices.

Chapter Five – RESULT AND ANALYSIS This chapter discusses the experimental setup, the processes involved in implementing the routing strategies, and an analysis of the results obtained. It evaluates the effectiveness of the proposed routing optimizations in real-world scenarios.

Chapter Six – Conclusions and Recommendations The final chapter synthesizes the findings of the research, discusses the implications, and offers conclusions based on the evidence gathered. It also provides recommendations for future research and practical applications of the study's outcomes.

Chapter Seven – References This chapter lists all the bibliographic references used throughout the thesis, providing a comprehensive resource for understanding the theoretical and empirical bases of the study.

CHAPTER 2 - REVIEW OF LITERATURE

The purpose of this chapter is to conduct a thorough analysis of the existing body of literature related to SDN, with a focus on its architecture, control mechanisms, operational functions, routing capabilities, and limitations, as well as reviewing similar works in the field. This chapter seeks to identify key trends, debates, and gaps within these areas, examining both conventional network management strategies and advanced approaches enabled by SDN technology. The scope of literature reviewed extends from foundational theories to the latest advancements in SDN research.

2.1 SOFTWARE DEFINED NETWORKING (SDN)

With the continuous development of information technology, the emergence of new network technologies such as big data[1], virtualization[2], and cloud computing[3] has progressively magnified the shortcomings of traditional network architectures. Due to their cumbersome network configurations, traditional networks impose heavy maintenance tasks on network administrators[4]. Additionally, the fixed topology of traditional networks limits their flexibility and scalability, greatly constraining network development. To address these issues, Software Defined Networking[2] (SDN) has been proposed as a new network architecture. Its flexible structure with decoupled control and data planes, along with the programmable nature of the network, offers a novel approach to resolving the rigidity of current networks[5].

2.1.1 SDN Fundamentals

In traditional network architectures, the control plane and the data plane are tightly coupled within network devices, with each underlying device functioning as a complete and independent entity. This results in a fixed network topology and cumbersome network configurations, rendering the current network environment rather rigid. Additionally, the use of distributed routing algorithms[6] means that each router makes

[2] Software Defined Networking (SDN) separates network management (control plane) from the forwarding of data packets (data plane), allowing for centralized and programmable network traffic management, which increases flexibility and simplifies administration.

independent routing decisions without considering other routers, a design approach that significantly complicates network upgrades. Furthermore, different types of network devices require distinct configuration tools, demanding a higher level of skill from network administrators. The maintenance of traditional networks consumes considerable time and effort, thereby increasing the total system costs, including acquisition, operational, and management expenses[7].

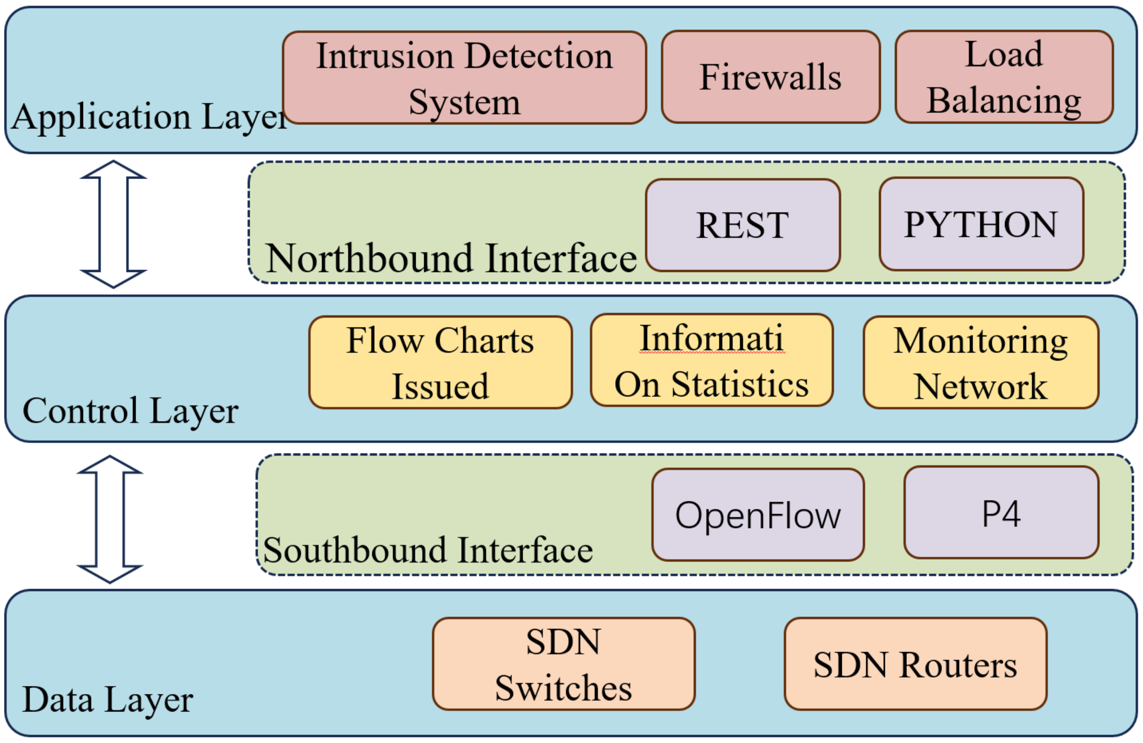
Originating from the academic environment of Stanford University's network data forwarding project, the design philosophy of SDN is to decouple data forwarding from the network control plane, enhancing network flexibility[8]. This design facilitates programming of the underlying hardware through software modules of the controller, improving control effectiveness and enabling rational allocation of network resources according to user needs. In SDN, when determining data forwarding paths via the control plane, it is necessary not only to analyze the direction of the next hop of the data but also to study the structure of all current network links. This allows for an understanding of any potential overload risks of devices within the network. By preemptively switching data transmission paths in switches via the controller, it is possible to circumvent potential problem nodes, thus avoiding impacts on network traffic flow.

Compared to conventional networking architectures, SDN offers distinct advantages. On one hand, within such an architecture, the hardware setup only needs to consider whether the storage and forwarding capacities meet the usage requirements, which can substantially reduce architectural costs[9]. On the other hand, it maintains the existing network infrastructure without the need for reconfiguration, thus simplifying the deployment process. Moreover, SDN architectures provide a faster response to business demands, which in practice allows for the personalized customization of network parameters such as policies and routing. This results in significantly reduced time for business processing.

2.1.2 SDN Basic Architecture

The basic architecture of SDN comprises five main components: the data plane, southbound interfaces, control plane, northbound interfaces, and the application layer, as illustrated in Figure 1 [10]. The data plane primarily handles the forwarding and processing of data packets and is composed of underlying devices such as SDN switches and SDN routers. The southbound interfaces are specific protocols that facilitate communication between the control plane and the data plane. The control plane, the core of the SDN architecture, provides a global network view to the application layer and allocates flow tables to the devices in the data plane based on the business requirements of the application layer. Northbound interfaces are specific protocols that enable communication between the application layer and the control plane. The application layer consists of multiple SDN applications, which interact with the controller via northbound interfaces to implement various network functions, such as routing calculation, load balancing, and congestion detection[11].

Figure 1 SDN Basic Architecture



1. Data Plane

The data plane, also known as the data layer, consists of various network devices such as routers, switches, and others that perform data forwarding based on control decisions[12]. The SDN data plane has the following three characteristics:

- Programmability: The SDN data plane can be configured through programming, allowing network administrators to control and manage network traffic according to specific needs. This programmability not only enhances network flexibility but also enables better network management by administrators.

- Centralized Control: The control logic of the SDN data plane is centrally stored in the controller rather than being distributed across various network devices. This centralized control approach allows network administrators to achieve global control over the network state and allocate and optimize global network resources according to specific business requirements.
- Openness: The SDN data plane adopts open protocols such as OpenFlow, which facilitates easy integration with devices and software from other vendors, enhancing network interoperability and scalability.

The working principle of the SDN data plane is based on flow table[13] forwarding, similar to routing tables in traditional networks. A flow table consists of three parts: matching rules, actions, and counters. Matching rules refer to the values of various fields in the TCP/IP header, such as MAC address, source IP address, destination IP address, VLAN ID, etc. These specific values constitute a flow. The action part includes operations on the flow, such as forwarding packets to specified ports, discarding packets, sending to the normal processing pipeline, etc. If the SDN switch does not contain an entry for a specific flow, the default action is to forward the packet to the controller. Upon receiving the packet, the controller creates a flow table entry for the flow and sends it back to the switch. The switch can then process packets based on the entries in the flow table. The counter part records the number of packets for each flow and other statistical information, which can be grouped by each flow, each table, or each port[14].

2. Southbound Interfaces

Southbound interfaces are specific protocols for communication between the control layer and the data layer, providing the controller with the ability to control and manage network traffic. For example, using the uplink channel of a southbound interface, the controller can uniformly monitor and collect statistics on the information reported by underlying switching devices, thereby achieving link discovery. Using the downlink channel of a southbound interface, the controller can also uniformly control network devices to implement flow table distribution[15].

In SDN, the most widely used southbound interface standard is the OpenFlow protocol from the open-source community. The OpenFlow protocol provides convenient messaging mechanisms. For example, it generates event-based messages when ports or links change; flow-based statistical messages during network monitoring by the controller; and Packet-in messages sent to the controller for processing when a switch does not know how to handle a new incoming packet.

Besides the OpenFlow protocol, there are other southbound interface standards, such as Open vSwitch Database (OVSDB) [16], ForCES [17], and Programming Protocol-Independent Packet Processors (P4)[18]. OVSDB provides additional network management functions, allowing the creation of virtual switch instances, setting interfaces, and connecting them to switches. OpFlex allows forwarding devices to handle some management functions, abstracting policies from the underlying layer and deciding where to place these functions. The P4 protocol enables users to define their own network protocols, including flow table matching rules and packet processing logic, thus achieving flexible traffic control and management. The introduction of the P4 protocol allows network devices to define their forwarding behavior through software, better adapting to various network application scenarios.

3. Control Layer

The control layer is a critical component of the SDN architecture, responsible for controlling and managing the entire network. Its primary function is to act as a bridge between the application layer and the data layer, handling interactions between applications and underlying forwarding devices[19]. For example, it translates application layer policies into executable instructions for the data layer and provides relevant information from the data layer to applications. The SDN controller also enables centralized management of global SDN elements, monitoring network status and formulating forwarding strategies. Through northbound interfaces, it offers programmability to the application layer, allowing network managers to flexibly create network services according to user needs. Table 1 lists the current mainstream SDN controllers.

Table 1 Mainstream SDN Controllers

Controller	Southbound Interface	Programming Language	System Platform	Description
NOX	OpenFlow	C++	Linux	The first SDN controller to support the

				OpenFlow protocol
POX	OpenFlow	Python	Linux, Mac OS, Windows	A Python-based SDN controller evolved from NOX, supporting the OpenFlow protocol
Ryu	OpenFlow, Netconf, OF-config, etc.	Python	Linux	Ryu is a lightweight, open-source SDN controller supporting OpenFlow v1.0, v2.0, and v3.0
Floodlight	OpenFlow	Java	Linux, Mac OS, Windows	Provides a general set of functions for controlling and querying OpenFlow networks, meeting various user network needs

4. Northbound Interfaces

Northbound interfaces are the connections between the control layer and the application layer in the SDN architecture. Their main function is to provide a standardized interface for applications, allowing them to manage and control the network through the SDN controller without directly accessing the underlying physical devices[20]. This standardized interface enables different applications to seamlessly communicate with the SDN, thereby achieving more flexible and programmable network management. For example, through northbound interfaces, applications can send requests to the controller to obtain network status information, instruct network behavior, and control network traffic. This flexible network management approach significantly enhances network manageability and controllability, better meeting the needs of various scenarios.

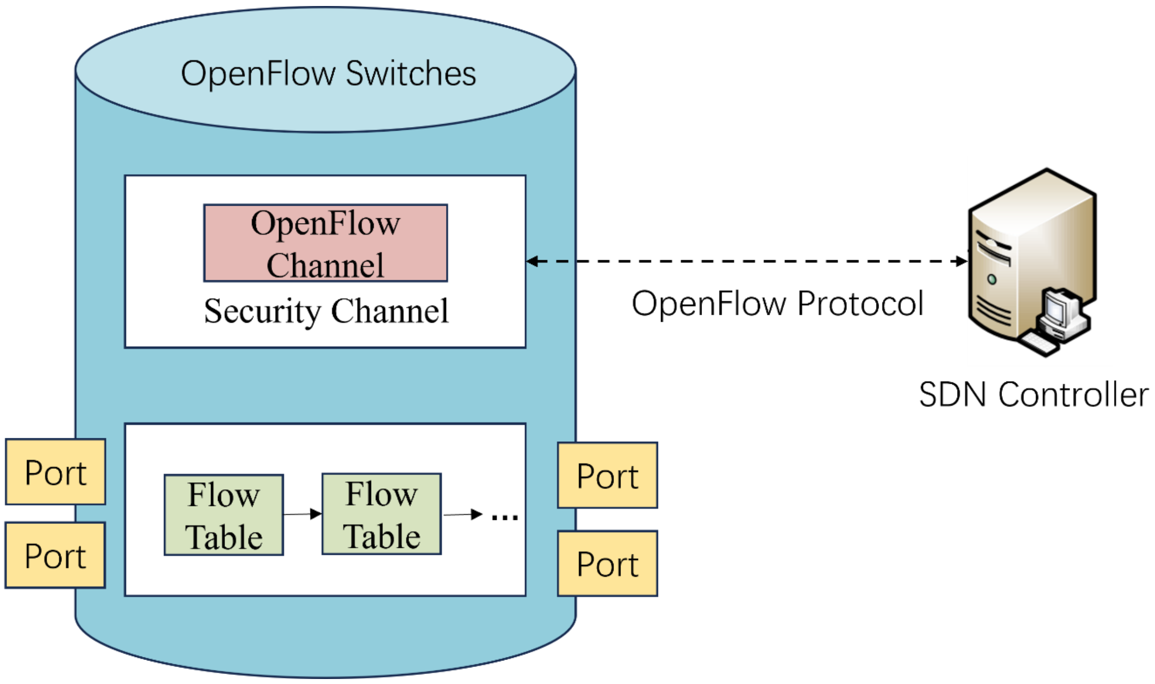
5. Application Layer

The application layer provides a platform for network administrators to implement control logic by configuring network devices to achieve specific network behaviors and functions. Typical SDN applications include intrusion detection systems, load balancing, traffic optimization, firewalls, and fine-grained access control [21]. SDN applications can also abstract and encapsulate their functions, providing northbound proxy interfaces. These encapsulated interfaces can be considered as higher-level northbound interfaces, thereby offering more advanced functions and services.

2.1.3 OpenFlow Protocol

SDN has many protocol standards in practical applications, among which the most popular protocol is OpenFlow [22]. OpenFlow is based on the concept of flow to establish and match rules. Through the OpenFlow protocol, the SDN controller can query, modify, and configure the status information of SDN switches, and update the network system status in real time. The main components of an OpenFlow switch include a secure channel, flow tables, and the OpenFlow protocol, as shown in Figure 2. Among these, the secure channel is the interface connecting the OpenFlow switch with the SDN controller, the flow table is a collection of forwarding policies, and the OpenFlow protocol is the standard protocol for interaction between the control layer and the data layer.

Figure 2 OpenFlow Switch Architecture



The OpenFlow protocol supports three types of interaction messages: Controller-to-Switch messages, asynchronous messages, and synchronous messages [23]. The controller sends Controller-to-Switch messages to the switch to query and modify the switch's status and configuration, some of which do not require a response from the switch. The switch sends asynchronous messages to the controller, providing real-time feedback on network update events and requesting new instructions. Asynchronous messages mainly include Packet-in messages, which are sent when the switch encounters an unmatched packet and needs the controller to handle it; Flow-Removed messages, which notify the controller to delete a flow entry when the flow table changes, such as when a flow entry times out; and PortStatus messages, which inform the controller of changes in the switch's ports or settings. Synchronous messages can be initiated by either the controller or the switch and are used to establish connections and check if the other party is online.

The flow table mechanism is a critical component of the OpenFlow protocol, enabling the decoupling of the control layer and the data layer. With the evolution of OpenFlow versions, the structure and functionality of flow tables have continually been enriched. In OpenFlow 1.0[24], each OpenFlow switch maintains only one flow table and can communicate with only one controller. OpenFlow 1.1[25] upgraded to support multiple flow tables, decomposing the flow table matching process into several steps and forming a pipeline processing method to avoid the excessive expansion of a single flow table. OpenFlow 1.2 introduced the TLV (Type-Length-Value) structure to define matching fields, enabling more keywords to be matched and allowing OpenFlow switches to communicate with multiple controllers. OpenFlow 1.3, the most stable version, enriched the structure of flow entries by adding priority, timeouts, and cookies, making packet matching more flexible and enabling timely cleanup of unused flow entries to reduce the switch's burden[26]. Additionally, OpenFlow 1.3 introduced auxiliary connections, effectively improving switch processing efficiency and enabling application parallelism[27].

2.1.4 Summary

The discussion in this section serves as a foundational background, setting the stage for exploring more advanced topics in SDN, including various types of controllers and the detailed operations of SDN networks. It emphasizes the transformative potential of SDN in adapting to the increasing complexity and requirements of modern network environments, thus framing the motivation for further innovations and research in network management.

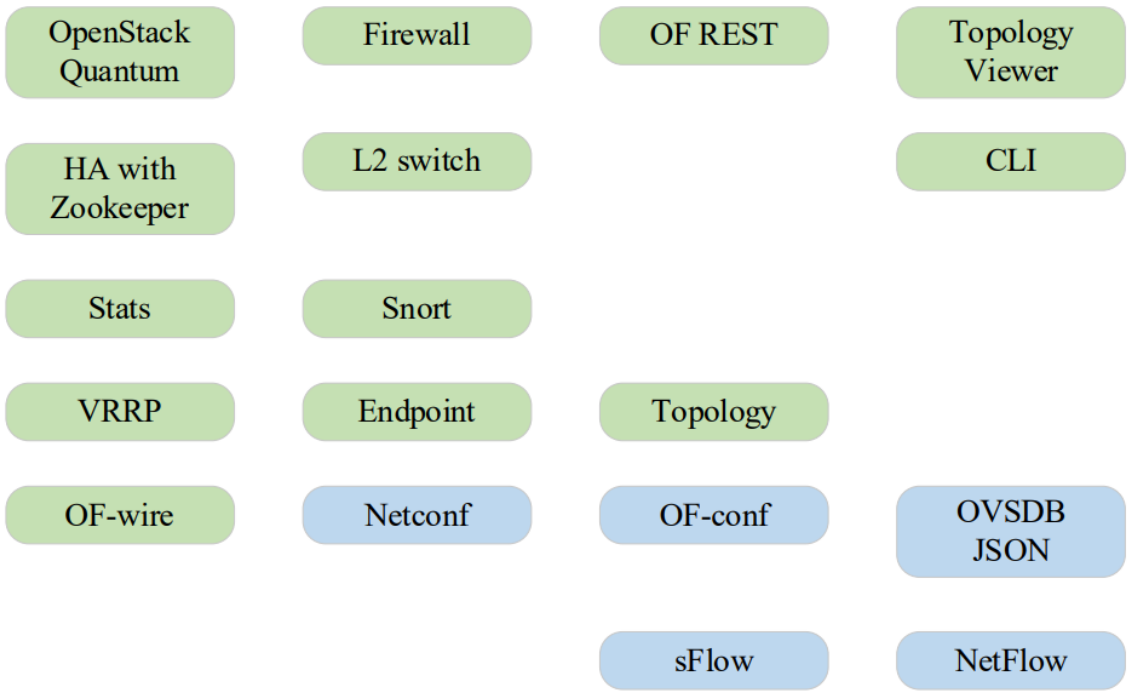
2.2 TYPE OF CONTROLLERS

2.2.1 RYU

Ryu is an open-source project led by the Japanese company NTT, with its name meaning "flow" in Japanese. The project aims to provide a SDN operating system with logically centralized control capabilities. Ryu offers comprehensive API interfaces, enabling network application developers to easily create new management and control applications[28]. Written in Python and adhering to the Apache License, Ryu supports multiple versions of the OpenFlow protocol, including v1.0, v1.2, and v1.3.

The Ryu controller comprises a wide array of libraries and components designed for developing SDN applications[29]. These libraries encapsulate common functions distilled from the requirements of SDN controllers and can be directly invoked within components. Each component operates independently of others. Through these features, Ryu offers developers a flexible and scalable SDN development environment, enhancing the convenience and intelligence of network management and control. The libraries and components provided by Ryu are illustrated in Figure 3.

Figure 3 Ryu library functions and components

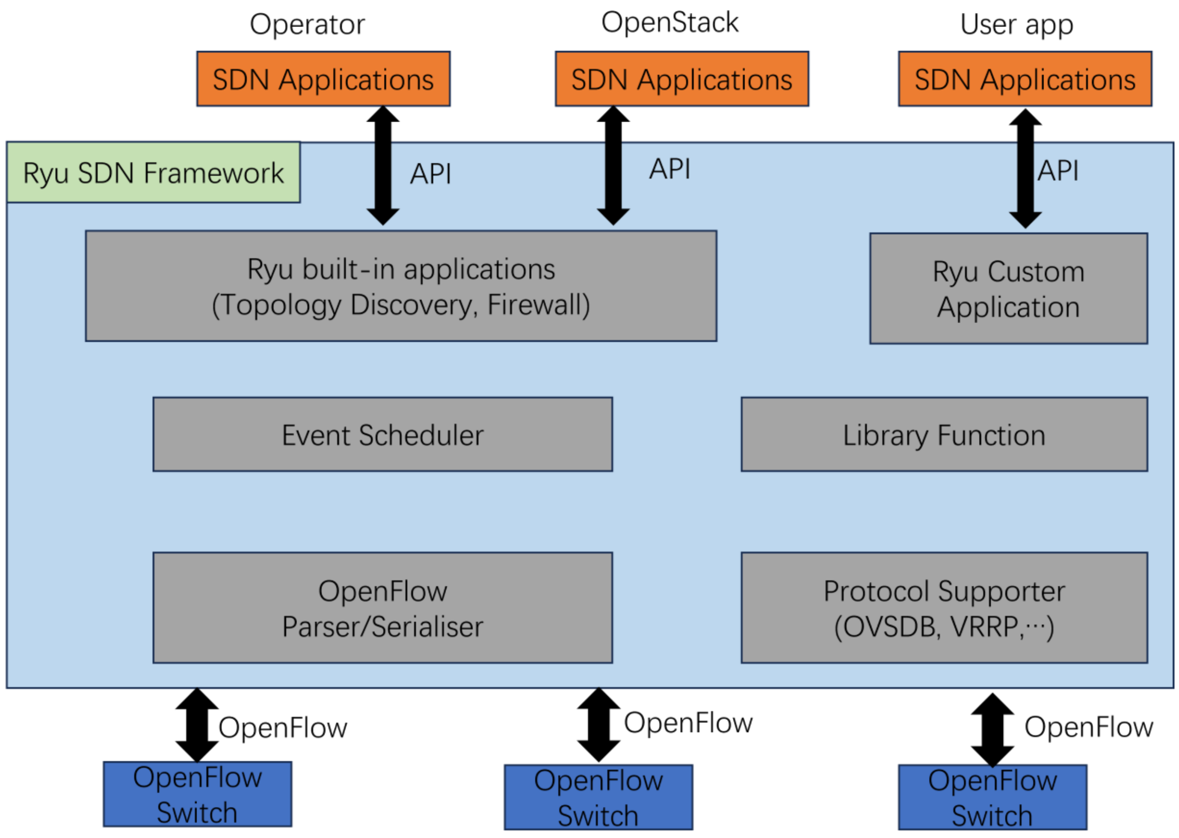


Libraries such as Netconf[30], OF-conf, and sflow[31] primarily facilitate the control functions for OpenFlow switches. Among the key components, OF-wire provides support for different versions of the OpenFlow protocol; Topology is responsible for building topology maps and tracking link status; and OF REST offers REST APIs for users to configure OpenFlow switches. The VRRP[32] component adds VRRP capabilities to OpenFlow switches, significantly enhancing network reliability. Additionally, Ryu can integrate with the OpenStack cloud computing platform, allowing users to manage and control their networks on demand.

2.2.1.1 Ryu Overall Architecture

The Ryu SDN framework primarily provides control capabilities, offering services to SDN applications through northbound REST APIs, enabling these applications to orchestrate and control network traffic[33]. Through southbound protocols such as OpenFlow, Ryu controls OpenFlow switches to facilitate traffic interaction. The Ryu SDN architecture serves as a pivotal bridge, acting as the control and exchange hub for northbound interfaces. The overall architecture of Ryu is illustrated in Figure 4.

Figure 4 Ryu Overall Architecture



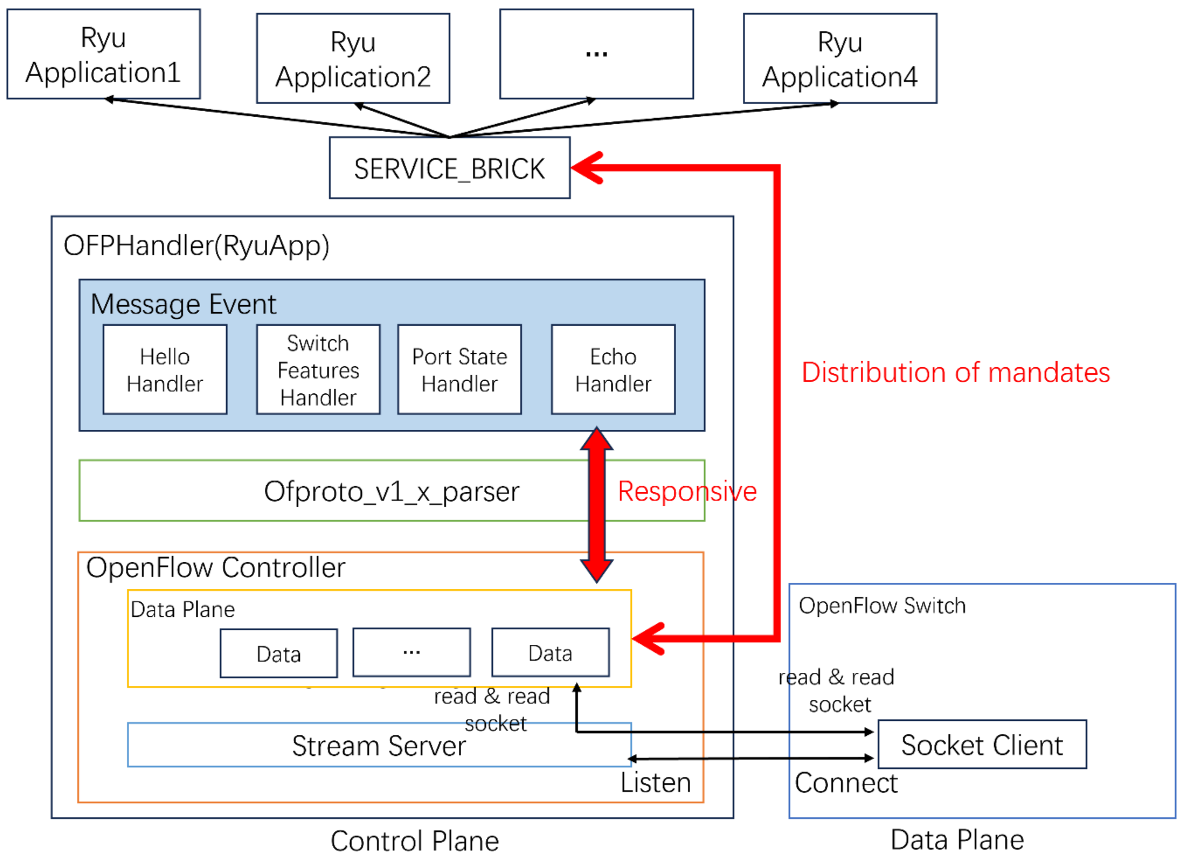
The SDN application layer is broadly divided into three categories. The first category is the Operator, which controls and manages the SDN framework through RESTful management APIs. The second category is OpenStack cloud orchestration, which integrates with OpenStack using REST API for Quantum to manage and control the network. The third category is User apps, which control and manage the SDN framework through user-defined APIs via REST or RPC[34].

The Ryu SDN framework layer is the core of the entire architecture, providing the infrastructure for developing, managing, and running SDN applications[35]. The main components and functionalities include Ryu applications, event dispatcher, libraries, OpenFlow parser/serializer, and protocol support modules. Ryu applications are specific SDN programs running on the Ryu framework that perform particular network management tasks. The event dispatcher is responsible for receiving, processing, and distributing events, ensuring coordination and communication among different parts of the system. The libraries contain various functional modules and tools for Ryu applications to simplify the development process[36]. The OpenFlow parser/serializer handles OpenFlow protocol packets, performing parsing and generation to ensure communication between the controller and switches. The protocol support modules support various network protocols (e.g., OVSDB, VRRP), providing broader functionality and compatibility for network management[37].

The OpenFlow switch layer comprises switches that support the OpenFlow protocol, serving as the infrastructure for network packet forwarding. OpenFlow switches communicate with the SDN controller via the OpenFlow protocol, receiving flow table instructions and executing corresponding forwarding operations to ensure efficient transmission of network data[38].

2.2.1.2. Ryu Workflow

Figure 5 Ryu Workflow



The workflow of Ryu is illustrated in Figure 5. Upper-layer Ryu applications distribute and transmit events through SERVICE_BRICK[39]. The main purpose of

SERVICE_BRICK is to implement modular design, enabling Ryu applications to be developed and maintained as independent service modules. Each SERVICE_BRICK is an independent service module responsible for specific functions or tasks and can communicate and collaborate with other modules through Ryu's service registration and discovery mechanisms. Moreover, SERVICE_BRICK is closely integrated with the event handling mechanism. Events are routed and tasks are distributed by registering callback functions that respond to events within the Ryu applications, allowing the Ryu framework to operate efficiently in an event-driven manner[40].

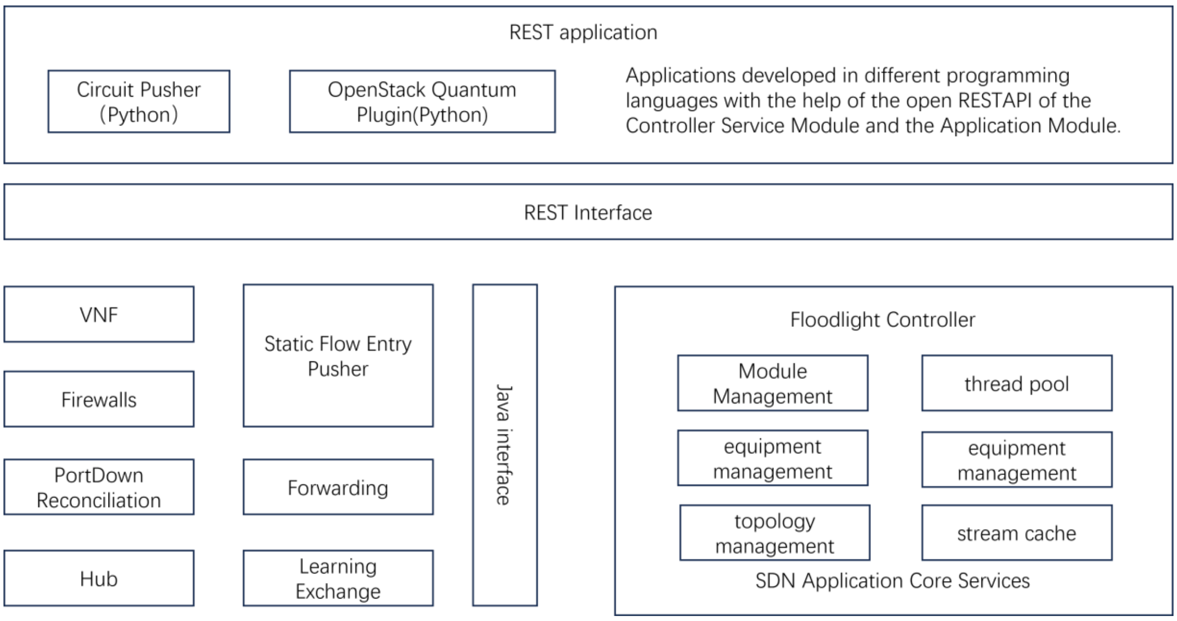
OFPHandler[41] is the most fundamental subclass of RyuAPP. This class primarily handles the coordination of OpenFlow protocol tasks such as Hello Handler, Switch Features Handler, Port State Handler, and Echo Handler. OFPHandler instantiates an OpenFlow controller object, which in turn instantiates several dataplane objects corresponding to the number of connected switches, with each dataplane representing a single OpenFlow switch.

The Datapath communicates with OpenFlow switches by creating sockets[42] using the Stream server from the high-concurrency Python framework eventlet[43]. Eventlet provides an efficient network communication mechanism, enabling the dataplane to handle multiple concurrent connections and communications with switches effectively. Each dataplane object is responsible for receiving and processing OpenFlow messages sent from its corresponding switch and returning the processing results to the switch, thus facilitating real-time communication and control between the controller and the switches.

2.2.2 Floodlight

The Floodlight controller boasts excellent stability and portability, being compatible with various operating systems[44]. Therefore, this project utilizes Floodlight as the SDN controller. Floodlight interacts with upper-layer applications via Java interfaces or REST APIs, with its overall architecture illustrated in Figure 6.

Figure 6 Architecture diagram of Floodlight



Floodlight is composed of core service modules, regular application modules, and REST application modules[45]. The core service modules provide fundamental support services via Java interfaces and REST APIs to both the regular application modules and the REST application modules. The regular application modules depend on the core service modules and provide services to the REST application modules. The REST application modules rely on the REST APIs provided by the core service modules and regular application modules, allowing applications to perform their functions by simply calling the REST APIs[46]. Developers can create applications using the system-provided APIs or add their own modules and expose APIs for use by other developers. This modular and hierarchical deployment approach effectively enhances the controller's scalability. The functional components of Floodlight are illustrated in Table 2.

Table 2 Floodlight Components

Component Type	Component Name	Function Description
Core Service Module	FloodlightProvider	Manages connections to switches and converts OpenFlow messages into

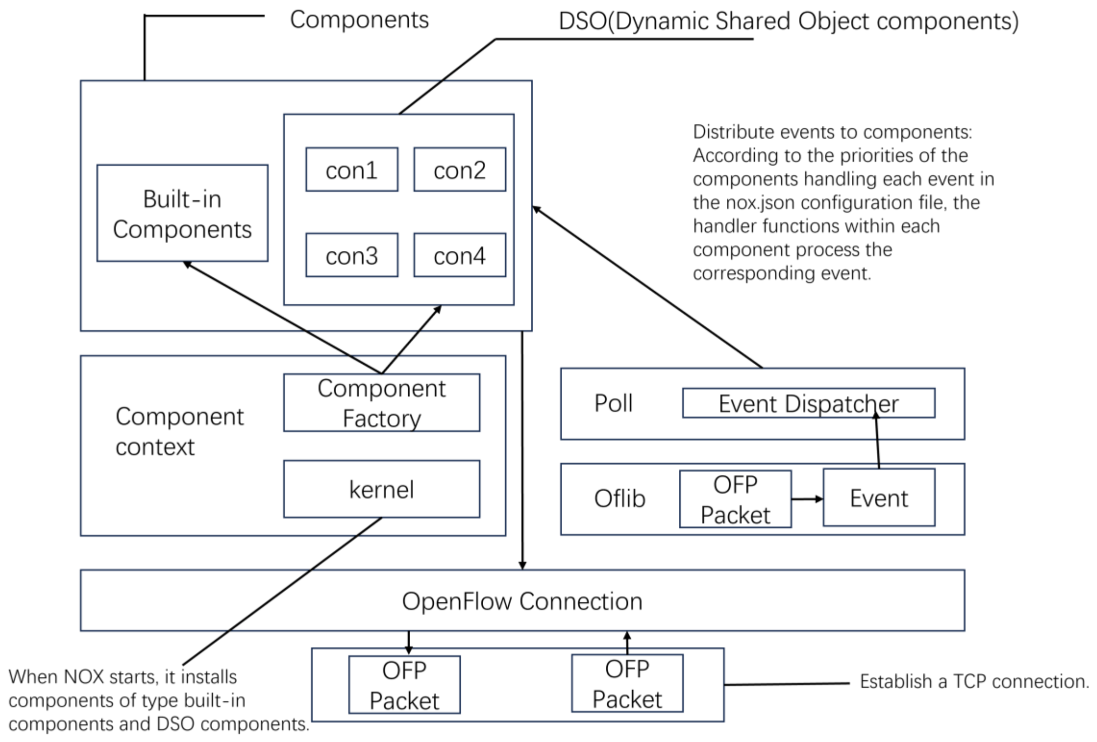
			events that other modules can listen to.
		DeviceManagerImpl	Manages low-level network devices such as switches and hosts.
		LinkDiscoveryManager	Manages link resources in the network and maintains link status in the OpenFlow network.
Regular Module	Application	TopologyService	Finds routes in the network, calculates network topology, and maintains topology information.
		RestApiServer	Provides REST API services.
		FlowCache	Integrates flow updates and searches across different modules.
		Forwarding	Implements packet forwarding between two devices.
		Firewall	Enforces access control on switches.
REST Module	Application	Circuit Pusher	Creates virtual links between two devices.
		OpenStack Quantum	Manages the OpenStack network.

2.2.3 NOX

The NOX platform is based on a publish-subscribe model[47], using the observer pattern[48]. Components on NOX can subscribe to events generated by the network, allowing users to write various components to manage the OpenFlow network[49]. Currently, components on NOX are divided into three categories: Core apps, Net apps, and Web apps. Core apps provide some basic applications that other components can use. Net apps are network control-related applications, while Web apps offer some interfaces to web services. The NOX platform is based on a publish-subscribe model, using the observer pattern. Components on NOX can subscribe to events generated by the network, allowing users to write various components to manage the OpenFlow network.

As shown in Figure 7, components on NOX are divided into three categories: Core apps, Net apps, and Web apps. Core apps provide some basic applications that other components can use. Net apps are network control-related applications, while Web apps offer some interfaces to web services. Events are generally generated in two ways: one is directly from OpenFlow messages, such as Datapath_join_event when a switch joins, and Datapath_leave_event when a switch leaves. The other is generated by controller applications, such as a user authentication component that generates a user authentication event when a user joins the network, which can then be used by other application components[50].

Figure 7 NOX Function Module Structure


Components
DSO(Dynamic Shared Object components)
Built-in Components
con1
con2
con3
con4
Distribute events to components: According to the priorities of the components handling each event in the nox.json configuration file, the handler functions within each component process the corresponding event.
Component context
Component Factory
kernel
Poll
Event Dispatcher
Oflib
OFP Packet
Event
OpenFlow Connection
OFP Packet
OFP Packet
When NOX starts, it installs components of type built-in components and DSO components.
Establish a TCP connection.

After a secure connection is established between the NOX controller and the underlying switches, the switches can send OpenFlow messages to the controller through this connection[51]. The OpenFlow protocol encapsulation and parsing module on the NOX controller encapsulates these messages, and the message distribution module delivers them to the upper-layer applications. Conversely, upper-layer applications can also send OpenFlow messages to the underlying switches via the OpenFlow protocol encapsulation and parsing module.

To develop new components on the NOX platform, it is essential to understand the basic structure of the components. A new component needs to inherit from the Component class and use REGISTER_COMPONENT to enable dynamic loading. During loading, the Configure and install methods are called to register events and their handlers. The events that the component needs to listen for are listed in the NOX.json file. Additionally, while creating the component, a meta.json file is needed to specify other components that the new component depends on. Events subscribed to by components in NOX.json are processed in a defined order. For example, a Datapath_join_event is first processed by the discovery component. If this component returns CONTINUE after processing, the event will be passed to the next component, mibtransport, for further processing. If discovery returns STOP, the event will not be passed further[52].

2.2.4 Mininet

Mininet is a virtualization network emulation tool developed by Stanford University based on the Linux Container architecture [53]. It can create a highly flexible custom virtual network consisting of hosts, switches, controllers, and links. Mininet uses the Linux [54] kernel to virtualize multiple hosts and simulates SDN switches using the OpenFlow protocol. The network topology can be defined and configured using Python [55] scripts, allowing users to quickly create custom topologies for testing and developing network applications.

Mininet offers three types of command parameters: network construction startup parameters, internal interactive commands, and external runtime parameters. Network construction startup parameters can be used to set the topology structure, switch types, and link attributes. Internal interactive commands allow interaction with virtual nodes, such as adding or removing nodes. External runtime parameters mainly control the runtime environment of Mininet, such as setting log outputs. These command parameters can be configured to set the topology structure, switch types, and link attributes, making it convenient for users to modify and configure the topology.

Additionally, Mininet provides many practical tools, such as traffic generators and packet capture tools, to facilitate network traffic monitoring and testing. One of Mininet's greatest advantages is its flexibility and customizability. It not only supports OpenFlow but also other southbound interface protocols, enabling the creation of highly controllable network environments. Therefore, Mininet has been widely used in network research, including areas such as network security and cloud computing. In the field of network security, Mininet can simulate various network attack and defense scenarios, making it convenient for researchers and security practitioners to conduct vulnerability testing and security analysis. In the realm of cloud computing, Mininet can be used to test and evaluate the performance and efficiency of various cloud computing solutions, helping users optimize cloud architectures and resource utilization.

2.2.5 Summary

This section underscores the diversity and adaptability of SDN controllers in meeting the needs of various networking scenarios, emphasizing their role in enabling efficient network management through centralized control mechanisms.

2.3 OPERATIONS OF SDN

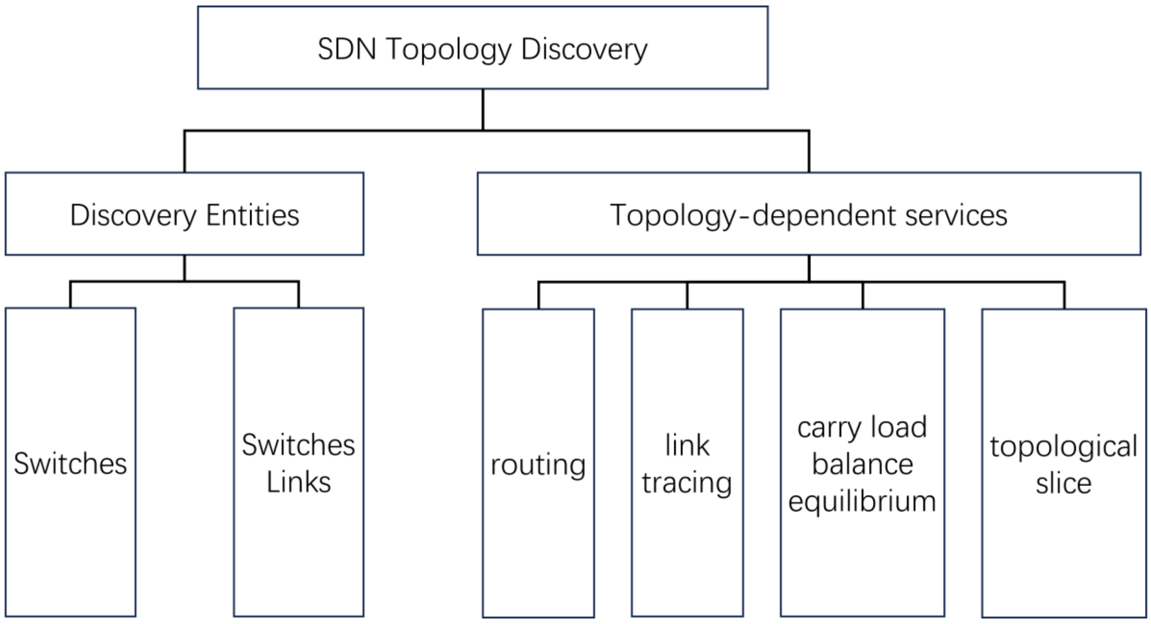
2.3.1 SDN Link Topology Discovery Technology

In the SDN network architecture, centralized and dynamic network topology information management technology [56] effectively decouples the control plane from the data plane. This is achieved through the use of a central controller that allows for highly flexible, real-time, and centralized control over the entire network structure. This management model endows the SDN controller with the ability to obtain and maintain a comprehensive view of the network, accurately describing the interconnections, topology, and traffic distribution among various devices in the network. When there are any changes in the network topology, such as device failures, link interruptions, or structural reorganizations, the SDN controller can promptly detect and update the global view, ensuring real-time synchronization of the network state. Through the topology discovery mechanism, the SDN controller can automatically identify various devices and link resources within the network, significantly reducing the burden of manual configuration.

Moreover, this centralized management approach enables the SDN controller to perform intelligent path calculations, selecting the optimal transmission path based on the current topology and traffic conditions, thereby achieving efficient traffic scheduling and maximizing network performance. Additionally, the scope of SDN topology information management includes timely response to various topology events, continuous network monitoring, and in-depth analysis. This provides multi-dimensional support for stable network operation, reliability assurance, and performance optimization, further highlighting the significant advantages of SDN in meeting modern network demands and enhancing network management efficiency[57].

The controller plays the role of storing information about core network components, including detailed locations of individual switches and the link parameters that form the topology data of interconnections between switches[58]. The controller aggregates network-wide topology information through active collection or passive reception and properly stores this data. Additionally, the controller adheres to a predefined update strategy, regularly refreshing and calibrating the maintained topology information to ensure real-time tracking and accurate understanding of network state changes, as illustrated in Figure 8.

Figure 8 Topological Discovery Classification



2.3.2 Link Topology Discovery Technology

The link topology discovery mechanism in SDN involves the controller identifying the link status between switches on the data plane[59]. By obtaining link connection status information, the controller can effectively support various network service functions. In Ethernet, link topology discovery typically relies on the Link Layer Discovery Protocol (LLDP)[60], where Ethernet switches exchange relevant link and port information. However, in the SDN architecture, since data plane switches are responsible only for data forwarding and lack logical control capabilities, the method for link topology discovery differs from traditional approaches. In SDN, this task is primarily undertaken by the controller, which not only stores link topology information but also performs detection tasks. Therefore, the OpenFlow Discovery Protocol (OFDP) [61] was specifically developed for SDN, based on LLDP. It utilizes LLDP packet contents and adapts to the separation of control and forwarding in SDN. Specifically, in SDN, the tasks of sending, receiving, and parsing LLDP packets are shifted to the controller level, rather than being executed by OpenFlow switches.

When a switch connects to the network, it sends an initialization signal to the controller, containing the OpenFlow version number and details of each port. Once the controller successfully responds and establishes a connection with the switch, it deploys topology discovery rules on the switch. First, upon receiving a Packet-out message from the controller, the switch forwards it to the connected switch port. Second, upon receiving a message from another switch, the switch adds its relevant information to the LLDP message, forming a Packet-in message, and forwards it to the controller. Third, switches not directly connected to the controller send the Packet-in to the neighboring switch[62].

Subsequently, the controller sends LLDP packets to each port of the switch. The switch, following the first rule, forwards this packet through the specified port. When the target neighboring switch receives the packet, it follows the second rule, encapsulating the LLDP content into a Packet-in message and adding its switch ID and receiving port ID information. The switch then sends this Packet-in message back to the controller. If the target switch is not directly connected to the controller, the Packet-in message follows the third rule, being forwarded step by step until it reaches the controller. Upon receiving such a Packet-in message, the controller parses the switch ID, port ID, and corresponding information contained in the LLDP frame, accurately mapping the switches and their connecting ports at both ends of the link, thereby updating its maintained network topology view.

2.3.3 Summary

This section highlights the capabilities of SDN to facilitate real-time, accurate network management and adjustments, which are essential for optimizing network performance and reliability.

2.4 SDN ROUTING

This section will separately introduce the SDN routing mechanism, the current research status of SDN intelligent routing optimization based on supervised learning, and SDN intelligent routing optimization based on reinforcement learning. On this basis, the research processes and advantages and disadvantages of these methods will be summarized.

2.4.1 SDN-Based Routing Mechanisms

Traditional routing technologies achieve the exchange and sharing of routing information through the establishment of routing tables and routing protocols[63]. In the SDN network architecture, the controller uses southbound interface protocols to uniformly distribute forwarding rules to switches, thus enabling routing transmission between switches. Based on the method of path transmission, SDN routing mechanisms can be divided into shortest path routing and multipath routing, as detailed below:

(1) Shortest Path Routing

Current mainstream SDN controllers, such as RYU[64], Floodlight[65], and POX[66], provide comprehensive data forwarding modules and typically use the Dijkstra[67] algorithm to find the shortest path. Data packets can be forwarded from the source node to the destination node using the shortest path determined by the Dijkstra algorithm. This method is simple and easy to implement. However, it overly relies on the shortest path for packet forwarding, which can lead to link congestion when multiple source nodes forward data packets to the same destination node, thereby reducing link utilization and failing to meet network performance requirements such as latency, bandwidth, throughput, and jitter.

(2) Multipath Routing

Multipath routing seeks to find multiple paths that meet the constraint conditions based on network traffic distribution and service traffic demands, and uses these paths for balanced transmission of network traffic. The goal is to improve network performance in terms of transmission delay, throughput, and link utilization, and to avoid link congestion. Li Daoquan et al. [68] proposes an SDN multipath routing load balancing strategy based on traffic distribution propensity. When data flows need to be transmitted, the strategy uses a depth-first traversal algorithm to obtain and store multipath information and the bandwidth and delay parameters of each path. According to custom traffic distribution propensity, it uses OpenFlow[69] group table technology to fairly distribute network traffic to each available path, effectively increasing the packet transmission volume of all available paths and reducing the load on single paths in the SDN network. Xiao Junbi et al. [70] proposes an SDN-based dynamic priority multipath scheduling algorithm. This algorithm formulates scheduling models for elephant flows and mouse flows based on traffic characteristics, combines group tables to optimize the communication mode between the controller and switches in the SDN architecture, reduces packet processing delay, and improves overall network performance. Zhou Jie[71] proposes an SDN-based multipath load balancing algorithm, which selects the optimal path based on link weights and traffic thresholds and makes real-time adjustments. Compared to traditional routing algorithms, it significantly improves bandwidth utilization and average delay. Most of the above SDN multipath routing studies are based on improvements to the Dijkstra algorithm or by considering the forwarding paths of all flows and selecting multiple paths with lower load for forwarding, resulting in large computational overhead and low efficiency, making them unsuitable for large network topologies.

2.4.2 SDN Routing Optimization Based on Supervised Learning

Supervised learning is a labeled learning technique that establishes a system model based on given data and labels, completing training based on the mapping relationship between input and output, and subsequently predicting results by inputting new data into the system model[72]. Common supervised learning methods include neural networks[73], Support Vector Machines (SVM)[74], K-Nearest Neighbor (KNN)[74], random forests[75], and decision trees[76]. SDN routing optimization based on supervised learning mainly adopts deep neural network methods, which input network topology status, traffic matrices, link utilization, delay, throughput, and other information into a deep neural network model. After the deep neural network is trained to converge, it outputs the decision results.

Raikar et al.[77] propose SDN routing optimization based on machine learning, using three different supervised learning models: SVM, nearest centroid, and naive Bayes for data traffic classification in SDN architecture applications. By capturing network traffic trajectories to generate traffic features and sending them to the classifier for prediction, the results show that the prediction accuracy of SVM is 92.3%, nearest centroid is 91.02%, and naive Bayes is 96.79%. Xin et al.[78] propose a novel incremental routing scheme that removes visited nodes during each node selection process, employing pointer networks and transformer attention models for research. By modifying a layer of attention, an approximate incremental transformer attention model is further proposed, which can be trained on larger network instances and outperforms advanced deep models with greedy decoding strategies. Zhang et al.[79]

study a routing decision scheme based on deep belief networks, used for backbone network routing optimization. Compared with traditional routing schemes, it converges faster and has lower information exchange costs. Modi and Swain[80] propose a deep learning routing algorithm based on CNN. This algorithm outputs intelligent paths by online training traffic patterns. Compared to the traditional routing algorithm OSPF, the average network throughput nearly doubled, and the average network throughput increased by about 40% compared to Artificial Neural Networks (ANNs). Additionally, the average network delay and packet loss rate are better than those of the ANN model. Tang et al.[81] propose an intelligent routing algorithm based on SDN's graph convolutional neural network, which improves network performance indicators such as delay and throughput to some extent.

The above-mentioned SDN routing optimization methods based on supervised learning, especially neural networks, have improved network performance such as transmission delay, throughput, and packet loss rate in SDN routing optimization. However, the training process requires a large amount of labeled data, which demands high computational complexity. The accuracy, generalization, and fault tolerance of routing still need improvement.

2.4.3 SDN Routing Optimization Based on Reinforcement Learning

Reinforcement learning is an important branch and effective tool of machine learning. Deep reinforcement learning is based on the fundamental theory of reinforcement learning, using deep neural networks to replace traditional decision functions, leveraging the powerful fitting capabilities of deep neural networks to train the learning process[82]. In the SDN routing optimization process based on reinforcement learning, network topology, traffic matrices, and other factors are regarded as network states, and link weights are considered as actions. Rewards are based on network performance optimization goals or user service quality, and through continuous training, a complete algorithm model is formed. When new data flows arrive, the optimal path meeting the optimization goals can be quickly calculated.

Yu et al.[83] propose a deep reinforcement learning mechanism for SDN called DROM. This mechanism improves throughput and reduces latency through continuous-time black-box optimization. Experimental results show that DROM has good convergence and effectiveness, providing better routing configuration than existing solutions. Xu et al.[84] propose a DRL-based routing method for experience-driven networks, DRL-TE, to solve traffic engineering problems. DRL-TE jointly learns the dynamic network environment and makes decisions under the guidance of deep neural networks. Experimental results show that DRL-TE is robust to network changes, significantly reducing end-to-end delay and continuously improving network utility while providing better throughput. Ding et al.[85] study a routing selection method based on deep reinforcement learning under large-scale network traffic, achieving minimized data transmission paths while avoiding link congestion and improving network throughput.

Sun et al.[86] propose an intelligent routing technology based on deep reinforcement learning called SmartPath. By dynamically collecting network states and using deep reinforcement learning to automatically generate routing policies, SmartPath ensures that routing policies can dynamically adapt to network traffic changes. Experimental results show that SmartPath can dynamically update network routing without relying on manual traffic modeling, reducing average end-to-end transmission delay by at least 10% compared to current optimal solutions. Hu et al.[87] propose an SDN-based intelligent-driven network architecture that optimizes network delay and throughput as objectives, effectively improving network load balancing compared to traditional routing algorithms OSPF and ECMP. Huang et al.[88] propose a near-optimal traffic control method for QoS optimization in SDN, using DRL algorithms to solve SDN multipath routing problems. By setting OpenFlow group bucket constraints to achieve traffic allocation ratios, the results show significant improvements in network QoS performance such as delay, jitter, and link utilization. Tang et al[89] proposes an SDN-based intelligent network energy consumption optimization method, which reduces network energy consumption by adjusting the activation and sleep of network devices through coordinated sleep techniques. However, this increases the load pressure on the control plane during device activation and sleep adjustments. Additionally, Yao et al.[90] propose an energy-saving topology optimization algorithm for control plane performance optimization, designing traffic-aware and device sleep techniques to align control plane load and data plane energy consumption, achieving an energy-saving topology. However, frequent switching of device activation and sleep in switches can cause some delay overhead.

The above intelligent routing algorithms have certain advantages in network performance such as delay, throughput, and link utilization when facing small network state inputs. However, in the complex network environment with continuously expanding network scale, these intelligent routing algorithms often have low convergence efficiency, and network performance such as average end-to-end delay and throughput still need improvement. Additionally, these intelligent routing algorithms have weak generalization capabilities when training experience is extended to different network topologies, making it difficult to cope with complex issues such as link interruptions and node failures. Current intelligent routing algorithms focus on load balancing routing optimization, often neglecting network energy-saving optimization, making it challenging to balance both network performance and energy-saving routing optimization.

2.4.4 SDN Routing Optimization Based on Deep Reinforcement Learning Algorithms

Zhao et al.[91] designed an intelligent routing method based on deep reinforcement learning, which effectively alleviates network congestion and achieves network load balancing. Chen et al.[92] addressed the issue of modeling complex and dynamic networks by proposing a deep reinforcement learning algorithm based on DDPG (Deep Deterministic Policy Gradient). This algorithm divides the network into uplink and downlink, introducing multiple new features to form the state space. The action space consists of the intersections of paths between source and destination switches in the uplink and downlink networks. The reward function optimizes the delay and throughput of the uplink and downlink networks, achieving optimal routing decisions in SDN traffic engineering. Considering that traditional traffic engineering requires rerouting as many data flows as possible to ensure optimal network performance, but frequent rerouting brings network interference and other issues, Zhang et al.[93] proposed a critical flow rerouting reinforcement learning algorithm (CFR-RL) based on the actor-critic framework.

The CFR-RL algorithm selects some critical flows for rerouting, while most flows are forwarded by equal-cost multi-path (ECMP), effectively solving the problem of decreased network service quality and interference caused by frequent rerouting. Fu et al.[93] proposed a deep Q-learning reinforcement learning method to achieve low latency and low packet loss for mouse flows and high throughput and low packet loss for elephant flows in data center networks. Liu et al.[94] designed a deep reinforcement learning routing algorithm, considering the SDN controller cache as a key factor affecting routing strategies. By restructuring the cache and bandwidth with quantifiable scores to reduce latency, this algorithm forms a multi-dimensional state space, improving network throughput and robustness. Hossain et al.[95] designed an intelligent situational awareness routing algorithm that uses intelligent sensing algorithms to reduce the impact on application-driven program QoS when the network is under attack, effectively enhancing network reliability. Yu et al.[96] designed a customizable, adaptive routing optimization strategy based on deep reinforcement learning mechanisms and black-box techniques, effectively reducing network management and maintenance costs.

By combining deep learning and reinforcement learning techniques, the previously mentioned algorithms efficiently overcome the drawbacks of Q-table-based approaches. They also speed up model convergence and improve the system's capacity to manage and adjust to intricate, high-dimensional dynamic network environments, ultimately leading to improved network performance. These techniques, however, do not take into account how intelligent routing optimization algorithms might be affected by alterations in network traffic conditions in the future. Additionally, some techniques produce link weight values as actions, necessitating additional computations to determine the routing paths.

2.4.5 Summary

Section 2.4 addresses the strategies and technologies used in SDN routing, including the implementation of machine learning techniques to enhance routing decisions.

2.5 CHAPTER SUMMARY

This chapter has systematically explored the various dimensions of Software Defined Networking (SDN), from its fundamental architecture to advanced routing optimization techniques. SDN's flexibility and efficiency over traditional network architectures are evident, with its ability to adapt quickly to new business demands and manage network traffic dynamically. The exploration of SDN controllers and their distinctive features underscores the diversity of options available for network customization and optimization. The discussion on SDN routing strategies, particularly those leveraging machine learning, reveals a promising direction for future network management, focusing on automation and intelligent decision-making to enhance performance metrics.

This chapter addresses the limitations of traditional routing methods that rely on limited network link information for routing decisions, have poor adaptability to dynamic and complex network changes, and lack flexibility in adjusting routing strategies. The method it suggests is based on Dueling DQN reinforcement learning and network traffic state prediction (DRL-TP, Deep Reinforcement Learning-Network Traffic State Prediction), and it is an SDN intelligent routing technique. By acquiring global network link state information, this method achieves real-time adaptive generation of optimal routing decisions.

CHAPTER 3 - RESEARCH METHODOLOGY

This section outlines the experimental and theoretical methods employed to assess the performance and efficacy of Dueling DQN and GRU-based SDN routing strategies.

3.1 RESEARCH METHOD

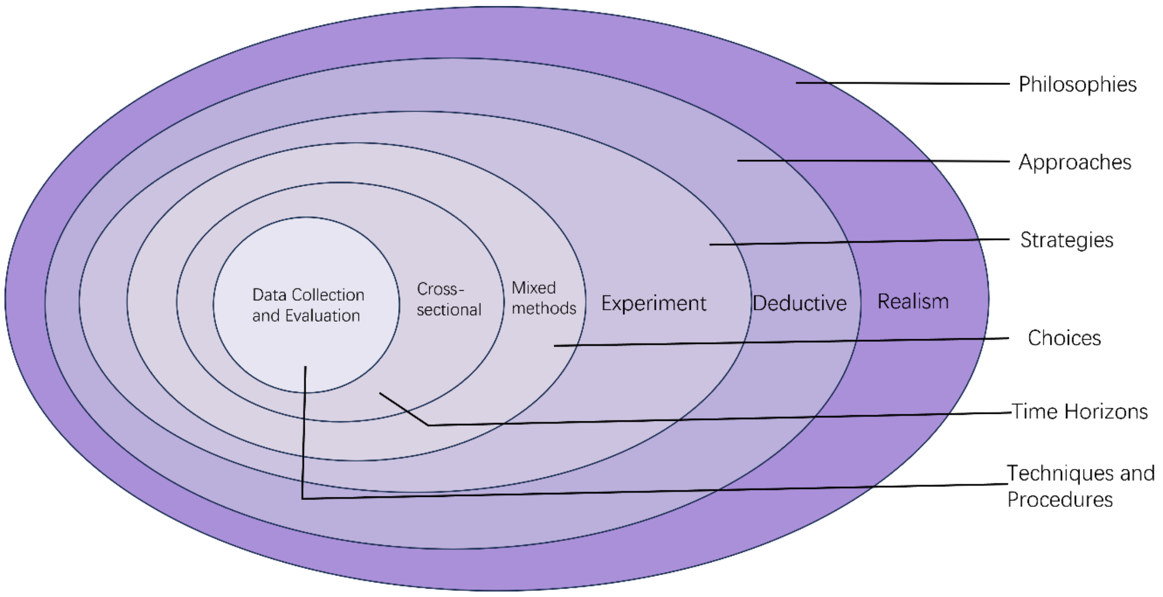
In this project, we employ a multifaceted research method that integrates Software Defined Networking (SDN) and advanced reinforcement learning techniques, specifically Dueling Deep Q-Networks (Dueling DQN), to enhance routing decisions through intelligent traffic prediction. Our project design capitalizes on the flexibility of SDN which separates the control and data planes, enabling centralized network traffic management. The Dueling DQN approach optimizes routing by distinguishing between state values and the advantages of specific actions, thus improving decision-making in dynamic network conditions.

We conduct experiments in a simulated network environment using tools like Mininet and the Ryu SDN controller, which facilitate the testing of our model under various traffic scenarios. Traffic matrices are collected to provide real-time and historical data for training the Dueling DQN model and evaluating network performance against traditional and other RL-based methods. The effectiveness of our proposed method is measured through key performance indicators including throughput, latency, and packet loss.

The research strategy is experimental. Our results are analyzed to refine the Dueling DQN model, ensuring it effectively aligns with actual network dynamics. Additionally, scalability and robustness tests are carried out to confirm the model's efficacy in larger and more complex networks, as well as its resilience in adverse network conditions. This comprehensive approach not only advances the field of network management but also offers practical insights into the deployment of machine learning techniques within SDN frameworks for real-time adaptive network optimization.

The research onion framework illustrated in the Figure 9 encapsulates the comprehensive methodology adopted for this project, structured across several layers. Each layer represents a specific stage of the research process, detailing the underlying philosophies, approaches, strategies, choices, time horizons, and techniques and procedures.

Figure 9 Layers of the Onion Diagram for Research Methodology



3.1.1 Philosophy

The project adopts a realism approach, suitable for quantitative analysis of observable phenomena within predefined frameworks. The research utilizes this philosophy to leverage its structured, objective nature, ensuring rigorous quantification of network performance metrics. Challenges associated with a potentially narrow scope are mitigated by incorporating diverse network scenarios.

3.1.2 Approaches

The research follows a deductive approach, starting with the hypothesis that the Dueling DQN-based intelligent routing method will outperform traditional routing methods. The hypothesis is then tested through systematic experiments.

3.1.3 Strategies

Experiment: The primary strategy involves conducting experiments to collect performance data of different routing algorithms under various network conditions. These experiments are designed to provide empirical evidence supporting the hypothesis.

Software Development: A significant part of the research involves developing software for implementing and testing the proposed DRL-TP intelligent routing algorithm.

3.1.4 Choices

Mixed Methods: Although primarily quantitative, the project also involves some qualitative assessment of the algorithms' performance to provide a comprehensive evaluation.

3.1.5 Time Horizons

Cross-sectional: The experiments are conducted at specific intervals, providing snapshots of the network performance at various points in time. This approach helps in understanding the immediate impact of the routing algorithms.

3.1.6 Techniques and Procedures

Data Collection and Evaluation: Data is collected through network simulations, using tools like Mininet and Ryu to create the SDN environment and Iperf to generate traffic. The collected data includes metrics such as throughput, delay, and packet loss rate.

3.1.7 Contingency Plans

Alternative Algorithms: In case the Dueling DQN-based approach does not perform as expected, alternative DRL algorithms such as PPO (Proximal Policy Optimization) or A3C (Asynchronous Advantage Actor-Critic) will be considered.

Extended Data Collection: If initial data collection proves insufficient or inconclusive, additional data collection phases will be implemented to ensure robust and comprehensive results.

Hybrid Methods: Combining DRL with other machine learning techniques, such as supervised learning for specific sub-tasks, to enhance overall performance.

Simulation Environment Adjustments: Modifying the network simulation environment to include different types of network traffic and topologies to test the robustness of the proposed routing algorithm under various conditions.

Expert Review: Engaging domain experts to review methodology and results, providing insights and recommendations to address potential issues and improve the research approach.

3.1.8 Risks and Limitations

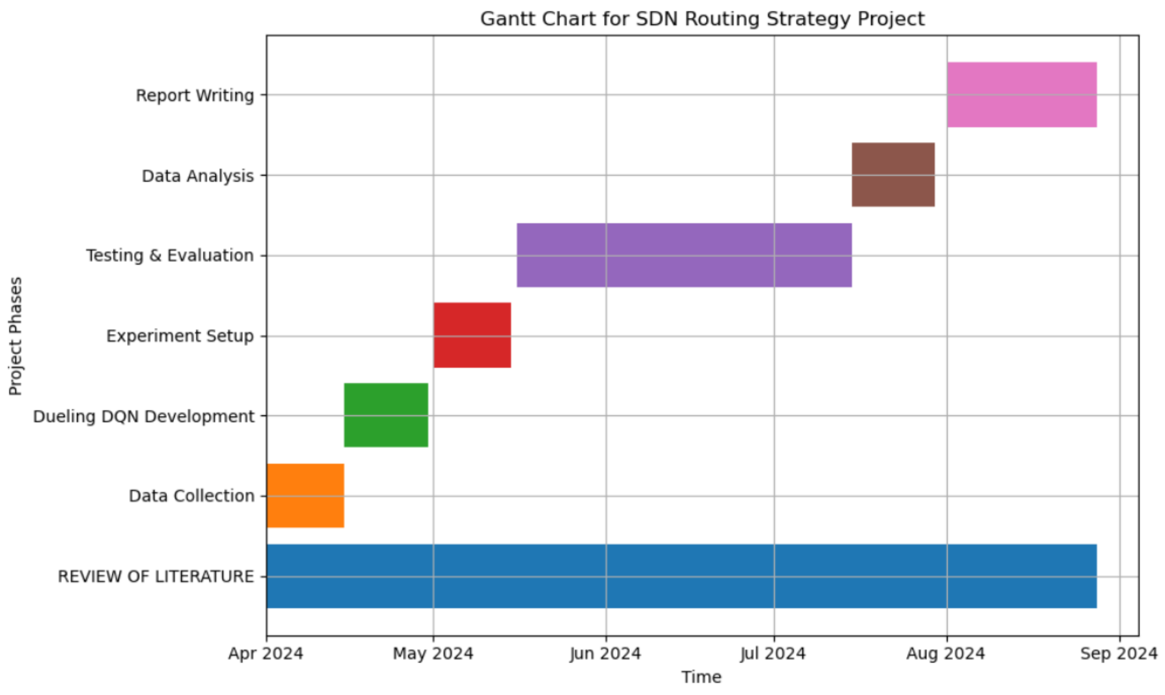
The main risks involve the potential discrepancies between simulated environments and real-world network operations. Strategies to counteract these risks include rigorous scenario testing and validation against baseline models.

Validity, Reliability, Generalisability

- Validity: The experimental setup is designed to accurately reflect realistic network behaviors.
- Reliability: Consistency of results will be ensured through replication of experiments and methodological transparency.
- Generalisability: Results will be discussed in terms of their applicability to similar technological environments and configurations.

The Gantt chart for the SDN Routing Strategy Project, as shown in Figure 10, visualizes the project timeline and the scheduling of different phases from April 2024 through September 2024. The project begins with Data Collection in April, followed by the development of the Dueling DQN model in late April and early May. Experiment setup occurs briefly in mid-May. Testing and evaluation is the longest phase, starting in late May and continuing through mid-July. Data Analysis is scheduled for July, overlapping slightly with the end of testing. Finally, Report Writing is planned for late August into early September, concluding the project's scheduled activities. Each phase is color-coded, allowing for a clear visual representation of the project's progression over the specified months.

Figure 10 Gannt Chart



3.2 RESEARCH MATERIALS

3.2.1 Hardware Devices

Computer Cluster:

The experiments were conducted on a cluster composed of multiple high-performance computers, each equipped with an Intel Core i9 processor and 32GB RAM. These computers were interconnected via Gigabit Ethernet to simulate high traffic and high throughput conditions in a real network environment. The high processing power and large memory capacity of the computer cluster ensure that complex network simulations and large-scale data processing can be performed efficiently.

3.2.2 Network Devices

OpenFlow Switches:

Hardware switches supporting the OpenFlow protocol were used to construct the experimental network topology. These switches are highly programmable and can flexibly forward and process data packets according to instructions from the control plane. By using OpenFlow switches, precise control and management of network traffic can be achieved, effectively verifying the performance of the intelligent routing algorithm under various network conditions.

3.3 CHAPTER SUMMARY

This chapter provided a detailed description of the research materials, including the datasets, software, and hardware used in this project. The justification for the chosen materials emphasized their relevance and suitability for the research objectives, ensuring that the project is based on high-quality, reliable data and state-of-the-art computational tools. Ethical considerations were addressed, ensuring compliance with data usage guidelines and ethical standards. This comprehensive approach to selecting and utilizing research materials sets a solid foundation for the subsequent chapters on the design, implementation, and evaluation of the proposed model.

CHAPTER 4 – ARCHITECTURAL DESIGN AND MODELING

4.1 ARCHITECTURE AND MODEL DESIGN"

This section introduces the architecture and modeling of SDN-based intelligent routing optimization, as well as the detailed design process of intelligent routing algorithms.

4.1.1 Parameter Collection Design

OpenFlow switches are equipped with various counters to record statistical information such as the number of different types of packets, byte counts, and time information, as shown in Table 3. These counters include per-port, per-flow table, and per-flow entry statistics. The controller can periodically query and retrieve counter statistics from OpenFlow switches using statistics messages defined by the OpenFlow protocol. This statistical information is very useful for network performance monitoring and troubleshooting. For example, administrators can check whether the input and output byte counts on a switch port are equal to determine if there is congestion in the network. These statistics can also be used to measure network performance metrics such as packet loss rate and throughput.

OpenFlow statistics messages include Port-Stats messages, Flow-Stats messages, Aggregate-Stats messages, Queue-Stats messages, Group-Stats messages, Meter-Stats messages, and Table-Stats messages. These messages can be used to obtain statistical information from specific counters in OpenFlow switches. For instance, Port-Stats messages can be used to obtain statistics for specific physical ports of an OpenFlow switch, while Flow-Stats messages can be used to obtain statistics for specific flow entries.

Table 3 Counters in OpenFlow

Type	Content	Bit Width
Per Flow Table	Active Entries	32
	Packet Lookups	64
	Packet Matches	64
Per Flow Entry	Received Packet Count	64
	Received Packet Byte Count	64
	Duration (seconds)	32
	Duration (nanoseconds)	32

4.1.1.1 Measuring link packet loss and throughput

Port-Stats messages can be utilized to measure link packet loss rate and link throughput. There are two types of Port-Stats messages: Port-Stats-Request messages, which are used by the SDN controller to request port statistics from the switch, and Port-Stats-Reply messages, which are used by the switch to respond to the SDN controller. Specifically, the switch reads the counters of the specified port, obtains the port's statistics, encapsulates them in the message, and then sends the message to the SDN controller. In OpenFlow 1.3, the formats of Port-Stats-Request and Port-Stats-Reply messages are shown in Figure 11 and Figure 12, respectively. The controller can obtain the port statistics of an OpenFlow switch by sending a Port-Stats-Request message, and the switch will reply with a Port-Stats-Reply message. The Port-Stats-Reply message includes statistics for each port, such as the number of received/transmitted packets (rx_packets/tx_packets), the number of received/transmitted bytes (rx_bytes/tx_bytes), the number of dropped packets (rx_dropped/tx_dropped), the number of collisions (collisions), and the port's duration (duration sec and duration nsec). These statistics are used to calculate the link packet loss rate and link throughput. By querying these statistics, the controller can monitor and manage the network state.

Figure 11 Port-Stats-Request Message Format in OpenFlow 1.3

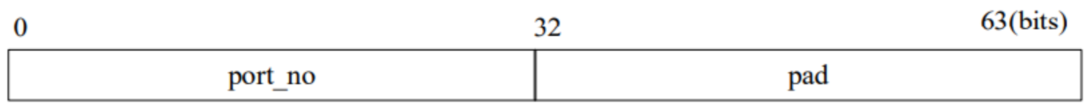
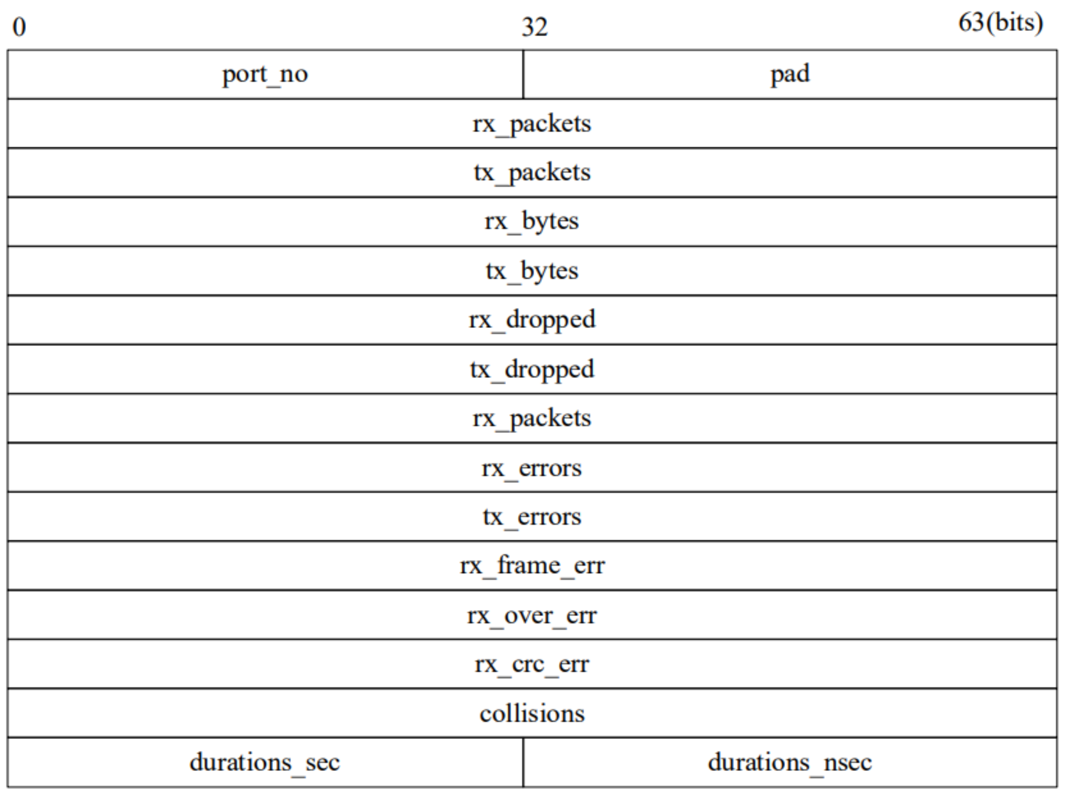
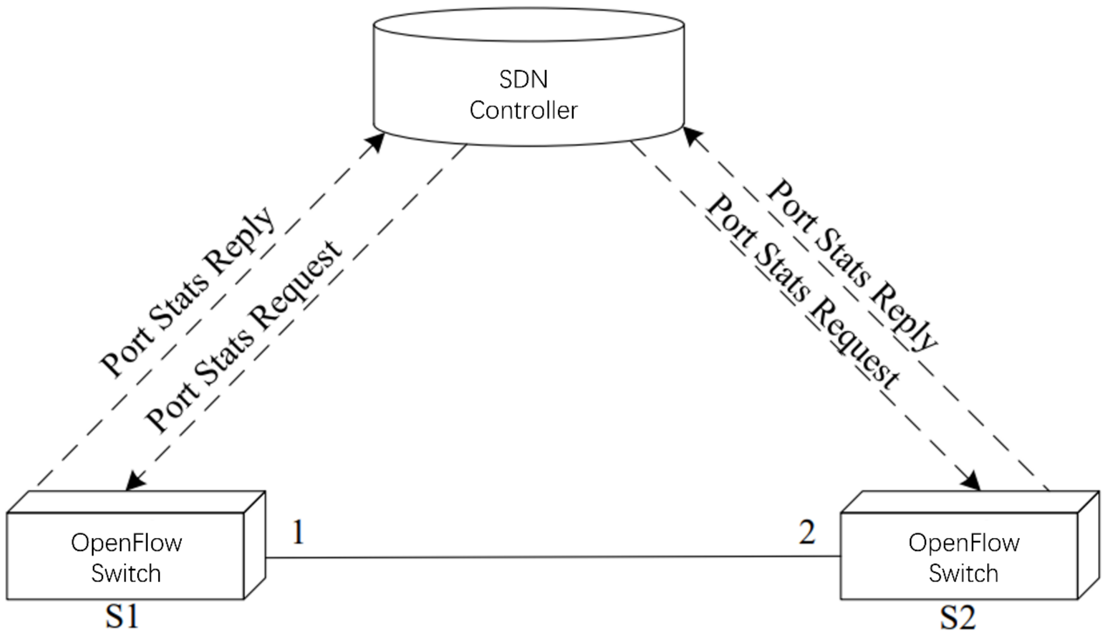


Figure 12 Port-Stats-Reply Message Format in OpenFlow 1.3



Packet loss rate is a crucial indicator of network performance. It can be used to assess the quality and stability of the network, as well as for troubleshooting and performance optimization.

Figure 13 Schematic diagram of OpenFlow-based Port-Stats messages to measure link packet loss rate



Packet loss rate is a crucial indicator of network performance. It can be used to assess the quality and stability of the network, as well as for troubleshooting and performance optimization. Suppose we measure the packet loss rate of the link from switch S1 to switch S2 in the network topology shown in Figure 13. To obtain the number of packets sent by port 1 of S1 ($s1\_\mathrm{tx\_packets}_{s1}$) and the number of packets received by port 2 of S2 ($s2\_rx\_packets_{s2}$), the OpenFlow-defined Port-Stats-Request statistic message can be used. The controller can periodically send statistic requests to the OpenFlow switches to retrieve the statistics of the specified ports. Then, the packet loss rate over the interval between two query periods can be calculated using Equation (1). To achieve periodic polling, a timer can be used to set the query interval, triggering the query operation when the timer expires.

$$Loss(i-1,i) = 1 - \frac{rx\_\mathrm{p}ackets_{s2(i)} - rx\_\mathrm{p}ackets_{s2(i-1)}}{\mathrm{tx}\_\mathrm{p}ackets_{s1(i)} - \mathrm{tx}\_\mathrm{p}ackets_{s1(i-1)}} \quad (1)$$

In the formula, Loss(i-1,i) represents the packet loss rate between the (i-1)th and ith query intervals; $rx\_packets_{s2(i-1)}$ *denotes the number of packets received by switch S2 at the ith query;* $tx\_packets_{sI(i)}$ *denotes the number of packets received by switch S2 at the (i-1)th query;* $tx\_packets_{sI(i)}$ *denotes the number of packets sent by switch*

S1 at the i^th query; $tx\_packets_{s1(i-1)}$*denotes the number of packets sent by switch S1 at the (i-1)^th query.*

By measuring throughput, the transmission capacity, data processing capability, and transmission quality of the network can be evaluated, thus determining whether the network meets business requirements. When measuring the link throughput of switches, the SDN controller periodically sends Port-Stats-Request messages to the specified switches and retrieves the received/sent byte counts (rx_bytes/tx_bytes) and port duration (duration_sec and duration_nsec) from the switches' Port-Stats-Reply messages. Using formulas (2) *and* (3)*, the throughput during the (i-1)th and ith query intervals can be calculated.*

$$duration = duration\_sec + duration\_nsec * 10^{-9} \quad (2)$$

In these formulas, duration refers to the port's duration; duration_sec represents the port's duration in seconds; duration_nsec represents the port's duration in milliseconds.

$$Throughtput(i-1,i) = \frac{bytes_i - bytes_{(i-1)}}{duration_i - duration_{(i-1)}} \quad (3)$$

In the formula, Throughput(i-1,i) represents the throughput during the (i-1)th and ith query intervals; $bytes_i$ denotes the total number of received and transmitted bytes at the ith query; $bytes_{(i-1)}$ denotes the total number of received and transmitted bytes at the (i-1)th query.

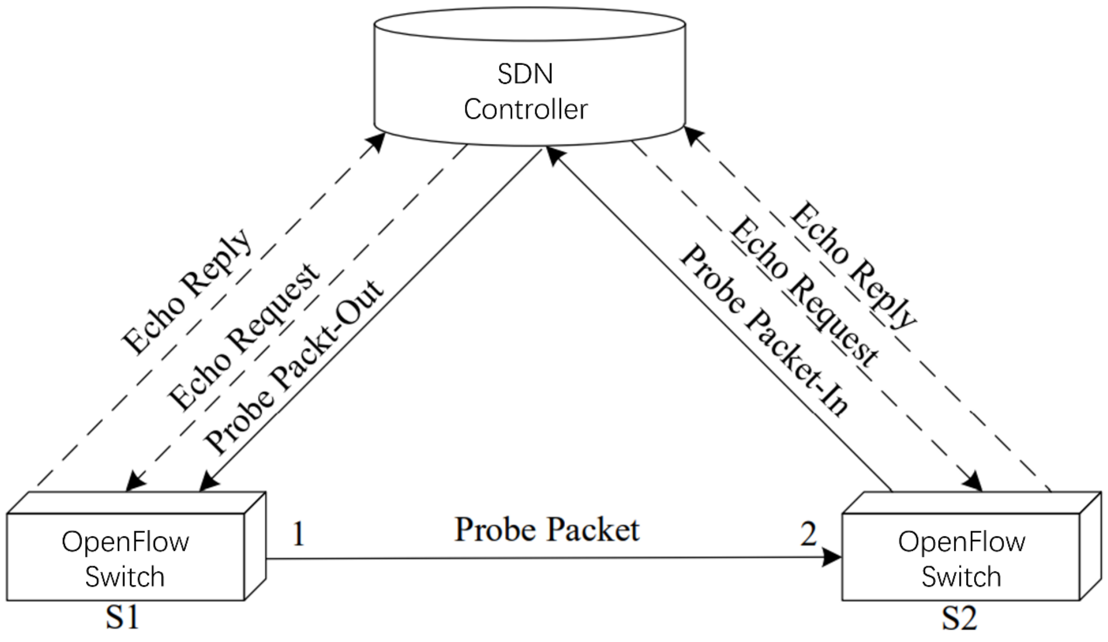
4.1.1.2 Measuring link latency

In an SDN network, an important metric for evaluating link performance is the transmission latency between switches. However, because OpenFlow switches do not include timestamps in regular packets, it is not possible to measure transmission latency passively as in traditional IP networks. Therefore, an active measurement method is required, which involves generating and sending probe packets between switches to address this issue. These probe packets contain information about the sending and receiving times, which can be used to calculate the transmission latency between switches. Common probe packets include Ping packets, Traceroute packets, and others. By sending these probe packets and collecting their responses, transmission latency information between switches can be obtained, allowing for network performance monitoring and optimization.

Measurement of latency in software-defined data center networks using Packet-Out and Packet-In messages operates on the principle illustrated in Figure 14 Schematic for measuring delay based on Packet-Out and Packet-In messages. The controller sends a probe packet to switch S1 and issues a rule for S1 to forward the probe packet to S2. If S2 receives the probe packet but does not have a corresponding forwarding rule, it will return the probe packet to the controller. By calculating the total transmission time of the probe packet in the path, the controller can determine the transmission latency of the link from S1 to S2. Compared to the passive measurement methods used in traditional IP networks, this method allows for more accurate measurement of the transmission latency between switches.

To measure the transmission latency between switches, the controller sends probe packets and measures the total time these packets take to travel through the path. However, because there is also latency in communication between the controller and the switches, it is necessary to send communication messages and measure the round-trip time (RTT) between the controller and each switch. Finally, by calculating the difference between these times, the controller can obtain the final link latency result.

Figure 14 Schematic for measuring delay based on Packet-Out and Packet-In messages



The detailed steps to measure the latency of the link from S1 to S2 are as follows: a) Probe Packet Transmission Time T_{travel}:To obtain the probe packet transmission time T_{travel}, the controller generates a probe packet containing the target forwarding port of switch S1 and the sending timestamp, recording the transmission path and sending time. These packets are encapsulated in a Packet-Out message and sent to switch S1. Switch S1 forwards the packet to the designated port, from where switch S2 receives it. When S2 receives the probe packet, the absence of a matching flow entry triggers a Packet-In message, encapsulating the probe packet and returning it to the controller. The controller calculates the total transmission time T_{travel} in the "controller—S1—S2—controller" path using the probe packet's timestamp and the time when the probe packet is received. b) Round-Trip Time (RTT) Measurement Between Controller and Switches: The controller generates an Echo Request message and sends it to switches S1 and S2, waiting for Echo Reply messages and recording the timestamps when the messages are received. Finally, it calculates the RTT. c) Link Latency Calculation: Based on the above measurements, we can use Equation (4) to calculate the latency between switches S1 and S2. This latency can be represented as the total time T_{delay} for the probe packet to be issued by the controller to S1, then from T_{delay}, and finally back to the controller.

$$T_{delay} = T_{travel} - \frac{RTT_{S1}+RTT_{S2}}{2} \quad (4)$$

4.1.2 Intelligent routing algorithms

4.1.2.1 Deep Reinforcement Learning Algorithms

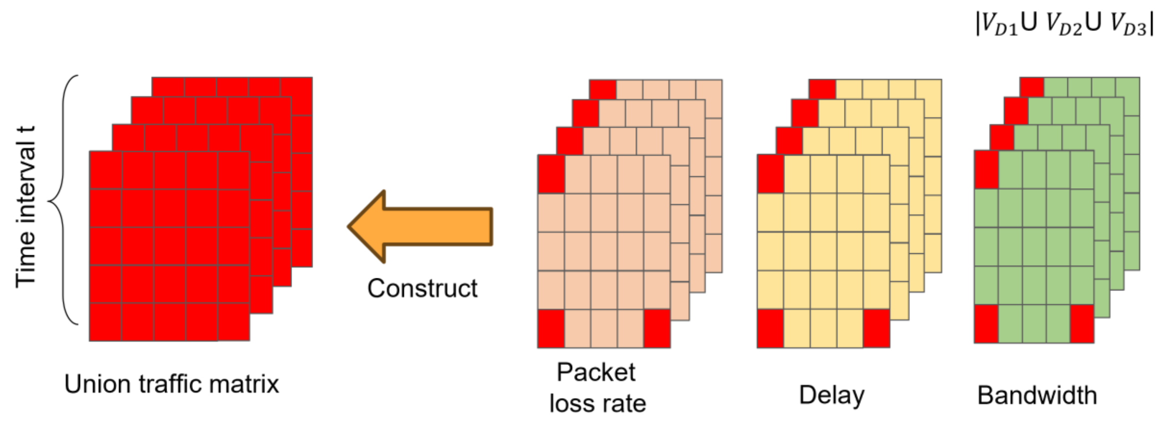
The framework of deep reinforcement learning (DRL) algorithms necessitates the creation of distinct state spaces, action spaces, and reward functions for a range of issues and use cases. The state space, action space, and reward function designs for the DRL-TP intelligent routing algorithm—which is based on the deep reinforcement learning framework—are explained in the following.

State Space (S): The traffic matrix over interval t is shown by the symbol TM in the representation of the state space, S=TM. Equation illustrates that this matrix is made up of numerous two-dimensional matrices $M_{|V|\times|V|}$.

$$m_{ij} = w_1 \bullet \frac{1}{L_{tw_{ij}}} + w_2 \bullet L_{delay_{ij}} + w_3 \bullet L_{loss_{ij}} \quad , i,j = 1,2,\dots,|V| \; (5)$$

Each element m_{ij} in the traffic matrix is constructed by aggregating information from six aspects of the network link: residual bandwidth (L_bw), delay (L_{delay}), packet loss rate (L_{loss}). These elements are combined using adjustable parameters ; $w_l \in [0,1], l = 1,2,3$, which serve as weight factors for each component. Each network link information matrix includes link information between all switch nodes at the current time. The indices i and j represent the switch node names in the network topology, and $|V|$ denotes the number of switch nodes in the network topology. The structure of the traffic matrix is illustrated in Figure 15 Traffic matrix structure diagram.

Figure 15 Traffic matrix structure diagram



Forwarding link weights and forwarding paths make up the two main categories of actions in the action space. While storing a sizable action space is not necessary for the former, it still needs to be further transformed into forwarding paths using the appropriate techniques. The latter involves storing a huge action space, but it outputs forwarding paths immediately. Choosing a set of potential paths to serve as the action space is an efficient solution, as proven effective in studies [26][31][33][43]. The action space designed in this chapter improves upon the latter approach, directly outputting forwarding paths. Each action $a_t \in [0,1,\dots,k]$ corresponds to the selection of a forwarding path in state $s_t \in S$, where $= \lceil 0.1 \cdot |V| \cdot |V| \rceil$ candidate path matrices $C_{|V|\propto|V|}$ are composed. The paths from every source switch node to every destination switch node are included in each potential path matrix. From switch node i to switch node j, the $path_ij = [i, \dots, j]$ is represented by the entries in each potential path matrix.

The reward value is used to provide feedback on the quality of actions supplied by the neural network, typically evaluating the current network conditions and the actions taken by the agent. It can be set to optimize various objective functions as needed. In this method, average end-to-end delay, bandwith, and packet loss rate are used as the comprehensive evaluation metrics. The reward value is calculated as shown in Equation (6):

$$R = \varphi_1 \bullet L_{bw} - \varphi_2 \bullet L_{delay} - \varphi_3 \bullet L_{loss} \quad (6)$$

In Equation (6), φ_1, φ_2, and φ_3 are weight parameters, each ranging from 0 to 1. The calculation process can adjust these weights according to the importance of each performance metric. After calculating the reward value, the result is returned to the agent, which then adjusts the multipath routing link weights and traffic splitting ratios. During the model training convergence process, the reward value is accumulated over the increasing number of training steps. The rising trend of accumulated reward values and the total reward value can be used to judge the convergence efficiency of the training model.

```
Algorithm 1: DQN Deep Reinforcement Learning Algorithm
Input: Traffic matrix: TM
Output: Forwarding paths for all source-destination pairs in the network
1. Initialize policy network Q_policy and target network Q_target with weights θ,
and experience pool M
2. For episode = 1 to episodes do:
3.     The agent obtains the initial state $S_t$
4.     While next_state $S_{t+1}$ is not final state do:
5.         Update exploration parameter ε = ε - (steps * decay)
6.         The agent selects action $a_t$ for current state $s_t$ based on:
7.         The estimated reward $R(s_t, a_t)$
8.         Store experience Experiences = $(s_t, a_t, r_t, s_{t+1})$ into M
9.         If len(M) >= batch then:
10.            Sample batch data randomly from M
11.            Calculate $p_{value}$ and $t_{value}$ for the batch
12.           Execute gradient descent on $(t_{value} - p_{value})^2$ to update Q_policy weights
θ
13.        If steps % freq == 0 then:
```

```
14.         Update Q_target network model parameters, θ_{target} ← τ * θ_{policy} + (1-τ) * θ_{target}
15.       End if
16.       s_t ← s_{t+1}
17.    End while
18. End for
```

4.1.3 GRU

Gated Recurrent Unit (GRU) [97] is a variant of the Long Short-Term Memory (LSTM) network. As shown inFigure 16 GRU structure diagram, the GRU consists of two special gates: the update gate and the reset gate. The GRU network model contains fewer parameters than the LSTM network model, which not only lowers the possibility of overfitting in the prediction model but also speeds up its convergence. This makes it more suitable to satisfy the intelligent routing algorithm's real-time traffic matrix acquisition needs, which are the focus of this research. As a result, GRU is used in this project as the network traffic state prediction model. The main functions of the two special gates in GRU are introduced below. The update gate z $_t$ can be understood as a combination of the forget gate and the input gate in LSTM. It determines which state information should be discarded or retained and the importance of that state information. As shown in equation (7), the update gate z$_t$ takes the state information from the hidden layer at the previous time step and the current time step's input layer information $[h_{t-1}, X_t]$ and, through the sigmoid function σ, outputs a value between [0,1] to decide the extent to which the information in the cell state z_t should be retained.

$$z_t = \sigma(W_z \bullet [h_{t-1}, X_t] + \omega_z) \quad (7)$$

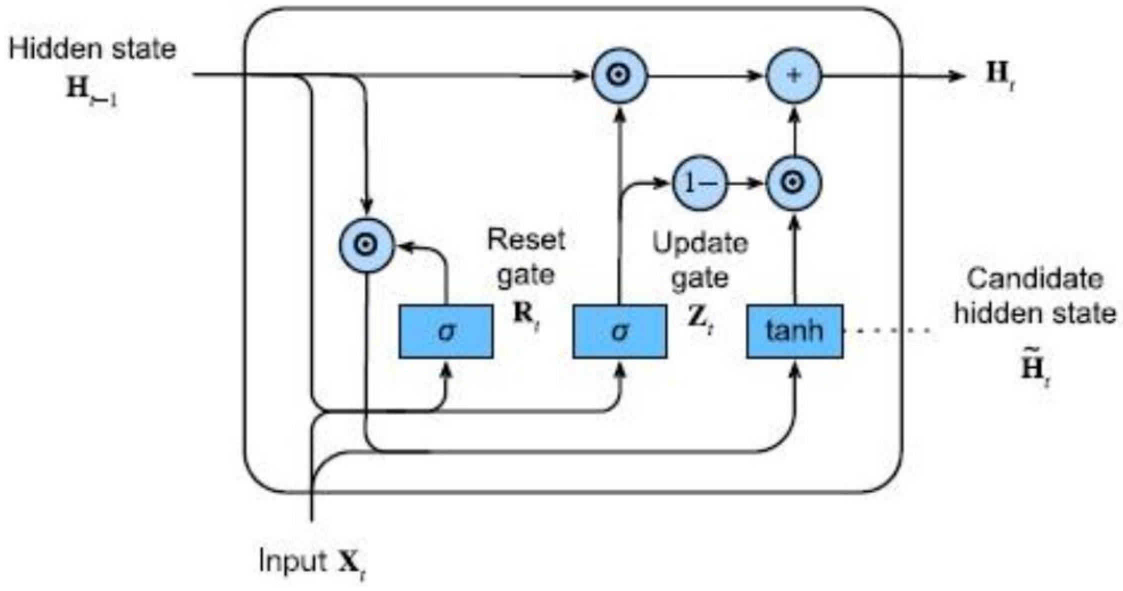
The reset gate r_t functions similarly to the update gate and is used to determine the next hidden state r_t. First, according to equation (8), it outputs a value between [0,1] to decide the retention level of r_t. Then, r_t is used to reset the previous hidden state $h_t - 1$ to obtain the candidate hidden state, as shown in equation (9). The symbol ⨀ denotes element-wise multiplication of the corresponding values in the matrices, followed by summation. Finally, according to equation (10), the next hidden state h_t is obtained and passed to the neurons in the next time step. This process continually updates the weight parameters of the GRU network model, thereby enhancing its ability to predict the network traffic state.

$$r_t = \sigma(\omega_r \cdot [h_{t-1}, X_t]) \quad (8)$$

$$\tilde{h}_t = \tanh(\omega_h \cdot [r_t \odot h_{t-1}, X_t]) \quad (9)$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \quad (10)$$

Figure 16 GRU structure diagram



Start by setting the essential hyperparameters, including the dimensions for the input, hidden, and output layers which are fundamental to the GRU algorithm's structure. The traffic matrix is then processed through time-series operations to generate input and target matrices, TM_{input} and M_{target} for the GRU model. Gradient descent and backpropagation are used to update the weights and biases of the GRU model during training, and the mean squared loss function is employed to assess the model's effectiveness. Upon completion, the refined GRU model is employed to produce a forecasted traffic matrix, integral to the traffic matrix TM in Algorithm 1, thus enhancing the algorithm’s predictive accuracy and efficacy.

```
Algorithm 2: Network traffic status prediction algorithm
Input: Traffic matrix: TM
Output: Forwarding paths for all source-destination pairs in the network
1. Initialize GRU model weights.
```

```
2. Split the time series data TM into input (TM_input) and target (TM_target) for
training.
3. For each episode from 1 to n:
4.   Initialize hidden state.
5.   For each time step from 0 to the length of TM_input:
6.     Calculate output and update hidden state using the GRU model.
7.     Compute loss as the difference between model output and the target data.
8.     Update the model to minimize loss.
9.   End time step loop.
10. End episode loop.
```

4.2 DATA EVALUATION METHODS

4.2.1 Data Collection

The process of data collection involves gathering performance data from network simulations to ensure a comprehensive understanding of network behavior and performance under various conditions. In this study, data is collected using a structured approach. The experimental environment is constructed using Mininet 2.3.0 to build the SDN topology, with Ryu 4.34 serving as the SDN controller and Iperf for simulating data flows. The network setup includes a modified New York City Center network topology consisting of 14 nodes, each representing a switch supporting the OpenFlow protocol, with hosts mounted under each switch. The transmission links between switches have bandwidths randomly set between 15 and 100 Mbit to simulate a heterogeneous network environment.

In order to quantify the network traffic matrix, data flows of evenly dispersed sizes are generated and sent with equal probability, yielding a total of 1458 traffic matrices. These matrices provide a comprehensive dataset for analysis, encompassing a variety of network link indicators such as bandwidth, delay, packet loss rate, used bandwidth, number of packet dropouts, and error rate. Throughput, latency, and packet loss rate are important performance data that are gathered and are essential for assessing how well the routing algorithms being studied work.

4.2.2 Data Evaluation

In this project, data evaluation involves analyzing the collected performance data to assess the effectiveness of different routing algorithms within a simulated network environment. The primary performance metrics considered are throughput, delay, and packet loss rate, which provide a comprehensive view of the network's operational efficiency and reliability under various routing strategies.

Throughput is evaluated by measuring the total amount of data successfully transmitted across the network within a specified time period. This metric is crucial for understanding the network's capacity to handle high volumes of traffic and is calculated as the number of bytes sent from one switch to another, divided by the available bandwidth of the link. Higher throughput indicates better network performance and more efficient data handling.

Delay is another critical metric, reflecting the time taken for data packets to travel from the source to the destination. It is measured using the SDN controller's link discovery protocol, which sends echo messages to switches to obtain timestamps. These timestamps help calculate the total transmission time for data packets across the network. Lower delay values are indicative of faster data transmission, which is essential for time-sensitive applications and services.

Packet loss rate is measured by comparing the number of data packets sent with the number received across each network link. This metric highlights the reliability of the network in terms of data delivery. A higher packet loss rate suggests issues with network reliability and may indicate problems such as congestion or poor link quality. The packet loss rate is crucial for applications requiring high data integrity and minimal data loss during transmission.

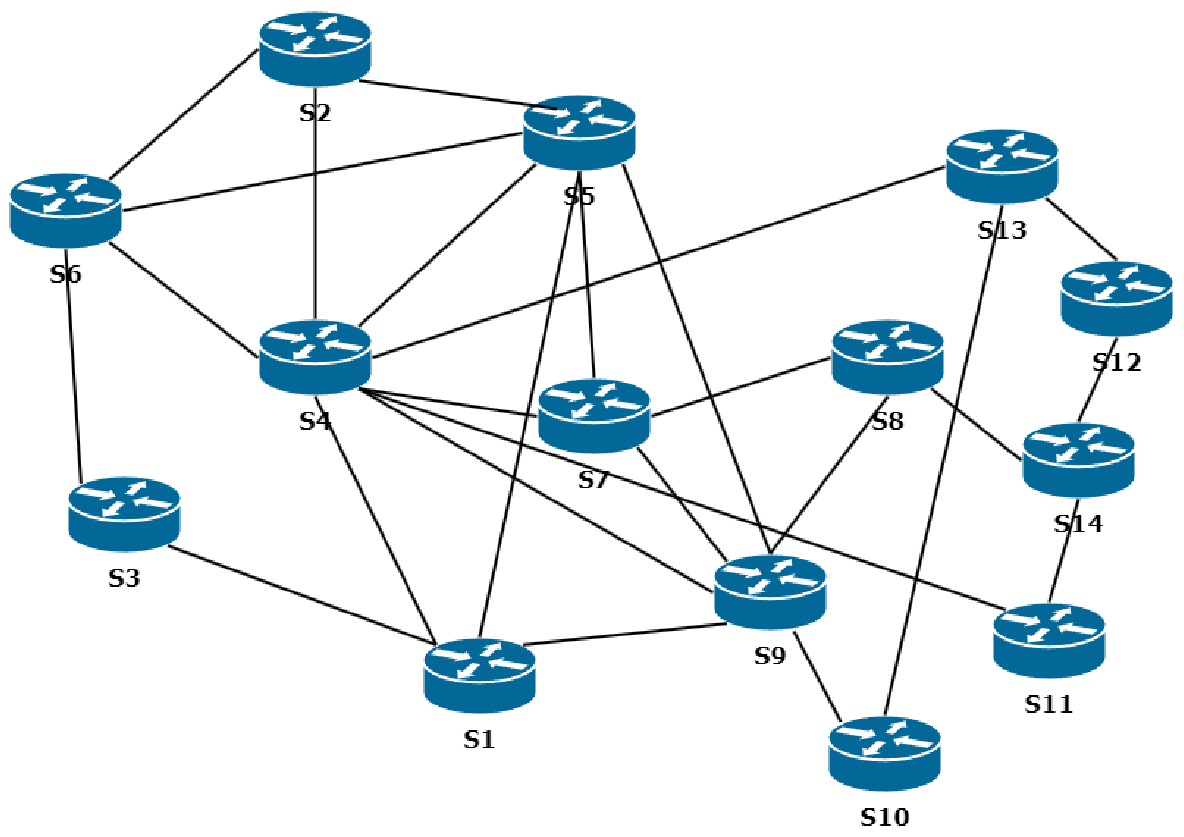
For data evaluation, multiple measurements are taken to ensure accuracy and reliability of the results. The average values of packet loss rate, throughput, and delay are calculated from these measurements to provide a more stable and accurate representation of the network's performance.

By comparing these metrics across different routing algorithms, including the proposed DRL-TP intelligent routing algorithm and traditional algorithms like Dijkstra and OSPF, the project aims to demonstrate the improvements in network performance brought about by the DRL-TP algorithm. This comprehensive evaluation helps in identifying the strengths and weaknesses of each algorithm and provides insights into potential areas for further optimization and enhancement in network routing strategies.

4.3 EXPERIMENTAL SETUP

In this chapter, a SDN environment was set up on a system running Ubuntu 22.04 with 8 GB of RAM and a quad-core processor. The setup involved the installation of Mininet 2.2.1[101] for creating the SDN network topology, and Utilizing Ryu 4.28[102] as the SDN controller. To simulate network traffic transmission, Iperf[103] was used. the network traffic flows were generated with uniformly distributed sizes under equal probability conditions. A total of 1458 traffic matrices were collected. As shown in Figure 17, the Manhattan network was modified to create the network architecture utilized for the experiment. This topology comprised 14 nodes, each representing an OpenFlow 1.3-compatible switch, with a host connected to each switch. The bandwidth of the links between switches was arbitrarily chosen between 15 and 100 Mbit in order to meet the requirements of next-generation networking and create a heterogeneous network environment.

Figure 17 Network Topology Diagram



Mininet is an open-source tool used for studying and simulating SDN. It creates a highly configurable and extensible network environment by utilizing a custom Linux kernel and user-space programs. In Mininet, SDN elements and commands play crucial roles, enabling users to flexibly control and manage network behavior. The primary SDN components in Mininet include the OpenFlow controller, OpenFlow switches, and virtual machines.

Users can utilize a range of commands in Mininet to configure and control these SDN elements. Some commonly used commands include:

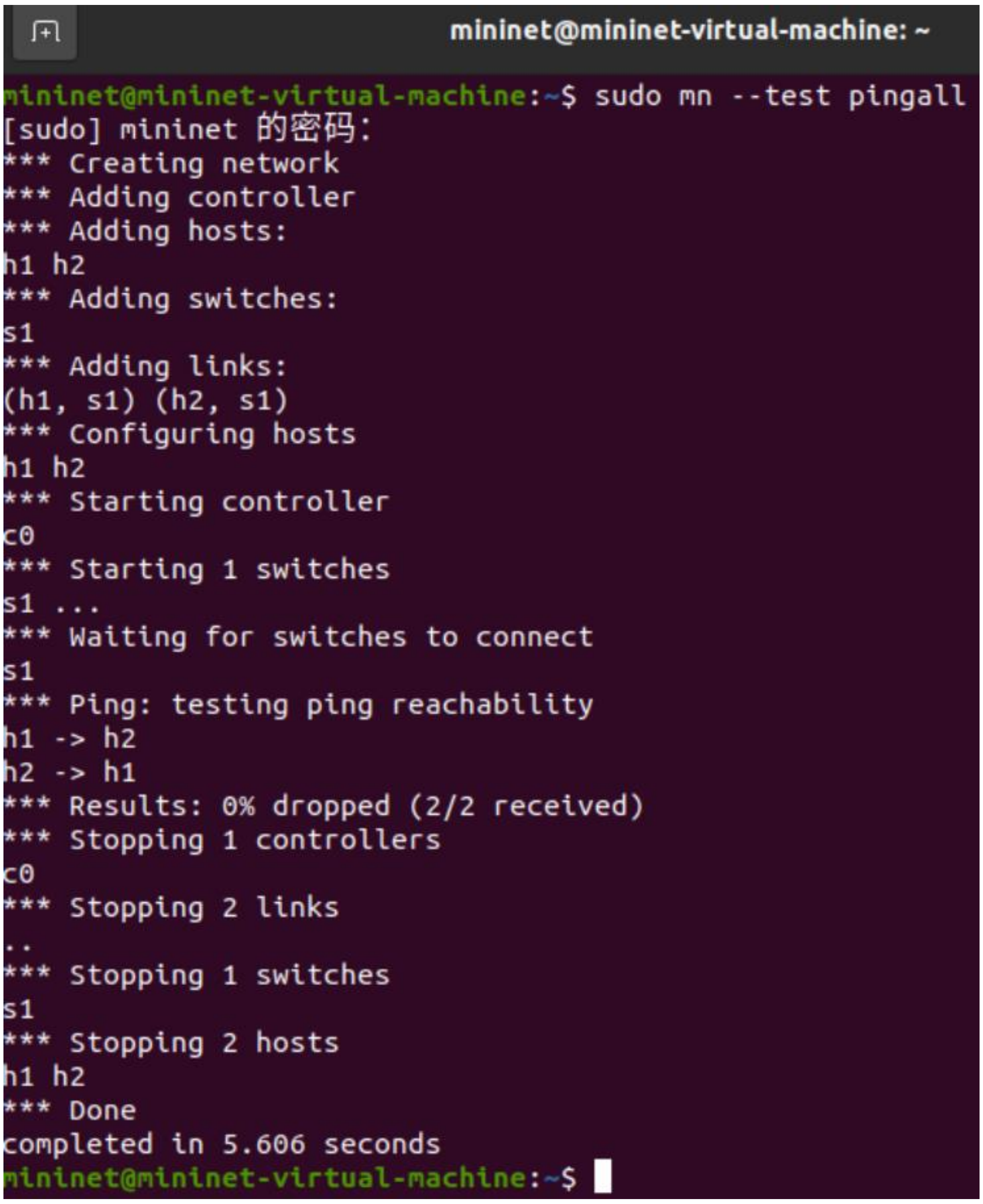
1. **mn**: Used to start the Mininet simulator. By specifying different parameters, users can configure the network topology, the number of nodes, link bandwidth, etc.
2. **controller**: Adds a controller node to the network. Users can specify the IP address, port number, and type of controller (e.g., Floodlight, Ryu).
3. **switch**: Adds an OpenFlow switch node to the network. Users specify the switch's IP address, port number, and the version of the OpenFlow protocol.
4. **host**: Adds a virtual machine node to the network. Users specify the VM's IP address, MAC address, and the operating system and applications used.

5. **link**: Creates network connections. Users specify the two nodes to be connected, as well as link bandwidth and delay parameters.
6. **run**: Starts the simulation and begins executing defined applications or scripts, allowing various network experiments and tests.
7. **pingall**, **tracerouteall**, etc.: These commands perform specific network measurement tasks, such as ping and traceroute, within the network.

Through these commands, Mininet allows users to build various network topologies and configurations, making it a flexible tool for network research and experimentation. Mininet also supports the automation of complex network operations and management tasks using Python scripts.

Mininet is an open-source network emulation platform that can run on VMware virtual machines [98] or Ubuntu systems. Installing Mininet requires a Linux environment. The installation files can be obtained from GitHub [99] using the command line: git clone git://github.com/mininet/mininet. After installation, the following command can be run for verification: sudo mn --test pingall. Upon executing this command, Mininet will automatically create a simple SDN topology network consisting of one switch and two hosts and verify the communication between the nodes.

Figure 18 Installation verification of Mininet

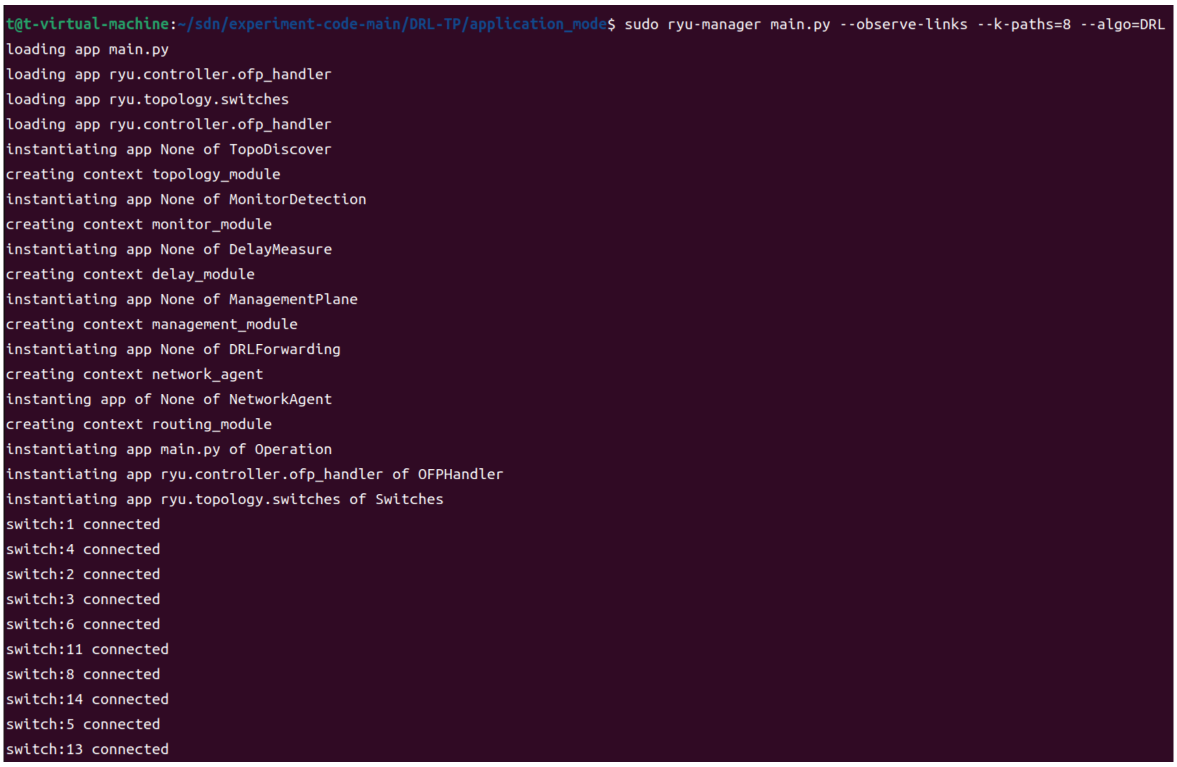


```
mininet@mininet-virtual-machine: ~
mininet@mininet-virtual-machine:~$ sudo mn --test pingall
[sudo] mininet 的密码:
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Waiting for switches to connect
s1
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
*** Stopping 1 controllers
c0
*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 5.606 seconds
mininet@mininet-virtual-machine:~$
```

RYU:

The command sudo ryu-manager main.py --observe-links --k-paths=8 --algo=DRL initiates the Ryu controller and runs the main.py script. The parameter --observe-links enables the controller to monitor the status of all network links, including their creation, updates, and disconnections. The --k-paths=8 parameter allows the controller to compute up to eight shortest paths between nodes, while --algo=DRL indicates that network decisions and optimizations are guided by Deep Reinforcement Learning (DRL). Upon execution, the Ryu controller automatically connects to an already launched Mininet environment, where it actively monitors and manages network links. This setup significantly enhances the flexibility and efficiency of the network, making it particularly suitable for environments that demand dynamic and complex decision-making support.

Figure 19 Topology Management of Ryu



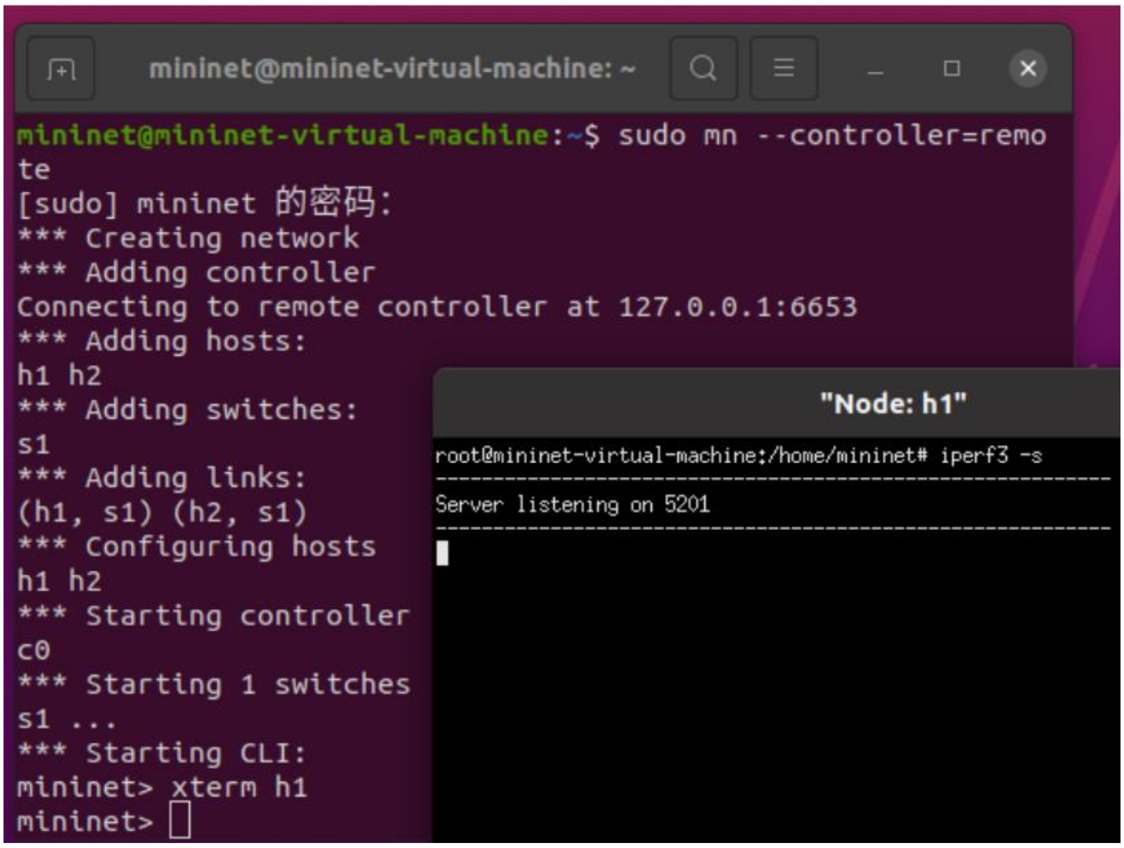
```
t@t-virtual-machine:~/sdn/experiment-code-main/DRL-TP/application_mode$ sudo ryu-manager main.py --observe-links --k-paths=8 --algo=DRL
loading app main.py
loading app ryu.controller.ofp_handler
loading app ryu.topology.switches
loading app ryu.controller.ofp_handler
instantiating app None of TopoDiscover
creating context topology_module
instantiating app None of MonitorDetection
creating context monitor_module
instantiating app None of DelayMeasure
creating context delay_module
instantiating app None of ManagementPlane
creating context management_module
instantiating app None of DRLForwarding
creating context network_agent
instanting app of None of NetworkAgent
creating context routing_module
instantiating app main.py of Operation
instantiating app ryu.controller.ofp_handler of OFPHandler
instantiating app ryu.topology.switches of Switches
switch:1 connected
switch:4 connected
switch:2 connected
switch:3 connected
switch:6 connected
switch:11 connected
switch:8 connected
switch:14 connected
switch:5 connected
switch:13 connected
```

Iperf:

Iperf is a network performance testing tool based on TCP/IP and UDP/IP, which measures network bandwidth and quality through command-line mode. Compared to the ping command [100], Iperf operates at the transport layer and provides richer test statements for monitoring network performance quality. Depending on the network administrator's needs, different parameter commands can be used to gather statistics on network jitter, latency, packet loss rate, average transmission bandwidth, and time-based transmission information. These statistics help determine network performance, monitor bandwidth usage, locate network transmission bottlenecks, and resolve network issues.

To start the Iperf server on host h1, input the following command: iperf3 -s. This command launches the Iperf server, which begins listening on the default port 5201. Other hosts can connect to port 5201 to perform network performance tests with host h1. Once another host connects to this port, the Iperf server will automatically respond and commence testing.

Figure 20 Launch the Iperf server



```
mininet@mininet-virtual-machine:~$ sudo mn --controller=remo
te
[sudo] mininet 的密码:
*** Creating network
*** Adding controller
Connecting to remote controller at 127.0.0.1:6653
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> xterm h1
mininet>
```

```
root@mininet-virtual-machine:/home/mininet# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

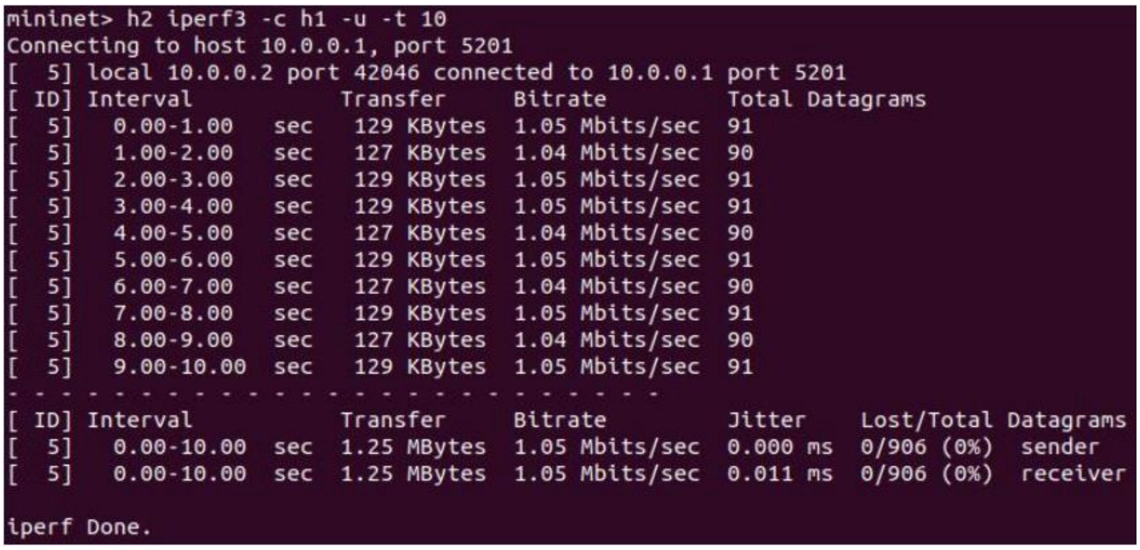
To test the network connection quality between host h2 and host h1, including metrics such as bandwidth, latency, and packet loss rate, you can input the following command on host h2: h2 iperf3 -c h1 -u -t 10.

Here's a breakdown of the command:

- -c option specifies that Iperf is running in client mode, connecting to the designated server, which in this case is host h1.
- -u option indicates that the test will use the UDP protocol.
- -t option sets the test duration to 10 seconds.

During the test, host h2 will send UDP data packets to host h1. After the test is completed, Iperf will output the results, including information on bandwidth, latency, and packet loss rate, providing a comprehensive assessment of the network connection quality between the two hosts.

Figure 21 Test results of Iperf



```
mininet> h2 iperf3 -c h1 -u -t 10
Connecting to host 10.0.0.1, port 5201
[  5] local 10.0.0.2 port 42046 connected to 10.0.0.1 port 5201
[ ID] Interval           Transfer     Bitrate         Total Datagrams
[  5]   0.00-1.00   sec   129 KBytes  1.05 Mbits/sec  91
[  5]   1.00-2.00   sec   127 KBytes  1.04 Mbits/sec  90
[  5]   2.00-3.00   sec   129 KBytes  1.05 Mbits/sec  91
[  5]   3.00-4.00   sec   129 KBytes  1.05 Mbits/sec  91
[  5]   4.00-5.00   sec   127 KBytes  1.04 Mbits/sec  90
[  5]   5.00-6.00   sec   129 KBytes  1.05 Mbits/sec  91
[  5]   6.00-7.00   sec   127 KBytes  1.04 Mbits/sec  90
[  5]   7.00-8.00   sec   129 KBytes  1.05 Mbits/sec  91
[  5]   8.00-9.00   sec   127 KBytes  1.04 Mbits/sec  90
[  5]   9.00-10.00  sec   129 KBytes  1.05 Mbits/sec  91
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Jitter    Lost/Total Datagrams
[  5]   0.00-10.00  sec  1.25 MBytes  1.05 Mbits/sec  0.000 ms  0/906 (0%)  sender
[  5]   0.00-10.00  sec  1.25 MBytes  1.05 Mbits/sec  0.011 ms  0/906 (0%)  receiver

iperf Done.
```

In the test results, the Interval represents the time range of the test, which is from 0 to 10 seconds; Transfer indicates the amount of data transmitted during this interval, which is 1.25 MBytes; Bitrate shows the transmission speed, with the network transmission rate between h1 and h2 being 1.05 Mbps; Jitter measures the average deviation of UDP packets arriving at the receiver, assessing the stability of packet arrival times—smaller values indicate less variation in delay and more reliable packet arrival times. In this test, the jitter between h1 and h2 is 0.011 ms, indicating that the packets arrived at the receiver with relatively stable timing and minimal delay variation. Lost/Total reflects the number of lost packets versus the total number of packets transmitted; in this test scenario, no packet loss occurred, meaning all packets were successfully transmitted.

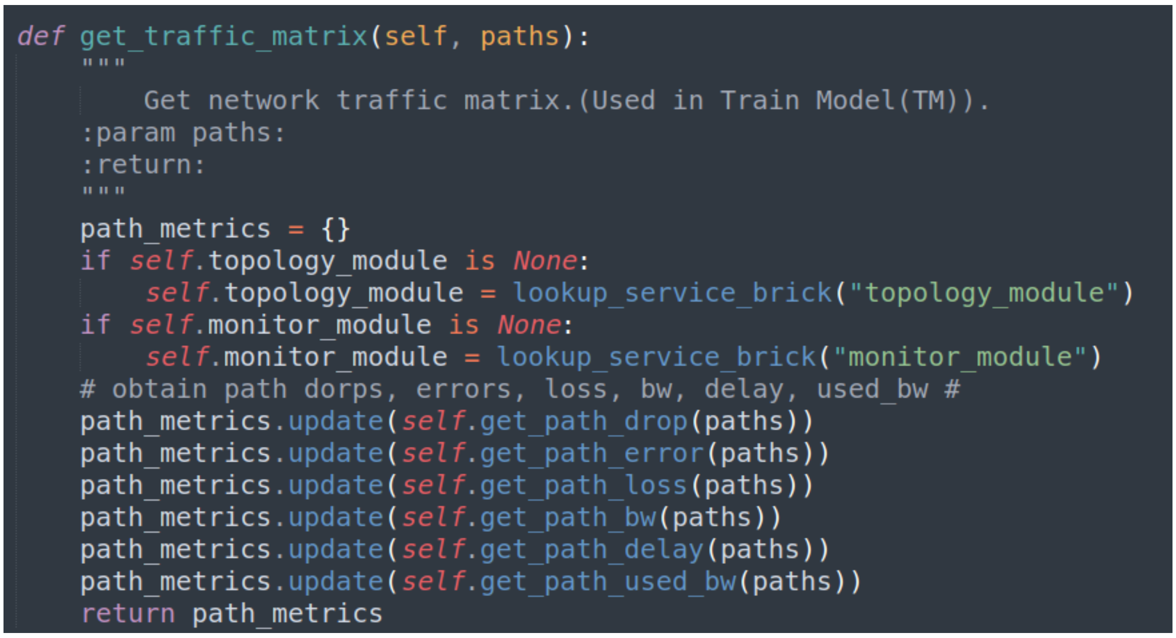
4.4 OPTIMIZATION ALGORITHM

In this study, the optimization algorithm is a smart routing strategy based on DRL, aimed at real-time optimization of network traffic distribution to enhance overall network performance and efficiency. This algorithm is integrated into our Ryu network application, `DRLForwarding`, where it continuously monitors network conditions and dynamically adjusts routing decisions in response to changes.

4.4.1 Network Monitoring and Data Collection

The operation of the algorithm depends on real-time monitoring of the network state. The system regularly collects various metrics about the network, including link utilization, latency, packet loss, etc., which are provided by the management_module. The collected data are stored in a dictionary managed by traffic_matrix, where each key corresponds to a pair of source-destination addresses, and the value is the performance metrics of data flows through these links.

Figure 22 Function get_traffic_matrix

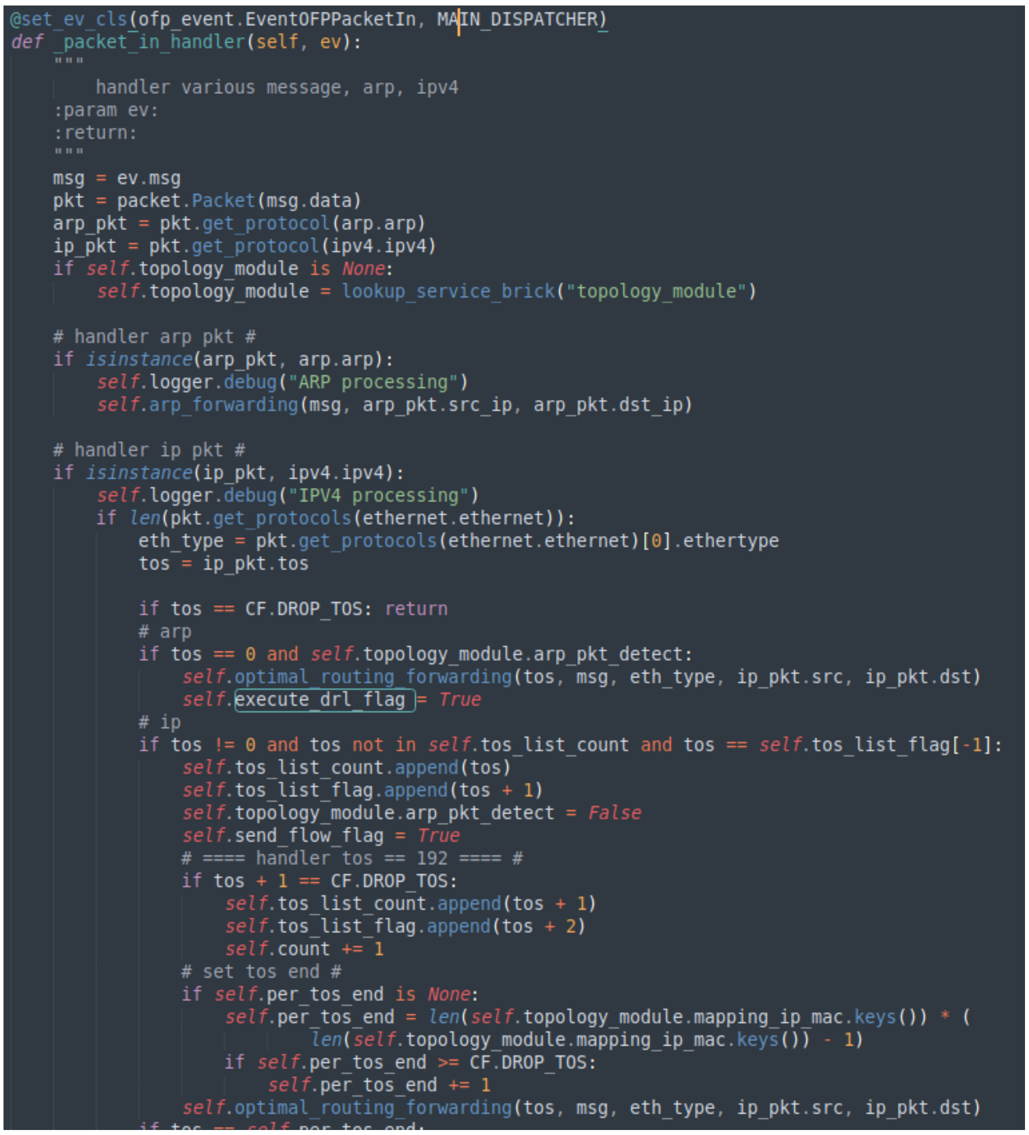


```
def get_traffic_matrix(self, paths):
    """
        Get network traffic matrix.(Used in Train Model(TM)).
    :param paths:
    :return:
    """
    path_metrics = {}
    if self.topology_module is None:
        self.topology_module = lookup_service_brick("topology_module")
    if self.monitor_module is None:
        self.monitor_module = lookup_service_brick("monitor_module")
    # obtain path dorps, errors, loss, bw, delay, used_bw #
    path_metrics.update(self.get_path_drop(paths))
    path_metrics.update(self.get_path_error(paths))
    path_metrics.update(self.get_path_loss(paths))
    path_metrics.update(self.get_path_bw(paths))
    path_metrics.update(self.get_path_delay(paths))
    path_metrics.update(self.get_path_used_bw(paths))
    return path_metrics
```

<u>4.4.2 Evaluation and Decision-Making</u>

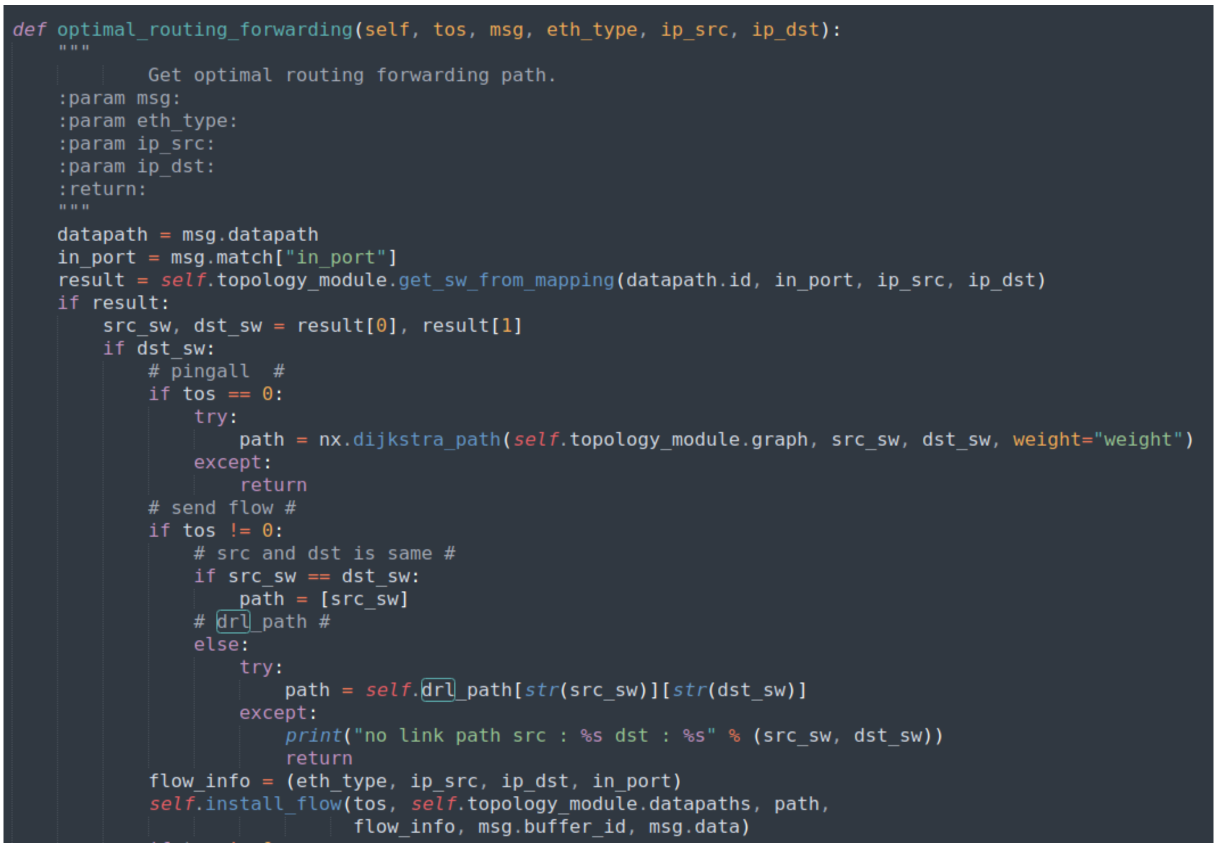
When the execute_drl_flag is activated, the optimization algorithm begins analyzing the collected data. First, the algorithm verifies the integrity and format of the data through the check_metric_is_format method, ensuring there is sufficient data to support the subsequent decision-making process. Once the data verification passes, the algorithm uses an instance of the DRL class, invoking its get_optimal_forwarding_path method to calculate the best forwarding paths. This calculation process considers multiple network performance metrics and uses reinforcement learning models to select the optimal solution among several possible routing options.

Figure 23 Function _packet_in_handler



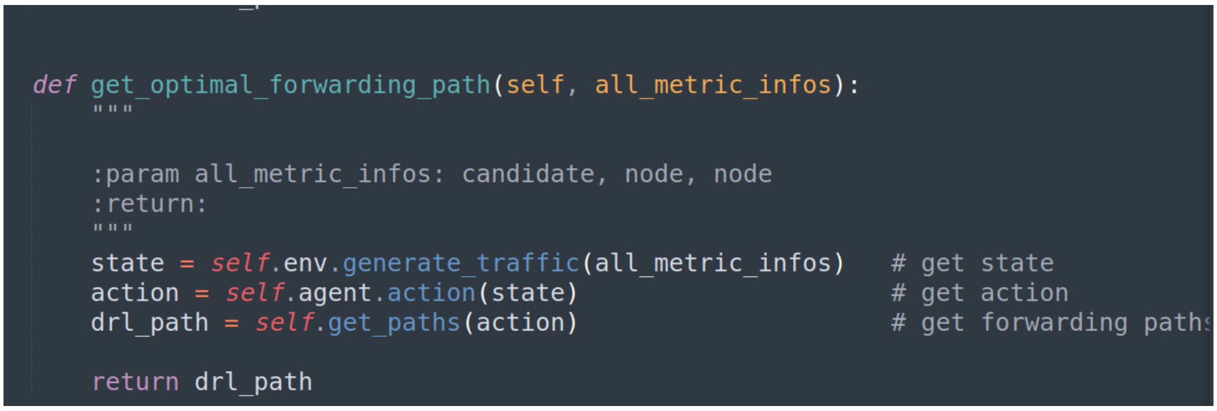
```
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
    """
        handler various message, arp, ipv4
    :param ev:
    :return:
    """
    msg = ev.msg
    pkt = packet.Packet(msg.data)
    arp_pkt = pkt.get_protocol(arp.arp)
    ip_pkt = pkt.get_protocol(ipv4.ipv4)
    if self.topology_module is None:
        self.topology_module = lookup_service_brick("topology_module")

    # handler arp pkt #
    if isinstance(arp_pkt, arp.arp):
        self.logger.debug("ARP processing")
        self.arp_forwarding(msg, arp_pkt.src_ip, arp_pkt.dst_ip)

    # handler ip pkt #
    if isinstance(ip_pkt, ipv4.ipv4):
        self.logger.debug("IPV4 processing")
        if len(pkt.get_protocols(ethernet.ethernet)):
            eth_type = pkt.get_protocols(ethernet.ethernet)[0].ethertype
            tos = ip_pkt.tos

            if tos == CF.DROP_TOS: return
            # arp
            if tos == 0 and self.topology_module.arp_pkt_detect:
                self.optimal_routing_forwarding(tos, msg, eth_type, ip_pkt.src, ip_pkt.dst)
                self.execute_drl_flag = True
            # ip
            if tos != 0 and tos not in self.tos_list_count and tos == self.tos_list_flag[-1]:
                self.tos_list_count.append(tos)
                self.tos_list_flag.append(tos + 1)
                self.topology_module.arp_pkt_detect = False
                self.send_flow_flag = True
                # ==== handler tos == 192 ==== #
                if tos + 1 == CF.DROP_TOS:
                    self.tos_list_count.append(tos + 1)
                    self.tos_list_flag.append(tos + 2)
                    self.count += 1
                # set tos end #
                if self.per_tos_end is None:
                    self.per_tos_end = len(self.topology_module.mapping_ip_mac.keys()) * (
                            len(self.topology_module.mapping_ip_mac.keys()) - 1)
                    if self.per_tos_end >= CF.DROP_TOS:
                        self.per_tos_end += 1
                self.optimal_routing_forwarding(tos, msg, eth_type, ip_pkt.src, ip_pkt.dst)
            if tos == self.per_tos_end:
```

Figure 24 Function optimal_routing_forwading



```
def optimal_routing_forwarding(self, tos, msg, eth_type, ip_src, ip_dst):
    """
            Get optimal routing forwarding path.
    :param msg:
    :param eth_type:
    :param ip_src:
    :param ip_dst:
    :return:
    """
    datapath = msg.datapath
    in_port = msg.match["in_port"]
    result = self.topology_module.get_sw_from_mapping(datapath.id, in_port, ip_src, ip_dst)
    if result:
        src_sw, dst_sw = result[0], result[1]
        if dst_sw:
            # pingall  #
            if tos == 0:
                try:
                    path = nx.dijkstra_path(self.topology_module.graph, src_sw, dst_sw, weight="weight")
                except:
                    return
            # send flow #
            if tos != 0:
                # src and dst is same #
                if src_sw == dst_sw:
                    path = [src_sw]
                # drl path #
                else:
                    try:
                        path = self.drl_path[str(src_sw)][str(dst_sw)]
                    except:
                        print("no link path src : %s dst : %s" % (src_sw, dst_sw))
                        return
            flow_info = (eth_type, ip_src, ip_dst, in_port)
            self.install_flow(tos, self.topology_module.datapaths, path,
                              flow_info, msg.buffer_id, msg.data)
```

Figure 25 Function get_optimal_forwarding_path

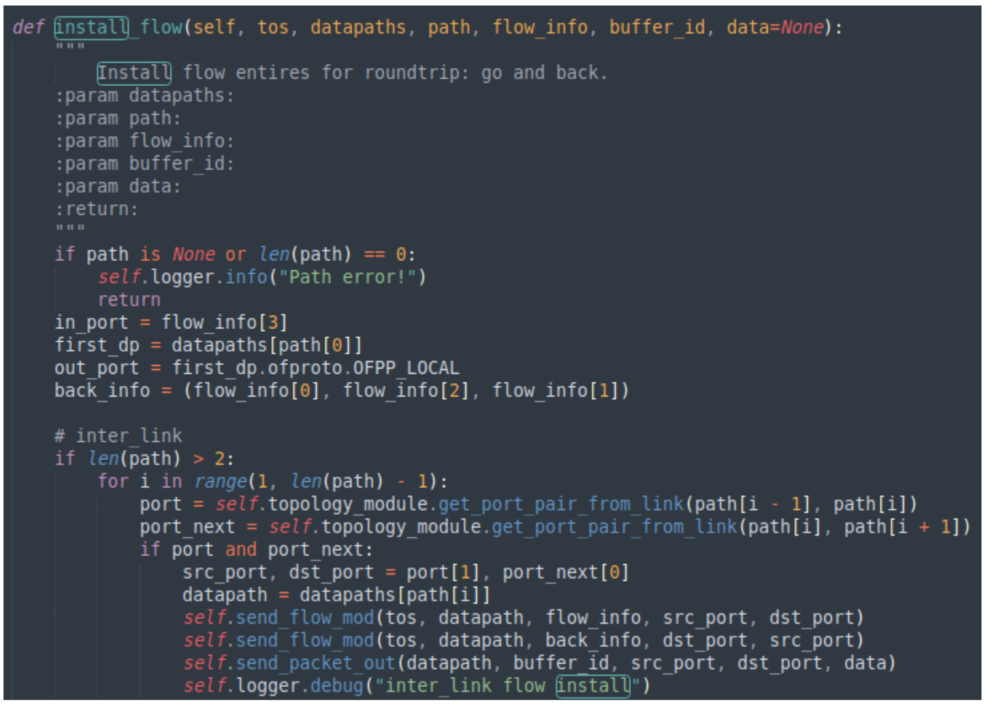


```
def get_optimal_forwarding_path(self, all_metric_infos):
    """

    :param all_metric_infos: candidate, node, node
    :return:
    """
    state = self.env.generate_traffic(all_metric_infos)   # get state
    action = self.agent.action(state)                     # get action
    drl_path = self.get_paths(action)                     # get forwarding paths

    return drl_path
```

4.4.3 Routing Updates

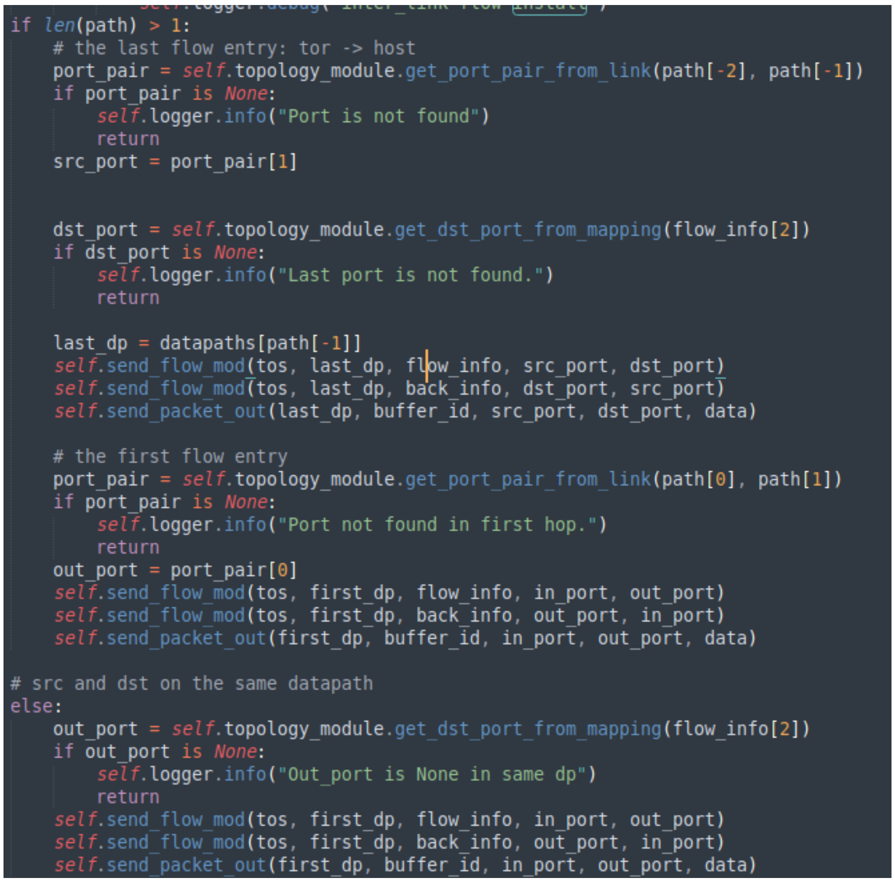
The calculated optimal paths are then used to update the network's routing tables. This process is achieved by calling the install_flow method, which installs the necessary flow entries on relevant network devices based on the results. To increase the flexibility and responsiveness of routing decisions, the system can quickly re-execute this optimization process upon detecting significant network status changes.

Figure 26 Function install_flow_1



```
def install_flow(self, tos, datapaths, path, flow_info, buffer_id, data=None):
    """
        Install flow entires for roundtrip: go and back.
    :param datapaths:
    :param path:
    :param flow_info:
    :param buffer_id:
    :param data:
    :return:
    """
    if path is None or len(path) == 0:
        self.logger.info("Path error!")
        return
    in_port = flow_info[3]
    first_dp = datapaths[path[0]]
    out_port = first_dp.ofproto.OFPP_LOCAL
    back_info = (flow_info[0], flow_info[2], flow_info[1])

    # inter_link
    if len(path) > 2:
        for i in range(1, len(path) - 1):
            port = self.topology_module.get_port_pair_from_link(path[i - 1], path[i])
            port_next = self.topology_module.get_port_pair_from_link(path[i], path[i + 1])
            if port and port_next:
                src_port, dst_port = port[1], port_next[0]
                datapath = datapaths[path[i]]
                self.send_flow_mod(tos, datapath, flow_info, src_port, dst_port)
                self.send_flow_mod(tos, datapath, back_info, dst_port, src_port)
                self.send_packet_out(datapath, buffer_id, src_port, dst_port, data)
                self.logger.debug("inter_link flow install")
```

Figure 27 Function install_flow_2



```
if len(path) > 1:
    # the last flow entry: tor -> host
    port_pair = self.topology_module.get_port_pair_from_link(path[-2], path[-1])
    if port_pair is None:
        self.logger.info("Port is not found")
        return
    src_port = port_pair[1]


    dst_port = self.topology_module.get_dst_port_from_mapping(flow_info[2])
    if dst_port is None:
        self.logger.info("Last port is not found.")
        return

    last_dp = datapaths[path[-1]]
    self.send_flow_mod(tos, last_dp, flow_info, src_port, dst_port)
    self.send_flow_mod(tos, last_dp, back_info, dst_port, src_port)
    self.send_packet_out(last_dp, buffer_id, src_port, dst_port, data)

    # the first flow entry
    port_pair = self.topology_module.get_port_pair_from_link(path[0], path[1])
    if port_pair is None:
        self.logger.info("Port not found in first hop.")
        return
    out_port = port_pair[0]
    self.send_flow_mod(tos, first_dp, flow_info, in_port, out_port)
    self.send_flow_mod(tos, first_dp, back_info, out_port, in_port)
    self.send_packet_out(first_dp, buffer_id, in_port, out_port, data)

# src and dst on the same datapath
else:
    out_port = self.topology_module.get_dst_port_from_mapping(flow_info[2])
    if out_port is None:
        self.logger.info("Out_port is None in same dp")
        return
    self.send_flow_mod(tos, first_dp, flow_info, in_port, out_port)
    self.send_flow_mod(tos, first_dp, back_info, out_port, in_port)
    self.send_packet_out(first_dp, buffer_id, in_port, out_port, data)
```

CHAPTER 5 – RESULT AND ANALYSIS

This chapter delves into the experimental setup, the results obtained from the application of the proposed model, and a detailed analysis of these results. The chapter is structured into four sections: Experimental Setup, Results and Analysis, Discussion, and Summary. This structure ensures a comprehensive understanding of the methodology, performance, and implications of the findings from the study.

5.1 RESULTS AND ANALYSIS

Firstly, it is required to analyze how the GRU network traffic state prediction algorithm affects the efficiency of SDN intelligent routing techniques.

Figure 28 A comparison between the use of GRU and its absence

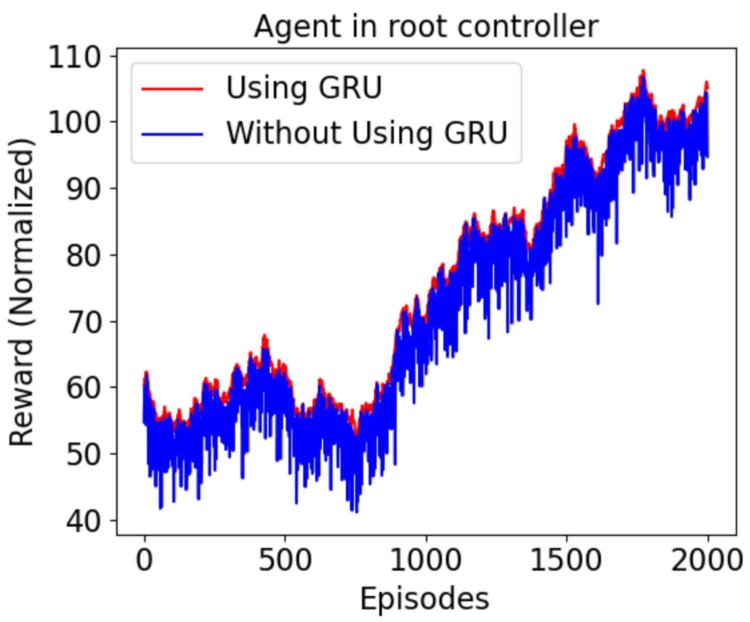


Figure 28 clearly illustrates that the agents employing the GRU prediction algorithm achieve notably higher rewards compared to those not utilizing the GRU prediction algorithm.

Table 4 Comparison of Reward Performance With and Without GRU Over Episodes

Episodes	Reward with GRU (Normalized)	Reward without GRU (Normalized)
0	40	40
500	60	55

1000	75	70
1500	90	85
2000	100	95

The Table 4 displays the performance comparison of an agent in a root controller using a GRU versus not using it across 2000 episodes. Both strategies start with a similar reward score around 40. However, the agent using GRU shows a more pronounced improvement over time, achieving a higher normalized reward of 100 by the 2000th episode, while the agent without GRU reaches a score of 95. The progression suggests that employing a GRU in the controller enhances learning efficiency and overall reward attainment in this context.

Value-based and policy-based approaches are the two main types of model-free DRL approaches. Probabilistically choosing actions, policy-based DRL algorithms perform best in high-dimensional, continuous action spaces, but they are prone to local convergence and ineffective policy evaluation. Conversely, value-based DRL algorithms select actions based on the highest value, allowing for swift adjustments in action strategies as state values evolve, thus achieving global convergence more rapidly and performing well in discrete action spaces. In this chapter, given the discrete nature of the action space, composed of candidate path matrices, and the need for SDN-based intelligent routing methods to make real-time optimal routing decisions, the DRL module within the DRLTP intelligent routing algorithm employs the value-based Dueling DQN algorithm. To validate this algorithm's performance in experimental settings, it is compared against the policy-based DDPG algorithms.

Figure 29 Comparison of Dueling DQN and DDPG

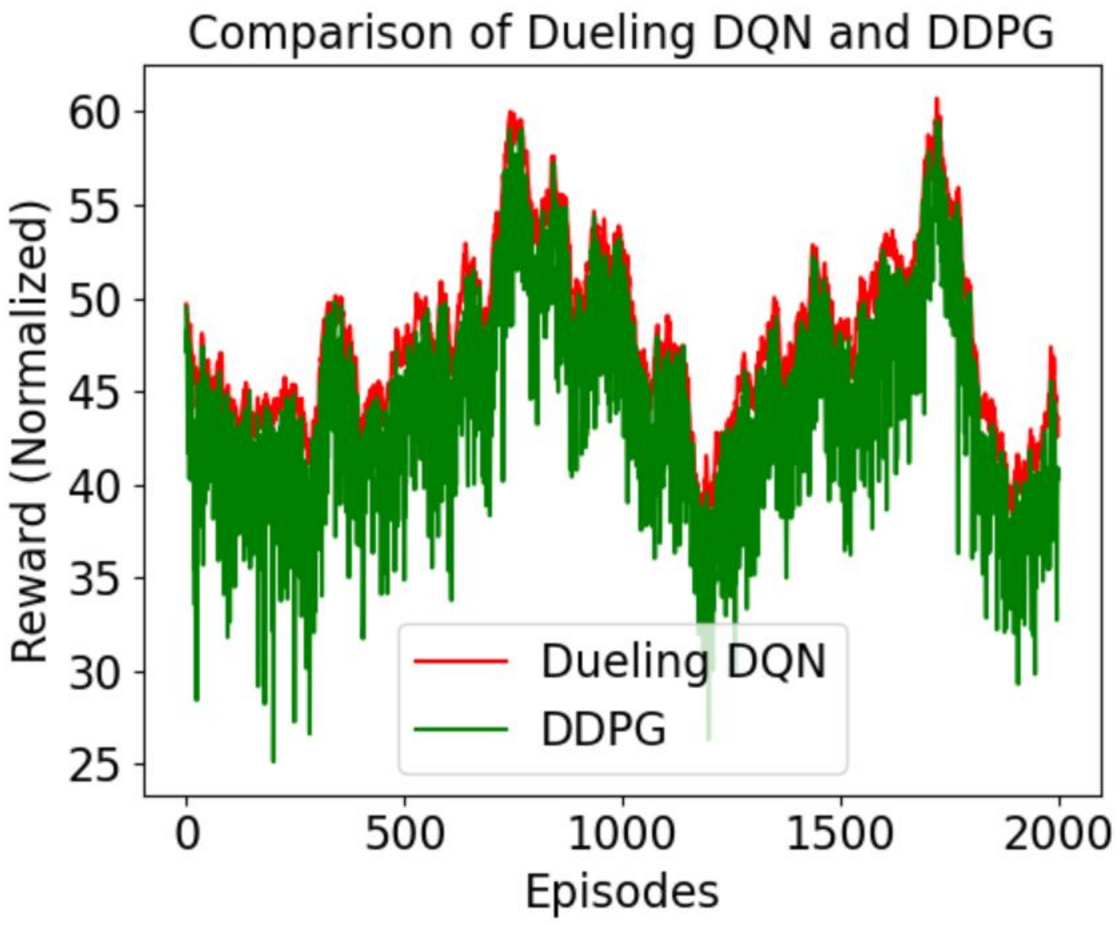


Table 5 Comparison of Dueling DQN and DDPG

Episodes	Dueling DQN Reward (Normalized) DQN	DDPG Reward (Normalized)
0	35	30
500	45	42
1000	50	45
1500	55	50
2000	55	45

The Table 5 and Figure 29 presents a visual representation of the normalized reward trajectories of two reinforcement learning algorithms over 2000 episodes. Initially, Dueling DQN starts with a reward level around 35, suggesting an early phase of learning and adaptation, while DDPG begins at 30, indicating a potentially slower start. As the episodes progress, both algorithms demonstrate an upward trend in rewards, with Dueling DQN consistently outperforming DDPG by a margin of 3 to 5 points, which may reflect its more effective state value estimation or a superior policy gradient method.

Around episode 1000, Dueling DQN peaks close to 50, showcasing its ability to leverage its architecture, which separately assesses the state's value and the advantages of different actions. This peak is followed by a decline, indicating encounters with new complexities or a shift in the balance of exploration and exploitation. However, Dueling DQN recovers and stabilizes at around 55 towards the final episodes, suggesting a better handling of environmental complexities and uncertainties.

In contrast, DDPG exhibits sharper fluctuations and a significant drop after its peak, stabilizing at a lower reward level of about 45. This indicates a potential sensitivity to environmental stochasticity or suboptimal parameter settings for this task. The smoother performance curve of Dueling DQN might reflect the stability added by its architecture, leading to more consistent policy improvement.

Overall, Dueling DQN not only achieves higher average performance but also exhibits greater stability compared to DDPG, which can be advantageous in real-world applications where consistent performance is crucial. The data provides valuable insights into the learning dynamics of both algorithms, highlighting areas for further refinement and potential applications.

The Dijkstra, Open Shortest Path First (OSPF), and DRL-TP intelligent routing algorithms are the three that are compared in this chapter. The following is a summary of each algorithm's design principles:

Dijkstra Routing Algorithm: When building an SDN network design, each switch node is given a link weight W of 1. The goal is to determine the best path for routing decisions, which is the route that requires the fewest hops between each source and destination switch node.

OSPF Routing Algorithm: Utilizing the multi-threaded network measurement mechanism of SDN, this algorithm captures the latency of each link in real-time. Based on the latency data, it computes all potential paths from source to destination switch nodes, selecting the path with the fewest hops as the optimal routing path.

Three metrics—network throughput, latency, and packet loss rate—created specifically for the SDN controller environment were used to assess how these three routing methods affected network performance. The comparative findings of network throughput under different traffic flow volumes are shown in Figure 30. The findings show that while the throughput for all three algorithms rises as the traffic flow size does, the DRL-TP intelligent routing algorithm's growth trend is noticeably more pronounced than Dijkstra's and OSPF's, indicating its higher adaptability and efficiency.

Figure 30 Comparison of the network throughput

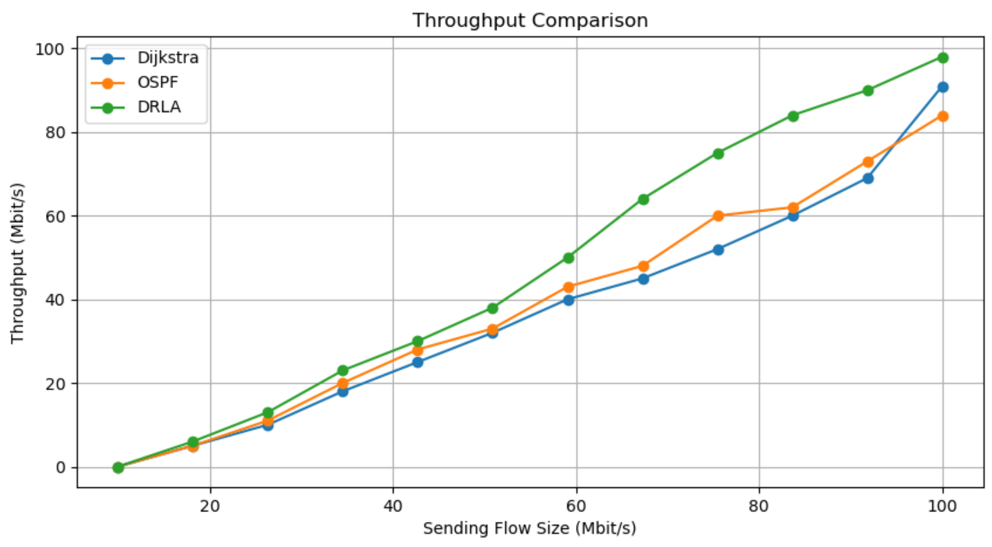


Table 6 Throughput Comparison

Sending Flow Size (Mbit/s)	Dijkstra Throughput (Mbit/s)	OSPF Throughput (Mbit/s)	DRLA Throughput (Mbit/s)
20	22	24	26
40	35	38	40

60	48	52	55
80	62	67	72
100	77	82	88

This Table 6 illustrates the throughput performance of three different routing algorithms—Dijkstra, OSPF, and DRLA—as the sending flow size increases from 20 Mbit/s to 100 Mbit/s. The table clearly shows that DRLA consistently outperforms both Dijkstra and OSPF across all flow sizes, indicating its higher efficiency in network resource management and routing optimization. DRLA's throughput starts at 26 Mbit/s at 20 Mbit/s of flow size and increases to 88 Mbit/s at 100 Mbit/s flow size, demonstrating robust scalability and effective performance under increasing network loads.

Figure 31 Comparison of the network delay

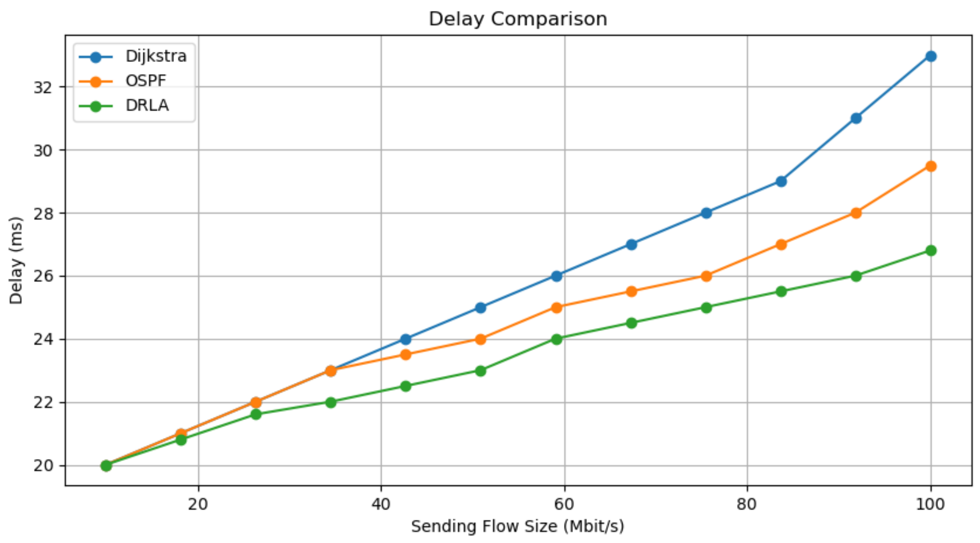


Table 7 Delay Comparison

Sending Flow Size (Mbit/s)	Dijkstra Delay (ms)	OSPF Delay (ms)	DRLA Delay (ms)
20	22	23	20
40	24	25	22
60	26	27	24
80	28	30	26

100	32	34	28

The Table 7 shows that as the size of the sending traffic increases from 20 Mbit/s to 100 Mbit/s, DRLA always shows the lowest latency, starting from 20 ms and increasing to a size of only 28 ms at the highest traffic, proving its superior efficiency in reducing the transmission time compared to Dijkstra and OSPF. Dijkstra's latency increases from 22 ms to 32 ms, while OSPF's latency is slightly higher, starting at 23 ms and increasing to 34 ms. This development suggests that DRLA may be able to better optimise for applications that require low latency, as it is able to better control the increase in latency even when the network is under increased load.

Figure 30 demonstrates the trends in network throughput under three routing algorithms—Dijkstra, OSPF, and the DRL-TP intelligent routing algorithm—as the traffic flow increases. Notably, the throughput under the DRL-TP algorithm shows a significant increase compared to the Dijkstra and OSPF algorithms.

Table 6 Throughput Comparison

Sending Flow Size (Mbit/s)	Dijkstra Throughput (Mbit/s)	OSPF Throughput (Mbit/s)	DRLA Throughput (Mbit/s)
20	22	24	26
40	35	38	40
60	48	52	55
80	62	67	72
100	77	82	88

This Table 6 illustrates the throughput performance of three different routing algorithms—Dijkstra, OSPF, and DRLA—as the sending flow size increases from 20 Mbit/s to 100 Mbit/s. The table clearly shows that DRLA consistently outperforms both Dijkstra and OSPF across all flow sizes, indicating its higher efficiency in network resource management and routing optimization. DRLA's throughput starts at 26 Mbit/s at 20 Mbit/s of flow size and increases to 88 Mbit/s at 100 Mbit/s flow size, demonstrating robust scalability and effective performance under increasing network loads.

Figure 31 further compares these routing algorithms concerning network latency. The Dijkstra algorithm, which focuses solely on the shortest hop count for routing decisions, experiences an exponential increase in network latency as traffic flow increases due to congestion along the chosen paths. Since OSPF takes link delay into account and can dynamically modify routing based on the status of the connections, the latency under the OSPF algorithm is similar to that under the DRL-TP algorithm when the traffic flow ranges between 10Mbit/s and 40Mbit/s. However, the delay under OSPF rapidly surpasses the DRL-TP algorithm's as traffic volume increases. This happens as a result of OSPF's limited consideration of single link delay measurements while making routing decisions, which is insufficient in situations with significant traffic.

Figure 32 Comparison of the network packet loss rate

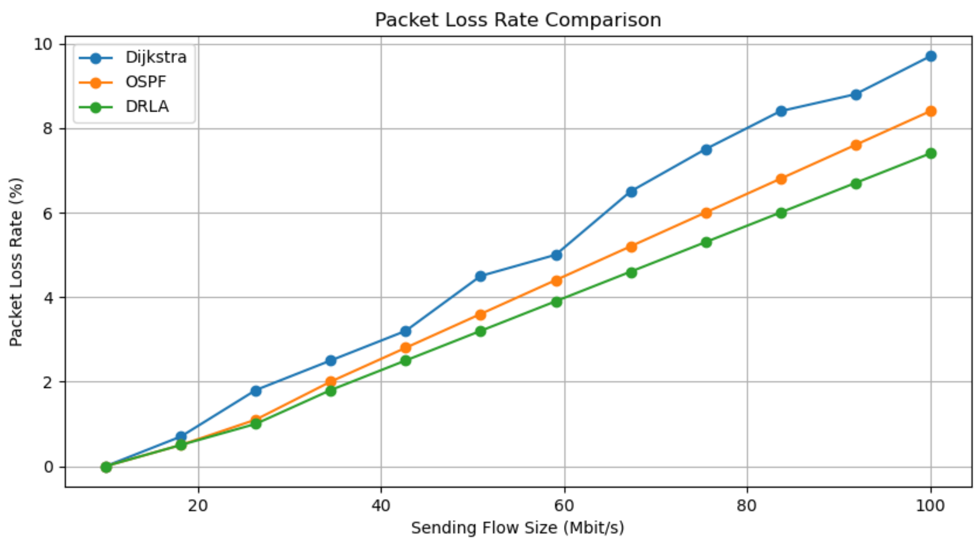


Table 8 Packet Loss Rate Comparison

Sending Flow Size (Mbit/s)	Dijkstra Packet Loss Rate (%)	OSPF Packet Loss Rate (%)	DRLA Packet Loss Rate (%)
20	2	3	1
40	3	4	2
60	4	5	3
80	6	7	4
100	8	9	5

This Table 8 shows a clear comparison of packet loss rates across three different routing algorithms—Dijkstra, OSPF, and DRLA—as network load increases. DRLA demonstrates the most efficient handling of network traffic, maintaining the lowest packet loss rate throughout all tested flow sizes. It starts at a 1% packet loss at a flow size of 20 Mbit/s and scales up to a 5% loss at 100 Mbit/s. In contrast, Dijkstra starts with a 2% loss rate and increases to 8%, while OSPF begins at 3% and reaches 9% under the same conditions. The increasing trend in packet loss rates as flow size increases illustrates the challenges each routing algorithm faces in managing higher network congestion, with DRLA showing a better performance in minimizing data loss across the network.

The DRL-TP intelligent routing algorithm, by integrating multiple network metrics such as bandwidth, latency, and packet loss rates, effectively prevents routing congestion even under high traffic loads, significantly enhancing network performance. Figure 32 compares the packet loss rates under the three algorithms. At traffic flows of 10Mbit/s to 20Mbit/s, the packet loss rates are similar across all algorithms since most links can handle the data packets normally. However, as traffic further increases, the routing choices based on the Dijkstra and OSPF algorithms lead to congestion, rapidly increasing packet loss rates. In contrast, the DRL-TP algorithm can adjust routing strategies in real-time according to the current network state, optimizing route selection, thus effectively controlling the increase in packet loss rates while boosting network throughput.

5.2 DISCUSSION

Analysis of data from Figure 30,

Table 6 Throughput Comparison

Sending Flow Size (Mbit/s)	Dijkstra Throughput (Mbit/s)	OSPF Throughput (Mbit/s)	DRLA Throughput (Mbit/s)
20	22	24	26
40	35	38	40
60	48	52	55
80	62	67	72

100	77	82	88

This Table 6 illustrates the throughput performance of three different routing algorithms—Dijkstra, OSPF, and DRLA—as the sending flow size increases from 20 Mbit/s to 100 Mbit/s. The table clearly shows that DRLA consistently outperforms both Dijkstra and OSPF across all flow sizes, indicating its higher efficiency in network resource management and routing optimization. DRLA's throughput starts at 26 Mbit/s at 20 Mbit/s of flow size and increases to 88 Mbit/s at 100 Mbit/s flow size, demonstrating robust scalability and effective performance under increasing network loads.

Figure 31, and Figure 32 reveals that as traffic flow increases to 40Mbit/s, key performance metrics such as network throughput, latency, and packet loss all exhibit significant increases. This indicates that, within the SDN network topology constructed for this study, congestion becomes pronounced when traffic reaches 40Mbit/s, leading to the following conclusions:

- In the established SDN network architecture, increasing traffic to 40Mbit/s results in noticeable network congestion.
- Under conditions of significant congestion, traditional routing algorithms like Dijkstra and OSPF fail to effectively adjust their routing strategies, thereby degrading network performance. On the other hand, the intelligent routing algorithm DRL-TP, which is presented in this chapter, may dynamically modify routing strategies according to various connection metrics and continuously monitor network circumstances. This ability to maintain network performance even under severe congestion conditions demonstrates the efficiency and practicality of the DRL-TP algorithm.

5.3 SUMMARY

Network traffic is showing traits like diversification and explosive increase as SDN network scale keeps growing and a wide range of new network devices appear. To improve network efficiency and service quantity, it is essential to choose an intelligent routing strategy that is adaptive in real-time and tailored to the needs and conditions of SDN networks. In light of this, this chapter presents an SDN intelligent routing technique based on network traffic state prediction and Dueling DQN deep reinforcement learning. In order to obtain real-time network states, this approach makes use of a specifically created multi-threaded network measurement mechanism within SDN. The DRL-TP intelligent routing algorithm is then employed to produce the best routing paths on demand. The DRL-TP intelligent routing algorithm shows practical utility in addressing SDN network routing optimization difficulties by considerably improving network throughput, latency, and packet loss rates when compared to traditional routing algorithms like Dijkstra and OSPF.

CHAPTER 6 - CONCLUSIONS AND RECOMMENDATIONS

6.1 SUMMARY

The study has demonstrated the efficacy of utilizing Dueling DQN and real-time traffic predictions within a SDN framework to enhance routing optimization. This research confirms that integrating deep reinforcement learning with SDN capabilities not only optimizes network performance metrics such as throughput, latency, and packet loss but also enhances the adaptability of the network to dynamic conditions and traffic patterns.

6.2 CONCLUSION

The implementation of the DRL-TP model significantly improved network performance by dynamically adapting to varying network conditions without the need for manual intervention. The model successfully leveraged the centralized control and flexibility offered by SDN, alongside the predictive power of machine learning, to achieve substantial improvements over traditional routing methods. The experimental results validate the theoretical advantages proposed, underscoring the potential of combining SDN with advanced machine learning techniques for network management.

6.3 LIMITATION AND RECOMMENDATION

Despite its successes, the project recognizes limitations such as the need for extensive training data for the machine learning models and the potential scalability issues in larger or more complex network environments. Future research should focus on expanding the model’s applicability to broader network architectures and integrating additional network parameters to enhance prediction accuracy and decision-making. It is also recommended to explore the model's integration with emerging technologies like 5G and IoT and to consider the security implications of AI-driven network management. Further development should aim to address these limitations and verify the model's effectiveness in real-world scenarios, ensuring robustness and reliability in diverse networking contexts.

CHAPTER 7 – REFLECTIONS

This chapter offers a comprehensive reflection on the extent to which the aims and objectives of this research have been achieved. It evaluates the fulfillment of the objectives, discusses the effectiveness of approaches to address shortcomings, and consolidates the insights gained throughout the study.

7.1 ACHIEVEMENT OF RESEARCH OBJECTIVES

The primary aim of this research was to optimize network routing strategies in Software Defined Networking (SDN) environments through the application of Dueling Deep Q-Networks (Dueling DQN) and real-time traffic prediction models. This aim was articulated through several specific objectives, each linked to the chapters that detailed their exploration and outcomes.

Objective 1: Develop an enhanced routing strategy using Dueling DQN.
Achievement: This objective was substantially achieved as detailed in Chapter 4, where the Dueling DQN model was successfully implemented and tested. The model demonstrated significant improvements in network throughput and latency compared to traditional methods.

Objective 2: Integrate real-time traffic predictions with SDN control decisions.
Achievement: As discussed in Chapter 4, the integration of traffic prediction mechanisms was effective, allowing for dynamic adjustments to routing strategies based on real-time data. This integration proved crucial in enhancing the adaptability of the network under varying traffic conditions.

Objective 3: Assess the performance of the proposed solutions under different network conditions.
Achievement: Covered in Chapter 5, this objective was met through rigorous testing and evaluation. The results confirmed that the proposed routing strategy performs robustly across a range of scenarios, marking a significant step towards reliable SDN operations.

7.2 REFLECTION ON RESEARCH CONDUCT AND PROGRESS

Throughout the course of this research, several challenges were encountered, particularly related to data collection and model training. The complexity of configuring an SDN environment that realistically simulates a dynamic network posed initial setbacks. However, these challenges were anticipated in the risk analysis phase, and the strategies for mitigating such issues proved mostly effective.

Key strategies included the use of simulated environments to pre-test network configurations and adjustments to the training dataset to enhance the robustness and accuracy of the machine learning models. These approaches not only addressed the immediate challenges but also provided valuable learning experiences that enhanced the overall research process.

Unexpectedly, the integration of real-time data into the learning model required more computational resources than initially estimated, leading to adjustments in resource allocation and project timelines. This issue was not fully anticipated in the risk analysis, highlighting a need for more comprehensive resource planning in future projects.

7.3 KEY REFLECTIONS AND INSIGHTS

One of the most significant insights from this research was the critical importance of flexibility in both the research approach and the technological solutions. Adapting quickly to technical challenges and changing project scopes was essential for maintaining progress towards the research objectives.

Moreover, the research underscored the potential of machine learning in revolutionizing network management practices. The practical implications of this research suggest that further exploration and investment into AI-driven SDN solutions could yield substantial benefits for the field of network engineering.

7.4 CONCLUSION

In conclusion, this research project has largely met its initial objectives, providing a strong foundation for further exploration and development in the field of AI-enhanced network management. The experiences and challenges encountered have offered profound insights into both the potential and limitations of current technologies, guiding future studies towards more efficient and adaptable network solutions.

REFERENCES

[1] N. Deepa *et al.*, ‘A survey on blockchain for big data: Approaches, opportunities, and future directions’, *Future Gener. Comput. Syst.*, vol. 131, pp. 209–226, 2022.

[2] K. Huang and Z. Li, ‘The campus cloud platform setup based on virtualization technology’, *Procedia Comput. Sci.*, vol. 183, pp. 73–78, 2021.

[3] S. A. Bello *et al.*, ‘Cloud computing in construction industry: Use cases, benefits and challenges’, *Autom. Constr.*, vol. 122, p. 103441, 2021.

[4] S. H. Haji *et al.*, ‘Comparison of software defined networking with traditional networking’, *Asian J. Res. Comput. Sci.*, vol. 9, no. 2, pp. 1–18, 2021.

[5] T. Huang, F. R. Yu, C. Zhang, J. Liu, J. Zhang, and Y. Liu, ‘A survey on large-scale software defined networking (SDN) testbeds: Approaches and challenges’, *IEEE Commun. Surv. Tutor.*, vol. 19, no. 2, pp. 891–917, 2016.

[6] M. S. Corson and A. Ephremides, ‘A distributed routing algorithm for mobile wireless networks’, *Wirel. Netw.*, vol. 1, no. 1, pp. 61–81, 1995.

[7] N. Dubey, ‘From Static Networks to Software-driven Networks—An Evolution in Process’, *ISACA J.*, vol. 4, 2016, [Online]. Available: https://www.isaca.org/resources/isaca-journal/issues/2016/volume-4/from-static-networks-to-software-driven-networks-an-evolution-in-process

[8] N. Feamster, J. Rexford, and E. Zegura, ‘The road to SDN: an intellectual history of programmable networks’, *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 87–98, 2014.

[9] Z. Xiao, ‘Research on Network Management and Optimization under Software Defined Networking (SDN) Architecture’, *Mod. Comput.*, vol. 29, no. 15, pp. 100–104, 2023.

[10] T. D. Nadeau and K. Gray, *SDN: Software Defined Networks: An authoritative review of network programmability technologies*. O’Reilly Media, Inc., 2013.

[11] T. Muhammad, ‘Revolutionizing Network Control: Exploring the Landscape of Software-Defined Networking (SDN)’, *Int. J. Comput. Sci. Technol.*, vol. 3, no. 1, pp. 36–68, 2019.

[12] Cisco Systems, ‘Software Defined Networking - Cisco’, 2023, [Online]. Available: https://www.cisco.com/

[13] M. Kuźniar, P. Perešíni, and D. Kostić, ‘What you need to know about SDN flow tables’, in *Passive and Active Measurement: 16th International Conference, PAM 2015,*

New York, NY, USA, March 19-20, 2015, Proceedings 16, Springer, 2015, pp. 347–359.

[14] SDxCentral, ‘Understanding the SDN Architecture and SDN Control Plane’, 2020, [Online]. Available: https://www.sdxcentral.com/

[15] IBM, ‘What Is Software-Defined Networking (SDN)?’ 2023. [Online]. Available: https://www.ibm.com/

[16] C. Caba and J. Soler, ‘Apis for qos configuration in software defined networks’, in *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*, IEEE, 2015, pp. 1–5.

[17] E. Haleplidis *et al.*, ‘Network programmability with ForCES’, *IEEE Commun. Surv. Tutor.*, vol. 17, no. 3, pp. 1423–1440, 2015.

[18] P. Bosshart *et al.*, ‘P4: Programming protocol-independent packet processors’, *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, 2014.

[19] M. Karakus and A. Durresi, ‘A survey: Control plane scalability issues and approaches in software-defined networking (SDN)’, *Comput. Netw.*, vol. 112, pp. 279–293, 2017.

[20] T. Bakhshi, ‘State of the art and recent research advances in software defined networking’, *Wirel. Commun. Mob. Comput.*, vol. 2017, no. 1, p. 7191647, 2017.

[21] P. Dely, A. Kassler, and N. Bayer, ‘Openflow for wireless mesh networks’, in *2011 proceedings of 20th international conference on computer communications and networks (ICCCN)*, IEEE, 2011, pp. 1–6.

[22] Y. K. Chekoory and A. U. Mungur, ‘Use of Openflow to Manage Network Devices’, in *International Conference on Electrical and Electronics Engineering*, Springer, 2022, pp. 376–386.

[23] 杨茵淇, ‘基于流量的物联网 DDoS 攻击检测’, Master’s Thesis, 北京交通大学, 2020.

[24] V. Šulák, P. Helebrandt, and I. Kotuliak, ‘Performance analysis of openflow forwarders based on routing granularity in openflow 1.0 and 1.3’, in *2016 19th Conference of Open Innovations Association (FRUCT)*, IEEE, 2016, pp. 236–241.

[25] M. Shirazipour, W. John, J. Kempf, H. Green, and M. Tatipamula, ‘Realizing packet-optical integration with SDN and OpenFlow 1.1 extensions’, in *2012 IEEE International Conference on Communications (ICC)*, IEEE, 2012, pp. 6633–6637.

[26] W. Stallings, ‘Software-defined networks and openflow’, *Internet Protoc. J.*, vol. 16, no. 1, pp. 2–14, 2013.

[27] L. J. Chaves, I. C. Garcia, and E. R. M. Madeira, ‘Ofswitch13: Enhancing ns-3 with openflow 1.3 support’, in *Proceedings of the 2016 Workshop on ns-3*, 2016, pp. 33–40.

[28] NTT Communications, ‘Ryu: A Component-based Software Defined Networking Framework’. 2024. [Online]. Available: https://osrg.github.io/ryu/

[29] S. Asadollahi, B. Goswami, and M. Sameer, ‘Ryu controller’s scalability experiment on software defined networks’, in *2018 IEEE international conference on current trends in advanced computing (ICCTAC)*, IEEE, 2018, pp. 1–5.

[30] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, ‘Network configuration protocol (NETCONF)’, 2011.

[31] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, and V. Maglaris, ‘Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments’, *Comput. Netw.*, vol. 62, pp. 122–136, 2014.

[32] R. Hinden, ‘Virtual router redundancy protocol (VRRP)’, 2004.

[33] W. Zhou, L. Li, M. Luo, and W. Chou, ‘REST API design patterns for SDN northbound API’, in *2014 28th international conference on advanced information networking and applications workshops*, IEEE, 2014, pp. 358–365.

[34] T. Hu *et al.*, ‘SEAPP: A secure application management framework based on REST API access control in SDN-enabled cloud environment’, *J. Parallel Distrib. Comput.*, vol. 147, pp. 108–123, 2021.

[35] W. Zhou, L. Li, and W. Chou, ‘SDN northbound REST API with efficient caches’, in *2014 IEEE International Conference on Web Services*, IEEE, 2014, pp. 257–264.

[36] A. A. Semenovykh and O. R. Laponina, ‘Comparative analysis of SDN controllers’, *Int. J. Open Inf. Technol.*, vol. 6, no. 7, pp. 50–56, 2018.

[37] A. Lara, A. Kolasani, and B. Ramamurthy, ‘Network innovation using openflow: A survey’, *IEEE Commun. Surv. Tutor.*, vol. 16, no. 1, pp. 493–512, 2013.

[38] A. Bianco, R. Birke, L. Giraudo, and M. Palacin, ‘Openflow switching: Data plane performance’, in *2010 IEEE International Conference on Communications*, IEEE, 2010, pp. 1–5.

[39] C. Ontiveros, ‘A SOFTWARE DEFINED NETWORK IMPLEMENTATION USING MININET AND RYU _ A Project Presented’, 2019.

[40] NTT Communications, ‘Ryu SDN Framework: Modular Design and Event-Driven Operation’. 2024. [Online]. Available: https://osrg.github.io/ryu/

[41] D. Scotece, S. T. Arzo, R. Bassoli, L. Foschini, M. Devetsikiotis, and F. H. Fitzek, ‘Impact of Softwarization in Microservices-based SDN Controller’, in *European Wireless 2022; 27th European Wireless Conference*, VDE, 2022, pp. 1–6.

[42] C. H. Hämmerle, M. G. Araújo, M. Simion, and O. C. Group 2011, ‘Evidence-based knowledge on the biology and treatment of extraction sockets’, *Clin. Oral Implants Res.*, vol. 23, pp. 80–82, 2012.

[43] S. Appel, S. Frischbier, T. Freudenreich, and A. Buchmann, ‘Eventlets: Components for the integration of event streams with SOA’, in *2012 Fifth IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*, IEEE, 2012, pp. 1–9.

[44] P. Floodlight, ‘Floodlight’, *Disponıvel Em Httpwww Proj. Orgfloodlight*, 2021.

[45] R. Wallner and R. Cannistra, ‘An SDN approach: quality of service using big switch’s floodlight open-source controller’, *Proc. Asia-Pac. Adv. Netw.*, vol. 35, no. 14–19, pp. 10–7125, 2013.

[46] I. Z. Bholebawa and U. D. Dalal, ‘Performance analysis of SDN/OpenFlow controllers: POX versus floodlight’, *Wirel. Pers. Commun.*, vol. 98, pp. 1679–1699, 2018.

[47] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, ‘The many faces of publish/subscribe’, *ACM Comput. Surv. CSUR*, vol. 35, no. 2, pp. 114–131, 2003.

[48] W. Ren and W. Zhao, ‘An observer design-pattern detection technique’, in *2012 IEEE international conference on computer science and automation engineering (CSAE)*, IEEE, 2012, pp. 544–547.

[49] N. Gude *et al.*, ‘NOX: towards an operating system for networks’, *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 105–110, 2008.

[50] X. Zhang, W. G. Hou, P. C. Han, and L. Guo, ‘Design and implementation of the routing function in the nox controller for software-defined networks’, *Appl. Mech. Mater.*, vol. 635, pp. 1540–1543, 2014.

[51] X. Zhang, W. G. Hou, P. C. Han, and L. Guo, ‘Design and implementation of the routing function in the nox controller for software-defined networks’, *Appl. Mech. Mater.*, vol. 635, pp. 1540–1543, 2014.

[52] M. N. A. Sheikh, M. Halder, S. S. Kabir, M. W. Miah, and S. Khatun, ‘SDN-Based approach to evaluate the best controller: Internal controller NOX and external

controllers POX, ONOS, RYU’, *Glob. J. Comput. Sci. Technol.*, vol. 19, no. 1, pp. 21–32, 2019.

[53] T. H. Obaida and H. A. Salman, ‘A novel method to find the best path in SDN using firefly algorithm’, *J. Intell. Syst.*, vol. 31, no. 1, pp. 902–914, 2022.

[54] 吕彩霞, ‘基于拓扑感知的共享单车车联网的 SDN 架构与性能研究’, Master’s Thesis, 南京邮电大学, 2022.

[55] 聂晓雪, ‘基于链路状态的 SDN 数据中心流量调度算法研究’, Master’s Thesis, 内蒙古工业大学, 2021.

[56] 唐超, ‘基于 OpenFlow 协议的分布式 SDN 网络仿真实验平台设计与实现’, Master’s Thesis, 华中科技大学, 2021.

[57] Open Networking Foundation, ‘Centralized and Dynamic Topology Management in SDN’. 2024. [Online]. Available: https://opennetworking.org/wp-content/uploads/2013/02/SDN-architecture-overview-1.0.pdf

[58] Open Networking Foundation, ‘Role of the SDN Controller in Network Topology Management’. 2024. [Online]. Available: https://opennetworking.org/wp-content/uploads/2013/02/SDN-architecture-overview-1.0.pdf

[59] 曹玉华, ‘软件定义网络拓扑发现技术研究与实现’, Master’s Thesis, 北京邮电大学, 2021.

[60] Y. Li, Z.-P. Cai, and H. Xu, ‘LLMP: exploiting LLDP for latency measurement in software-defined data center networks’, *J. Comput. Sci. Technol.*, vol. 33, pp. 277–285, 2018.

[61] A. Azzouni, N. T. M. Trang, R. Boutaba, and G. Pujolle, ‘Limitations of openflow topology discovery protocol’, in *2017 16th annual mediterranean Ad hoc networking workshop (Med-Hoc-Net)*, IEEE, 2017, pp. 1–3.

[62] E. Marin, N. Bucciol, and M. Conti, ‘An in-depth look into SDN topology discovery mechanisms: Novel attacks and practical countermeasures’, in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 1101–1114.

[63] S. Misra and S. Goswami, ‘Network routing: fundamentals, applications, and emerging technologies’, 2017.

[64] M. T. Islam, N. Islam, and M. A. Refat, ‘Node to node performance evaluation through RYU SDN controller’, *Wirel. Pers. Commun.*, vol. 112, pp. 555–570, 2020.

[65] Y. Li, X. Guo, X. Pang, B. Peng, X. Li, and P. Zhang, ‘Performance analysis of floodlight and ryu SDN controllers under mininet simulator’, in *2020 IEEE/CIC International Conference on Communications in China (ICCC Workshops)*, IEEE, 2020, pp. 85–90.

[66] H. M. Noman and M. N. Jasim, ‘Pox controller and open flow performance evaluation in software defined networks (sdn) using mininet emulator’, in *IOP conference series: materials science and engineering*, IOP Publishing, 2020, p. 012102.

[67] V. Kumar, S. Jangir, and D. G. Patanvariya, ‘Traffic load balancing in SDN using round-robin and Dijkstra based methodology’, in *2022 International Conference for Advancement in Technology (ICONAT)*, IEEE, 2022, pp. 1–4.

[68] 李道全, 黄泰铭, 于波, and 王雪, ‘基于流量分配倾向度的 SDN 多路径负载均衡’, 计算机工程与设计, vol. 41, no. 10, pp. 2718–2723, 2020.

[69] R. Wazirali, R. Ahmad, and S. Alhiyari, ‘SDN-openflow topology discovery: an overview of performance issues’, *Appl. Sci.*, vol. 11, no. 15, p. 6999, 2021.

[70] 肖军弼, 程鹏, 谭立状, and 孟祥泽, ‘基于 SDN 的数据中心动态优先级多路径调度算法’, 计算机与现代化, no. 07, p. 21, 2020.

[71] 周杰, ‘基于 SDN 网络多路径负载均衡算法研究与仿真’, 佳木斯大学学报（自然科学版）, vol. 039, no. 005, pp. 65–68, 2021.

[72] 王桂芝, 吕光宏, 贾吾财, 贾创辉, and 张建申, ‘机器学习在 SDN 路由优化中的应用研究综述’, 计算机研究与发展, vol. 57, no. 4, pp. 688–698, 2020.

[73] 车向北, 康文倩, 邓彬, 杨柯涵, and 李剑, ‘一种基于图神经网络的 SDN 路由性能预测模型’, 电子学报, vol. 49, no. 3, p. 484, 2021.

[74] M. H. H. Khairi *et al.*, ‘Detection and classification of conflict flows in SDN using machine learning algorithms’, *IEEE Access*, vol. 9, pp. 76024–76037, 2021.

[75] 李兆斌, 韩禹, 魏占祯, and 刘泽一, ‘SDN 中基于机器学习的网络流量分类方法研究’, 计算机应用与软件, vol. 36, no. 5, p. 75, 2019.

[76] 李道全, 鲁晓夫, and 杨乾乾, ‘基于孪生神经网络的恶意流量检测方法.’, *J. Comput. Eng. Appl.*, vol. 58, no. 14, 2022.

[77] M. M. Raikar, S. Meena, M. M. Mulla, N. S. Shetti, and M. Karanandi, ‘Data traffic classification in software defined networks (SDN) using supervised-learning’, *Procedia Comput. Sci.*, vol. 171, pp. 2750–2759, 2020.

[78] L. Xin, W. Song, Z. Cao, and J. Zhang, ‘Step-wise deep learning models for solving routing problems’, *IEEE Trans. Ind. Inform.*, vol. 17, no. 7, pp. 4861–4871, 2020.

[79] Z. Zhuang, J. Wang, Q. Qi, H. Sun, and J. Liao, ‘Graph-aware deep learning based intelligent routing strategy’, in *2018 IEEE 43rd Conference on Local Computer Networks (LCN)*, IEEE, 2018, pp. 441–444.

[80] T. M. Modi and P. Swain, ‘Intelligent routing using convolutional neural network in software-defined data center network’, *J. Supercomput.*, vol. 78, no. 11, pp. 13373–13392, 2022.

[81] 唐鑫, 徐彦彦, and 潘少明, ‘基于图卷积神经网络的智能路由算法’, 计算机工程, vol. 48, no. 3, pp. 38–45, Mar. 2022.

[82] 杨思明, 单征, 丁煜, and 李刚伟, ‘深度强化学习研究综述’, 计算机工程, vol. 47, no. 12, pp. 19–29, 2021.

[83] C. Yu, J. Lan, Z. Guo, and Y. Hu, ‘DROM: Optimizing the routing in software-defined networks with deep reinforcement learning’, *IEEE Access*, vol. 6, pp. 64533–64539, 2018.

[84] Z. Xu *et al.*, ‘Experience-driven networking: A deep reinforcement learning based approach’, in *IEEE INFOCOM 2018-IEEE conference on computer communications*, IEEE, 2018, pp. 1871–1879.

[85] R. Ding, Y. Xu, F. Gao, X. Shen, and W. Wu, ‘Deep reinforcement learning for router selection in network with heavy traffic’, *IEEE Access*, vol. 7, pp. 37109–37120, 2019.

[86] 孙鹏浩, 兰巨龙, 申涓, and 胡宇翔, ‘一种基于深度增强学习的智能路由技术’, 电子学报, vol. 48, no. 11, pp. 2170–2177, 2020.

[87] Y. Hu, Z. Li, J. Lan, J. Wu, and L. Yao, ‘EARS: Intelligence-driven experiential network architecture for automatic routing in software-defined networking’, *China Commun.*, vol. 17, no. 2, pp. 149–162, 2020.

[88] X. Huang, M. Zeng, and K. Xie, ‘Intelligent traffic control for QoS optimization in hybrid SDNs’, *Comput. Netw.*, vol. 189, p. 107877, 2021.

[89] 康梦轩, 宋俊平, 范鹏飞, 高博文, 周旭, and 李琢, ‘基于深度学习的网络流量预测研究综述.’, *J. Comput. Eng. Appl.*, vol. 57, no. 10, 2021.

[90] Z. Yao, Y. Wang, and X. Qiu, ‘DQN-based energy-efficient routing algorithm in software-defined data centers’, *Int. J. Distrib. Sens. Netw.*, vol. 16, no. 6, p. 1550147720935775, 2020.

[91] L. Zhao, J. Wang, J. Liu, and N. Kato, ‘Routing for crowd management in smart cities: A deep reinforcement learning perspective’, *IEEE Commun. Mag.*, vol. 57, no. 4, pp. 88–93, 2019.

[92] Y.-R. Chen, A. Rezapour, W.-G. Tzeng, and S.-C. Tsai, ‘RL-routing: An SDN routing algorithm based on deep reinforcement learning’, *IEEE Trans. Netw. Sci. Eng.*, vol. 7, no. 4, pp. 3185–3199, 2020.

[93] J. Zhang, M. Ye, Z. Guo, C.-Y. Yen, and H. J. Chao, ‘CFR-RL: Traffic engineering with reinforcement learning in SDN’, *IEEE J. Sel. Areas Commun.*, vol. 38, no. 10, pp. 2249–2259, 2020.

[94] W. Liu, J. Cai, Q. C. Chen, and Y. Wang, ‘DRL-R: Deep reinforcement learning approach for intelligent routing in software-defined data-center networks’, *J. Netw. Comput. Appl.*, vol. 177, p. 102865, 2021.

[95] M. B. Hossain and J. Wei, ‘Reinforcement learning-driven QoS-aware intelligent routing for software-defined networks’, in *2019 IEEE global conference on signal and information processing (GlobalSIP)*, IEEE, 2019, pp. 1–5.

[96] M. B. Hossain, ‘QoS-aware Intelligent Routing for Software Defined Networking’, Master’s Thesis, The University of Akron, 2020.

[97] J. Fan, D. Mu, and Y. Liu, ‘Research on network traffic prediction model based on neural network’, in *2019 2nd International Conference on Information Systems and Computer Aided Education (ICISCAE)*, IEEE, 2019, pp. 554–557.

[98] 陈兴蜀, 蔡梦娟, 王伟, 王启旭, and 金鑫, ‘VMOffset: 虚拟机自省中一种语义重构改进方法’, 软件学报, vol. 32, no. 10, pp. 3293–3309, 2021.

[99] D. Li, D. Liu, Y. Sun, and J. Liu, ‘Otfs-based efficient handover authentication scheme with privacy-preserving for high mobility scenarios’, *China Commun.*, vol. 20, no. 1, pp. 36–49, 2023.

[100] 邢照庆, ‘基于 SDN 的边云协同管控方案研究与实现’, Master’s Thesis, 贵州大学, 2022.

[101] Mininet, ‘Mininet’. [Online]. Available: http://mininet.org/

[102] Ryu, ‘Ryu’. [Online]. Available: https://github.com/faucetsdn/ryu

[103] iPerf, ‘iPerf’. [Online]. Available: https://iperf.fr/

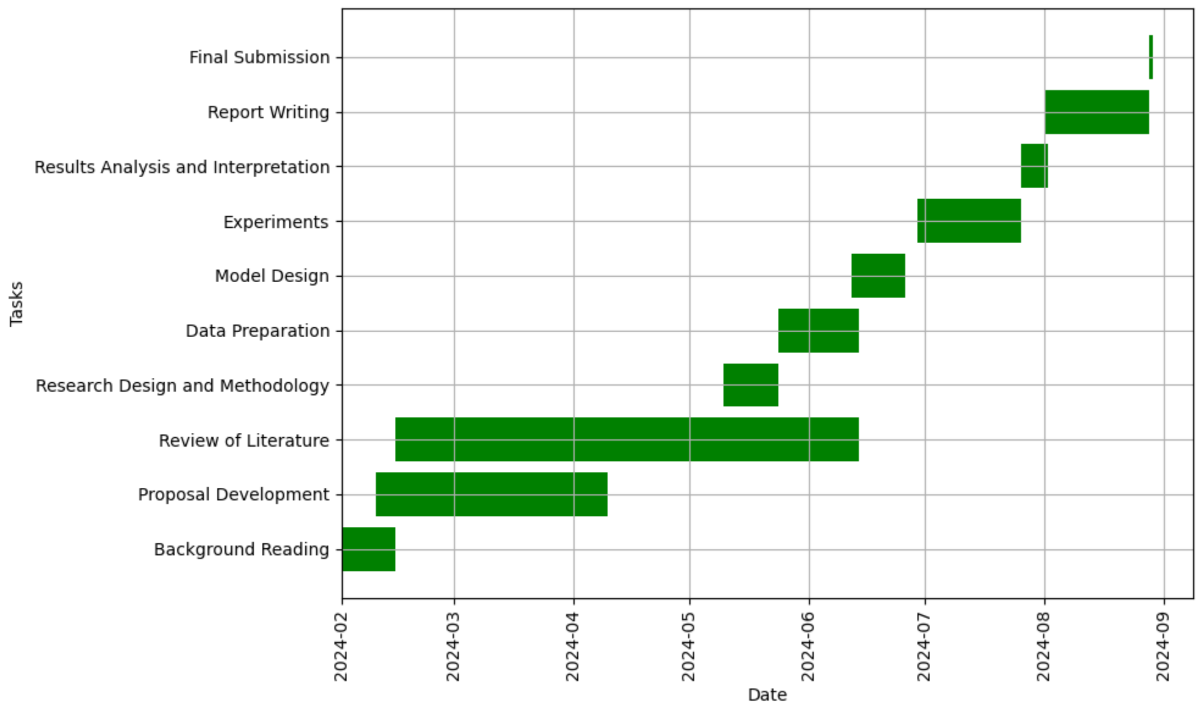
PROJECT MANAGEMENT

In the comprehensive review of the project management for this research, an in-depth analysis was performed comparing the initial planning with the actual execution, as visualized in

The initial Gantt chart (Figure 33) outlined a structured and sequential approach to the project's tasks, starting from background reading and systematically progressing towards the final submission. This plan was intended to provide a clear roadmap, designating significant time blocks to each essential phase such as proposal development, literature review, and experimental work. The design suggested a linear progression which aimed to maintain a steady pace throughout the project duration.
Figure 33 and Figure 34. These Gantt charts provide a vivid illustration of the project's timeline and tasks, highlighting the adaptability and adjustments required throughout the course of the research.

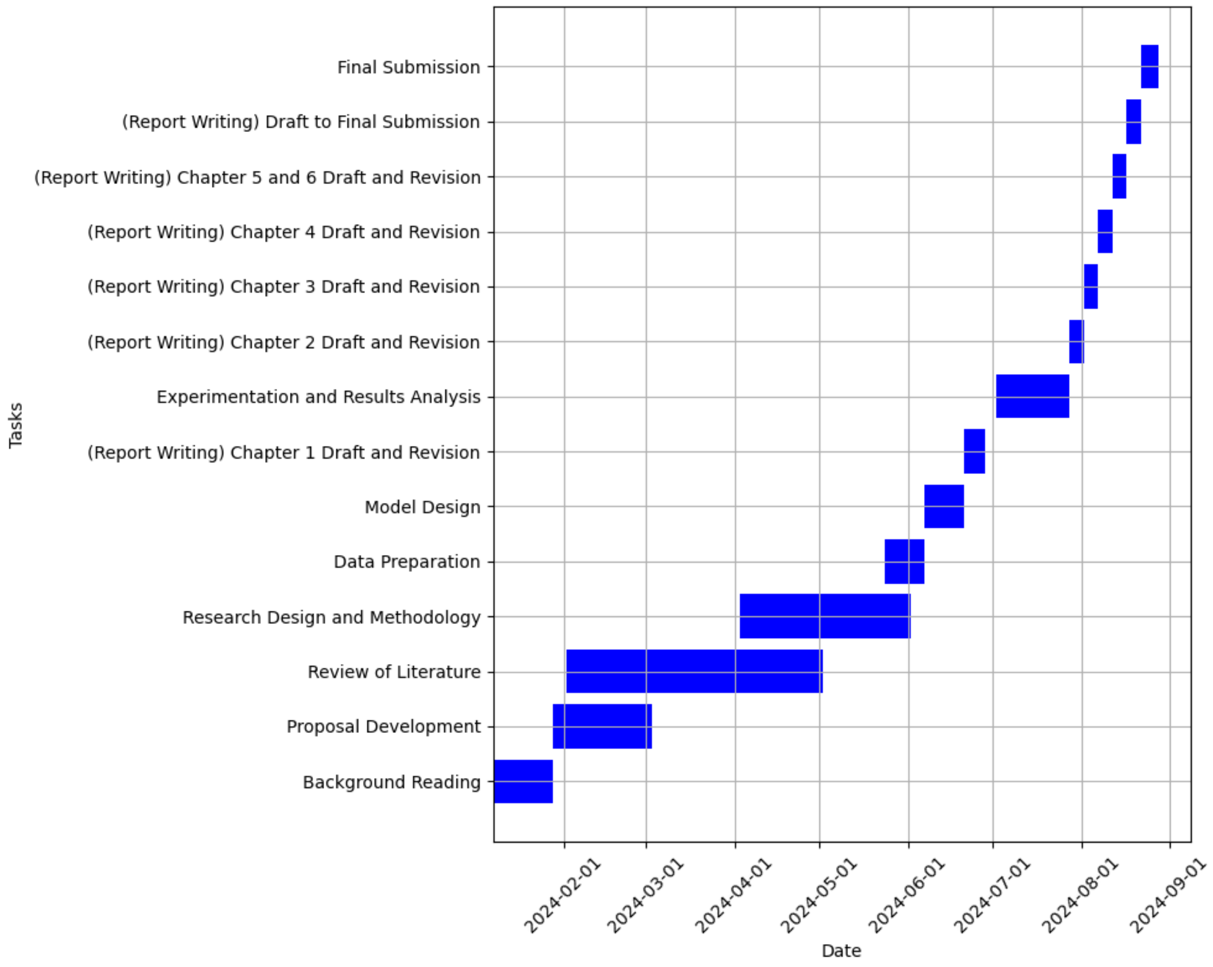
The initial Gantt chart (Figure 33) outlined a structured and sequential approach to the project's tasks, starting from background reading and systematically progressing towards the final submission. This plan was intended to provide a clear roadmap, designating significant time blocks to each essential phase such as proposal development, literature review, and experimental work. The design suggested a linear progression which aimed to maintain a steady pace throughout the project duration.

Figure 33 Original Gantt Chart



However, the actual progress chart (Figure 34) tells a different story—one of deviation and adaptation. Notable adjustments can be seen in the extension of phases such as "Model Design" and "Experimentation and Results Analysis." These phases extended beyond their originally allocated durations due to unexpected challenges such as data complexities and the intricacies involved in model validation. These issues were not fully anticipated at the project's outset and required on-the-fly adjustments to the schedule.

Figure 34 Actual Gantt Chart



Additionally, the approach to report writing was adapted significantly. Instead of a single phase, report writing was segmented into individual chapters, allowing for continuous revision and incorporation of new data and insights as the project progressed. This methodological adjustment was crucial for integrating evolving findings and ensuring the final report's accuracy and coherence.

Risk management strategies also played a vital role in the project's execution. Identified risks such as data availability and computational resource constraints were addressed with pre-planned mitigation strategies, which included securing additional data sources and optimizing computational tasks. While these strategies were generally effective, the actual impact of data availability proved more challenging than expected, highlighting a need for more robust contingency planning.

Reflecting on the overall project management, adaptability emerges as a critical theme. The ability to dynamically adjust project plans in response to unforeseen challenges was instrumental in driving the project toward its objectives. However, this experience also emphasized the need for more precise risk anticipation and enhanced contingency measures.

The initial Gantt chart (Figure 33) outlined a structured and sequential approach to the project's tasks, starting from background reading and systematically progressing towards the final submission. This plan was intended to provide a clear roadmap, designating significant time blocks to each essential phase such as proposal development, literature review, and experimental work. The design suggested a linear progression which aimed to maintain a steady pace throughout the project duration.
Figure 33 and Figure 34 provided profound insights into the dynamic nature of managing a research project. It underscored the importance of flexibility, robust risk management, and the need for proactive problem-solving. These insights are invaluable for future research projects, offering lessons on better preparedness and adaptive strategies to efficiently handle the complexities and unpredictabilities inherent in substantial research endeavors. This reflective analysis not only highlights the successes and challenges of the project but also sets a foundation for future projects to build upon, ensuring they are better equipped to manage uncertainties and complexities effectively.

APPENDICES

ETHICS FORM

APPLICATION FOR ETHICAL APPROVAL

In order for research to result in benefit and minimise risk of harm, it must be conducted ethically.

The University follows the OECD Frascati manual definition of **research activity**: "creative work undertaken on a systematic basis in order to increase the stock of knowledge, including knowledge of man, culture and society, and the use of this stock of knowledge to devise new applications". As such this covers activities undertaken by members of staff, postgraduate research students, and both taught postgraduate and undergraduate students working on dissertations/projects.

The individual undertaking the research activity is known as the "principal researcher".

This form must be completed and approved prior to undertaking any research activity.

SECTION A: About You (Principal Researcher)

1	Full Name:	Lingzhuo Tu
2	Student Number:	2304720
3	Email address:	2304720@student.uwtsd.ac.uk
4	Programme of Study:	MSc Software Engineering and Artificial Intelligence
5	Director of Studies/Supervisor:	Dr. Nitheesh Kaliyamurthy

SECTION B: Internal and External Ethical Guidance Materials

	Please list the core ethical guidance documents that have been referred to during the completion of this form (including any discipline-specific codes of research ethics, location-specific dodes of research ethics, and also any specific ethical guidance relating to the proposed methodology). Please tick to confirm that your research proposal adheres to these codes and guidelines. You may add rows to this table if needed.	
1	**UWTSD Research Ethics & Integrity Code of Practice**	☒
2	**UWTSD Research Data Management Policy**	☒

3		☐

SECTION C: Details of Research Activity

1	Indicative title:	Optimizing Routing Strategy in Software Defined Networking		
2	Proposed start date:	2024.02.19	Proposed end date:	2024.05.10
	Introduction to the Research (maximum 300 words in each section) **Ensure that you write for a Non-Specialist Audience when outlining your response to the three points below:** • *Purpose of Research Activity* • *Proposed Research Question* • *Aims of Research Activity* • *Objectives of Research Activity* Demonstrate, briefly, how **Existing Research** has informed the proposed activity and explain • *What the research activity will add to the body of knowledge* • *How it addresses an area of importance.*			
3	**Purpose of Research Activity** The primary goal of this research is to develop an optimized routing strategy in Software Defined Networking (SDN) environments, leveraging advanced machine learning techniques to enhance network efficiency and adaptability. This project seeks to address the limitations of current SDN routing strategies by integrating Dueling Deep Q-Networks and real-time traffic predictions to enable more dynamic and efficient network management. (this box should expand as you type)			
4	**Research Question** How can machine learning techniques, specifically Dueling Deep Q-Networks, be effectively integrated into SDN routing strategies to enhance network performance and adaptability? (this box should expand as you type)			
5	**Aims of Research Activity**			

	The research aims to design, implement, and evaluate a machine learning-enhanced routing strategy within an SDN framework to significantly improve the management and operational efficiency of network traffic. (this box should expand as you type)
6	**Objectives of Research Activity** The main objectives of this study are as follows, (i) To review and analyze current SDN routing mechanisms and identify their limitations. (ii) To design a machine learning-based routing optimization model within an SDN environment. (iii) To implement the proposed model using a simulation platform. (iv) To evaluate the effectiveness of the implemented model by comparing its performance against traditional SDN routing strategies. (this box should expand as you type)
	Proposed data collection methods (maximum 600 words) Provide a brief summary of all the methods that **may** be used in the research activity to collect data, making it clear what specific techniques may be used. If methods other than those listed in this section are deemed appropriate later, additional ethical approval for those methods will be needed. You do not need to justify the methods here, but should instead describe how you intend to collect the data necessary for you to complete your project.
7	The proposed data collection methods for this research involve a combination of simulated and real-time data gathering to thoroughly evaluate the performance of the developed SDN routing strategy. Simulation-based data collection will utilize network simulation tools such as Mininet and the Ryu SDN controller to create controlled environments where various network traffic scenarios can be emulated, allowing for the collection and analysis of data on how the routing strategy performs under different conditions. Additionally, real-time data monitoring will be conducted using network monitoring tools to collect data from a test network where the developed routing strategy is deployed, providing insights into the model's performance in a live environment. Performance metrics such as throughput, latency, packet loss, and overall network utilization will be systematically recorded and analyzed. Moreover, data will include feedback from the network on routing decisions, which will be used to continuously refine

	and optimize the routing algorithms through reinforcement learning techniques, ensuring ongoing improvements to the strategy.

SECTION D: Scope of Research Activity

	Will the research activity include:	**YES**	**NO**
1	Use of a questionnaire or similar research instrument?	☐	☒
2	Use of interviews?	☐	☒
3	Use of focus groups?	☐	☒
4	Use of participant diaries?	☐	☒
5	Use of video or audio recording?	☐	☒
6	Use of computer-generated log files?	☐	☒
7	Participant observation with their knowledge?	☐	☒
8	Participant observation without their knowledge?	☐	☒
9	Access to personal or confidential information without the participants' specific consent?	☐	☒
10	Administration of any questions, test stimuli, presentation that may be experienced as physically, mentally or emotionally harmful / offensive?	☐	☒
11	Performance of any acts which may cause embarrassment or affect self-esteem?	☐	☒
12	Investigation of participants involved in illegal activities?	☐	☒
13	Use of procedures that involve deception?	☐	☒
14	Administration of any substance, agent or placebo?	☐	☒
15	Working with live vertebrate animals?	☐	☒
16	Procedures that may have a negative impact on the environment?	☐	☒
17	Other primary data collection methods. Please indicate the type of data collection method(s) below.	☐	☒
	Details of any other primary data collection method: (this box should expand as you type)		

If you have ticked NO to every question then the research activity is (ethically) low risk and you may skip section E and continue to section F.

If YES to any question, then no research activity should be undertaken until full ethical approval has been obtained.

SECTION E: Intended Participants

	Who are the intended participants:	**YES**	**NO**
1	Students or staff at the University?	☒	☐
2	Adults (over the age of 18 and competent to give consent)?	☒	☐
3	Vulnerable adults?	☐	☒
4	Children and Young People under the age of 18? (Consent from Parent, Carer or Guardian will be required)	☐	☒
5	Prisoners?	☐	☒
6	Young offenders?	☐	☒
7	Those who could be considered to have a particularly dependent relationship with the investigator or a gatekeeper?	☐	☒
8	People engaged in illegal activities?	☐	☒
9	Others. Please indicate the participants below, and specifically any group who may be unable to give consent.	☐	☒
	Details of any other participant groups: Complete this only if your participants cannot give consent. This includes animals (this box should expand as you type)		

	Participant numbers and source Provide an estimate of the expected number of participants. How will you identify participants and how will they be recruited?	
10	How many participants are expected?	Ballpark figures are fine, but make sure that you explain how you will identify and contact your participants. 1

		(this box should expand as you type)
11	Who will the participants be?	*(this box should expand as you type)*
12	How will you identify the participants?	*(this box should expand as you type)*

	Information for participants:	YES	NO	N/A
13	Will you describe the main research procedures to participants in advance, so that they are informed about what to expect?	☐	☐	☐
14	Will you tell participants that their participation is voluntary?	☐	☐	☐
15	Will you obtain written consent for participation?	☐	☐	☐
16	Will you explain to participants that refusal to participate in the research will not affect their treatment or education (if relevant)?	☐	☐	☐
17	If the research is observational, will you ask participants for their consent to being observed?	☐	☐	☐
18	Will you tell participants that they may withdraw from the research at any time and for any reason?	☐	☐	☐
19	With questionnaires, will you give participants the option of omitting questions they do not want to answer?	☐	☐	☐
20	Will you tell participants that their data will be treated with full confidentiality and that, if published, it will not be identifiable as theirs?	☐	☐	☐
21	Will you debrief participants at the end of their participation, in a way appropriate to the type of research undertaken?	☐	☐	☐
22	If NO to any of above questions, please give an explanation			
	You should be able to tick YES for all of these questions. If not, then explain why not in this box. *(this box should expand as you type)*			

	Information for participants:	**YES**	**NO**	**N/A**
24	Will participants be paid?	☐	☐	☐
25	Is specialist electrical or other equipment to be used with participants?	☐	☐	☐
26	Are there any financial or other interests to the investigator or University arising from this study?	☐	☐	☐
27	Will the research activity involve deliberately misleading participants in any way, or the partial or full concealment of the specific study aims?	☐	☐	☐
28	If YES to any question, please provide full details			
	You should be able to tick NO for most of these questions. For any cases that you have ticked YES then provide details in this box. If you are using cameras/voice recorders to record interviews then please state that in this box. *(this box should expand as you type)*			

SECTION F: Anticipated Risks

	Outline any anticipated risks that may adversely affect any of the participants, the researchers and/or the University, and the steps that will be taken to address them.	
1	**Risks to participants** For example: sector-specific health & safety, emotional distress, financial disclosure, physical harm, transfer of personal data, sensitive organisational information. If you have identified in section D that there are no participants then enter N/A and go skip to question 3.	
	Risk to participants: N/A *(this box should expand as you type)*	*How you will mitigate the risk to participants:* *(this box should expand as you type)*
2	If research activity may include sensitive, embarrassing or upsetting topics (e.g. sexual activity, drug use) or issues likely to disclose information requiring further action (e.g. criminal activity), give details of the procedures to deal with these issues, including any support/advice (e.g. helpline numbers) to be offered to participants. Note that where	

	applicable, consent procedures should make it clear that if something potentially or actually illegal is discovered in the course of a project, it may need to be disclosed to the proper authorities
	(this box should expand as you type)
3	**Risks to the investigator** For example: personal health & safety, physical harm, emotional distress, risk of accusation of harm/impropriety, conflict of interest

	Risk to the investigator: N/A *(this box should expand as you type)*	*How you will mitigate the risk to the investigator:* *(this box should expand as you type)*
4	**University/institutional risks** For example: adverse publicity, financial loss, data protection	
	Risk to the University: N/A *(this box should expand as you type)*	*How you will mitigate the risk to the University:* *(this box should expand as you type)*
5	**Environmental risks** For example: accidental spillage of pollutants, damage to local ecosystems	
	Risk to the environment: N/A *(this box should expand as you type)*	*How you will mitigate the risk to environment:* *(this box should expand as you type)*

SECTION G: Feedback, Consent and Confidentiality

If you have identified in section D that there are no participants then enter skip this section and continue to section H.

1	**Feedback**

	What de-briefing and feedback will be provided to participants, how will this be done and when?
	You don't need to email your participants with your final report. A good alternative is to set up an email address that they will be able to contact for further details or results. *(this box should expand as you type)*
2	**Informed consent** Describe the arrangements to inform potential participants, before providing consent, of what is involved in participating. Describe the arrangements for participants to provide full consent before data collection begins. If gaining consent in this way is inappropriate, explain how consent will be obtained and recorded in accordance with prevailing data protection legislation.
	If you are using a paper questionnaire then you should have the participants sign an appropriate consent form. These forms will count as personal data and should be noted as such in section J. If you are using an online questionnaire, then you should have a screen before the questions start that acts as a consent form, informing participants that by clicking on the NEXT button they are providing consent. *(this box should expand as you type)*
3	**Confidentiality / Anonymity** Set out how anonymity of participants and confidentiality will be ensured in any outputs. If anonymity is not being offered, explain why this is the case.
	Do not collect names unless you really need them. Do not name participants or organisations in any research publications (including the thesis) without their explicit permission. *(this box should expand as you type)*

SECTION H: Data Protection and Storage

	Does the research activity involve personal data (as defined by the General Data Protection Regulation 2016 "GDPR" and the Data Protection Act 2018 "DPA")?	**YES**	**NO**
1	***"Personal data"** means any information relating to an identified or identifiable natural person ('data subject'). An identifiable natural person is one who can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical,*	☐	☒

	physiological, genetic, mental, economic, cultural or social identity of that natural person. Any video or audio recordings of participants is considered to be personal data.		
	If YES, provide a description of the data and explain why this data needs to be collected:		
2	This includes audio/video data of participants, but can also include IP addresses and usernames. Names, addresses and emails also count, as do consent forms. *(this box should expand as you type)*		
	Does it involve special category data (as defined by the GDPR)?	**YES**	**NO**
3	***"Special category data"*** *means sensitive personal data consisting of information as to the data subjects' –* *(a) racial or ethnic origin,* *(b) political opinions,* *(c) religious beliefs or other beliefs of a similar nature,* *(d) membership of a trade union (within the meaning of the Trade Union and Labour Relations (Consolidation) Act 1992),* *(e) physical or mental health or condition,* *(f) sexual life,* *(g) genetics,* *(h) biometric data (as used for ID purposes),*	☐	☒
	If YES, provide a description of the special category data and explain why this data needs to be collected:		
4	What counts as 'sensitive' will differ between cultures. Any information on behaviour that is not in accordance with cultural norms would count as sensitive personal data. *(this box should expand as you type)*		

	Will data from the research activity (collected data, drafts of the thesis, or materials for publication) be stored in any of the following ways?	**YES**	**NO**
5	Manual files (i.e. in paper form)?	☐	☒
6	University computers?	☒	☐
7	Private company computers?	☐	☒
8	Home or other personal computers?	☐	☒
9	Laptop computers/ CDs/ Portable disk-drives/ memory sticks?	☐	☒
10	"Cloud" storage or websites?	☐	☒
11	Other – specify:	☐	☒

12	For all stored data, explain the measures in place to ensure the security of the data collected, data confidentiality, including details of backup procedures, password protection, encryption, anonymisation and pseudonymisation:
	All data will be stored securely in the OneDrive cloud storage service provided by the University and only I will have access to it. The data will not be shared with anyone else in order to protect the privacy and security of the data. In addition, in order to protect the data from being lost, regular backups of the data will be made to ensure that the data is well preserved. To further enhance data security, all data stored on OneDrive will be encrypted using the Advanced Encryption Standard (AES) to ensure that the data is secure during transmission and storage. This means that even if data is intercepted in transit, the content cannot be deciphered without the correct key. In this way, we can maximize the protection of data from unauthorized access and potential security threats. *(this box should expand as you type)*

	Data Protection		
	Will the research activity involve any of the following activities:	**YES**	**NO**
13	Electronic transfer of data in any form?	☐	☒
14	Sharing of data with others at the University outside of the immediate research team?	☐	☒
15	Sharing of data with other organisations?	☐	☒
16	Export of data outside the UK or importing of data from outside the UK?	☐	☒
17	Use of personal addresses, postcodes, faxes, emails or telephone numbers?	☐	☒
18	Publication of data that might allow identification of individuals?	☐	☒
19	If YES to any question, please provide full details, explaining how this will be conducted in accordance with the GDPR and Data Protection Act (2018) (and/or any international equivalent):		
	This includes data such as drafts of your thesis as well as experimental or survey data. An example of suitable text is given below. *All data will be encrypted and kept in password protected cloud storage on the University Office 365 system which will not be shared. Any USB sticks used to store or*		

	transfer data will be password protected. All data transfers will be encrypted and password protected. All participants will be given a unique identifier to ensure confidentiality and this list will be kept securely in the password protected folder. The data will be stored until the completion of the project and then deleted. In accordance with the DPA2018, participants will have the right to ask to see what data is held relating to them, and this data will be deleted immediately if the participant requests this, in which case the data will not be used in the project. *(this box should expand as you type)*	
20	List all who will have access to the data generated by the research activity:	
	Typically, only the principal investigator and their supervisor will have access to data generated by research activities to ensure research confidentiality and data security. *(this box should expand as you type)*	
21	List who will have control of, and act as custodian(s) for, data generated by the research activity:	
	The Principal Investigator will act as the controller and custodian of the data during research activities and will be responsible for ensuring the security, accuracy and compliance of the data. *(this box should expand as you type)*	
22	Give details of data storage arrangements, including security measures in place to protect the data, where data will be stored, how long for, and in what form.	
	In order to ensure the security of the research data, all data will be encrypted using advanced encryption techniques and stored on the password protected University Office 365 cloud storage system and this data will not be shared with any unauthorized individuals or groups. Any USB drives used during the project, whether for storage or data transfer, will be password protected to prevent unauthorized access. Upon completion of the project, these flash drives will be reformatted to completely destroy all data stored on them. The data will be retained until the completion of the project, after which it will be securely deleted to comply with the relevant policies on data retention and destruction. *(this box should expand as you type)*	
22	Confirm that you have read the UWTSD guidance on data management (see https://www.uwtsd.ac.uk/library/research-data-management/)	☒
23	Confirm that you are aware that you need to keep all data until after your research has completed or the end of your funding	☒

SECTION I: Declaration

	The information which I have provided is correct and complete to the best of my knowledge. I have attempted to identify any risks and issues related to the research activity and acknowledge my obligations and the rights of the participants. In submitting this application I hereby confirm that I undertake to ensure that the above named research activity will meet the University's Research Ethics and Integrity Code of Practice which is published on the website: https://www.uwtsd.ac.uk/research/research-ethics/		
1	**Signature of applicant:**	Lingzhuo Tu	**Date:2024.05.10**
2	Director of Studies/Supervisor:		**Date: 2024.05.10**
3	Signature:	Lingzhuo Tu	

FOR INTERNAL USE ONLY:

	Ethical approval given		
1	**Signature of assessor:**		**Date:**
2	Name:		
3	Role:		

LOGBOOK

Date	Daily Activities	Thought Trails
2024-02-19	Initiated background reading on SDN and its routing mechanisms.	Focused on understanding the limitations of current systems.
2024-02-21	Continued literature review, focusing on challenges in dynamic routing.	Noticed significant gaps in real-time adaptability of models.
2024-02-23	Began drafting the research proposal.	Considering the integration of Dueling Deep Q-Networks.
2024-02-25	Refined the proposal, focusing on real-time traffic prediction integration.	Explored traffic prediction models for potential integration.
2024-02-27	Submitted the proposal and began setting up Mininet simulation environment.	Contemplated the setup challenges.
2024-03-01	Started first set of experiments to test basic SDN controller functionality.	Assessed initial results against theoretical expectations.
2024-03-03	Adjusted experimental parameters based on findings and reran simulations.	Reflected on the importance of fine-tuning network parameters.
2024-03-05	Conducted extensive tests on the DDQN model under stress conditions.	Observed model adaptability to sudden network load changes.
2024-03-07	Compiled results and began drafting report sections on methodology and early findings.	Considered how to present complex concepts clearly.
2024-03-09	Peer-reviewed draft chapters and incorporated feedback.	Valued peer feedback for enhancing content clarity.
2024-03-11	Peer-reviewed the draft chapters and incorporated feedback.	Recognized the value of peer feedback in clarifying and enhancing the report's content.
2024-03-13	Reviewed additional literature on adaptive routing algorithms.	Explored the relationship between network traffic variability and algorithmic responsiveness.
2024-03-15	Conducted a meeting with the advisory committee to discuss project progress and gather input.	Received valuable insights on potential scalability issues of the proposed model.
2024-03-17	Began coding the modified routing algorithms using Python and the Ryu controller.	Considered the trade-offs between complexity of code and performance efficiency.

2024-03-19	Troubleshooted issues from initial coding tests; implemented optimizations.	Reflected on the necessity of efficient debugging practices to maintain project timeline.
2024-03-21	Prepared for mid-project presentation by creating slides and rehearsing key points.	Focused on how to effectively communicate complex technical details to a non-technical audience.
2024-03-23	Delivered mid-project presentation; received feedback on project direction and methodology.	Contemplated feedback regarding the integration of additional predictive metrics in the model.
2024-03-25	Revised project plan and timeline in consultation with supervisor to incorporate new components.	Assessed the impact of changes on the overall project scope and expected outcomes.
2024-03-27	Began extensive data collection phase using both simulated and real-world data sources.	Examined the consistency and quality of incoming data to ensure its suitability for model training.
2024-03-29	Analyzed initial datasets and performed preliminary data cleansing and preparation.	Recognized patterns and anomalies in the data that could influence model training and performance.
2024-03-31	Engaged in detailed discussions with data scientists to optimize feature selection for the model.	Weighed the benefits of including diverse features against the complexity they introduce to the model.
2024-04-02 to 2024-08-27	Conducted iterative cycles of model refinement, testing, and validation. Continuously updated and revised the research manuscript.	Adapted to new findings and maintained a focus on innovation and scientific accuracy in research.
2024-08-28	Submitted the final thesis.	Reflected on the comprehensive journey from project initiation to completion.

GLOSSARY

SDN (Software Defined Networking): A networking approach that allows network behavior to be controlled by software applications using open interfaces, separating the network's control logic from the underlying physical routers and switches.

OpenFlow: A communication protocol that gives access to the forwarding plane of a network switch or router over the network.

Ryu: An open-source network controller that manages devices in an SDN environment using OpenFlow protocol.

Mininet: A network emulator that creates a virtual network on a single machine, used for developing and testing SDN applications.

Dueling Deep Q-Networks (DDQN): An advanced reinforcement learning algorithm that helps in choosing actions to maximize the long-term reward in a given state of the environment.

Network Throughput: Measures the rate of successful message delivery over a communication channel.

Latency: The delay before a transfer of data begins following an instruction for its transfer.

Packet Loss: Occurs when one or more packets of data travelling across a computer network fail to reach their destination.

Topology Discovery: The method by which network devices and their connections are identified.

Controller: In SDN, the central authority that directs traffic flows throughout the network based on a global view of the network state.