Routing and Scheduling in Future Data Center Networks

by **Zequn Jia**

Thesis submitted in fulfilment of the requirements for the degree of

Doctor of Philosophy

under the supervision of Prof. Ren Ping Liu and Dr. Ying He

School of Electrical and Data Engineering

Faculty of Engineering and IT

University of Technology Sydney

November 6, 2024

Certificate of Authorship / Originality

I, Zequn Jia, declare that this thesis is submitted in fulfilment of the requirements for the award of Doctor of Philosophy, in the School of Electrical and Data Engineering at the University of Technology Sydney.

This thesis is wholly my own work unless otherwise referenced or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

I certify that the work in this thesis has not previously been submitted for a degree nor has it been submitted as part of the requirements for a degree at any other academic institution except as fully acknowledged within the text. This thesis is the result of a Collaborative Doctoral Research Degree program with Beijing Jiaotong University.

This research is supported by the Australian Government Research Training Program.

Signature: Production Note: Signature removed prior to publication.

Date: November 6, 2024

Abstract

Data centers, essential for processing and transmitting vast data volumes, rely on efficient networks to maintain global service delivery. Software-defined networking has emerged as a revolutionary approach to enhance data center network flexibility and manageability. Despite its potential, software-defined networking faces challenges like inefficient routing, control plane bottlenecks, and inadequate fault detection and traffic management during failures, affecting network performance and stability. This thesis addresses these critical challenges in software-defined data center networks by proposing innovative solutions for routing control, fault detection, and traffic scheduling.

Firstly, it highlights the deployment of network topology regularity in designing efficient routing methods, namely cRetor and sRetor. cRetor leverages topology description language for low-overhead network setup and effective path calculation, significantly reducing the control plane's overhead. In addition, sRetor minimizes flow startup times and controller load by distributing decision-making to switches, demonstrating efficient routing control suited for large-scale networks. Simulation results validate these methods' ability to enhance network operation performance.

Furthermore, the thesis explores proactive end-to-end fault detection, introducing a heuristic algorithm and a deep reinforcement learning-based algorithm for generating fault probing matrices. These algorithms optimize the detection path, enhancing detection efficiency and accuracy. The heuristic algorithm selects detection paths progressively, introducing a new metric for quick performance assessment of fault detection matrices. The deep reinforcement learning-based algorithm improves detection accuracy through exploration and optimization, showing superior performance in experimental comparisons.

Additionally, it addresses network traffic scheduling under failure scenarios, proposing a fault-

aware traffic scheduling algorithm that combines graph neural networks with deep reinforcement learning. This approach adapts to dynamic topology changes, ensuring efficient and reliable traffic management during failures. The algorithm's integration of the graph neural network's generalization capabilities with deep reinforcement learning's decision-making enhances scheduling performance, outperforming existing methods.

In summary, this thesis systematically enhances software-defined data center networks' efficiency, reliability, and scalability by optimizing routing control, fault detection, and traffic scheduling. It provides a comprehensive solution framework for intelligent adaptive network control, offering significant contributions to the field. This research advances data center network technology and holds practical value for future development and deployment in commercial settings.

Dedication

To my parents, supervisors and friends.

Acknowledgements

As I reflect on my doctoral journey at the University of Technology Sydney, I am filled with gratitude for the many individuals who have guided, supported, and inspired me along this challenging and rewarding path.

Firstly, I must express my appreciation to my supervisors at Beijing Jiaotong University, Prof. Yantao Sun and Prof. Qiang Liu, who introduced me to the realm of scientific research. Their initial guidance was instrumental in setting the foundation for my academic pursuits.

My deepest thanks also go to my supervisors at UTS, Prof. Ren Ping Liu and Dr. Ying He, whose support and guidance have been pivotal to my research and academic growth. Professor Liu's willingness to involve me in the joint PhD program broadened my academic perspective, enriching my doctoral experience significantly. Dr. He's meticulous supervision and invaluable feedback have transformed me from a novice to a researcher capable of independent inquiry and contribution to my field. Their encouragement and unwavering support have been fundamental to my achievements.

I would also like to express my sincere thanks to the wonderful colleagues in both BJTU and UTS, whose companionship and assistance made my doctoral life colorful and fulfilling. Special thanks go to Qianqian Wu, for her advice and support during my thesis writing.

I am also grateful to my family and friends, whose love, understanding, and support have been my strongest pillars throughout this journey.

Zequn Jia

Beijing, China, 2024

Contents

List of Figures

List of Tables

List of Publications

Journal Publications

- **Zequn Jia**, Baihe Ma. "Deep Reinforcement Learning and Graph Neural Networks-based Fault-aware Flow Scheduling for Data Center Networks" IEEE Wireless Communications Magazine. (Under revision)
- **Zequn Jia**, Yantao Sun, Qiang Liu, Song Dai, and Chengxin Liu. "cRetor: An SDN-based routing scheme for data centers with regular topologies." IEEE Access 8 (2020): 116866-116880.
- **Zequn Jia**, Qiang Liu, and Yantao Sun. "sRetor: a semi-centralized regular topology routing scheme for data center networking." Journal of Cloud Computing 12, no. 1 (2023): 150.
- **Zequn Jia**, Qiang Liu, Ying He, Qianqian Wu, Ren Ping Liu, and Yantao Sun. "Efficient end-to-end failure probing matrix construction in data center networks." Journal of Communications and Networks 25, no. 4 (2023): 532-543.
- **Zequn Jia**, Yantao Sun, and Qiang Liu. "OPSDN: an enhanced SDN simulation framework for OPNET Modeler." Journal of Open Source Software 8, no. 83 (2023): 4815.
- Qianqian Wu, Qiang Liu, **Zequn Jia**, Ning Xin, and Te Chen. "P4SQA: A P4 Switch-based QoS Assurance Mechanism for SDN." IEEE Transactions on Network and Service Management (2023).

Chapter 1

Introduction

With the development of cloud computing and artificial intelligence (AI) technologies, the demand for data center networks is rapidly growing. These applications require robust, efficient, and scalable data center networks to process and transmit massive amounts of real-time data. However, the development of data center networks still needs to improve in terms of efficient routing, fault detection, and traffic scheduling.

This chapter introduces the background and challenges that data center networks face, followed by the research objectives and content. Finally, it presented the organizational structure of this thesis.

1.1 Background

In the digital and Internet-driven era, the role of data centers has become increasingly important, emerging as the core infrastructure that supports the continuous progress and development of modern society. Particularly with the rapid development of technologies including artificial intelligence [1], [2], the metaverse [3], [4], the Internet of Things (IoT) [5], [6], Large Language Models (LLM) [7], [8], and multimodal technologies [9], [10], various applications have set higher requirements for the performance, stability, and scalability of data center networks. As the critical infrastructure supporting these cutting-edge technology applications, data centers are not only places for information storage and processing but also the core driving force of an intelligent society.

Firstly, artificial intelligence and machine learning applications have penetrated various industries and daily life, from simple personal assistants to complex autonomous driving, medical diagnostics, and stock market analysis. These applications rely on powerful computing capabilities, massive information processing capacities, and high-speed stable data transmission abilities [11]. For instance, large language models like ChatGPT are currently increasing, showing great potential for application in natural language processing, content generation, and intelligent writing. The application and training of these large language models require vast datasets and extremely high computational power. It is estimated that OpenAI, the company behind the development of ChatGPT, possesses more than 3617 A100 GPU servers [12]. These servers' interconnectivity and collaborative training require strong support from data centers, providing high bandwidth, low latency, and highly reliable underlying networks [13].

Moreover, the continuous development of technologies like the metaverse and the IoT also poses higher demands on data centers and their networks [14]. The construction and development of the metaverse require massive data processing and real-time interaction, in which data centers play crucial roles in data sharing, processing, and interactive response. Numerous servers are tasked with storage, computation, access, and other functions within data centers. Data center networks enable high-speed interconnectivity among these servers, ensuring the seamless operation of virtual environments and smooth user experiences. The widespread application of IoT technologies, from smart homes to Industry 4.0, generates massive amounts of endpoints that need to efficiently exchange data with data centers, raising higher requirements for the stability and scalability of data center networks [15].

In addition to the innovative applications mentioned above, some traditional commercial applications also pose higher demands on data center networks as they scale up. Applications like e-commerce, video streaming, and online office suites are expanding their user bases, increasing the load pressure on data center networks.

The performance of data center networks directly affects the efficiency of upper-layer applications and the user experience. Different applications depend on data center networks based on their specific requirements for bandwidth, latency, reliability, and stability [16]–[18].

From the perspective of **bandwidth requirements**, on the one hand, the continuous development of traditional video streaming services, large file transfer services, and cloud storage require significant bandwidth. On the other hand, applications such as the metaverse contain a large amount of finely modeled resources, which also require sufficient bandwidth resources to support high-definition virtual reality content streams. Moreover, as the model size of AI applications like LLM and multimodal increases, the scale of datasets used for training also grows significantly, leading to an increasing demand for data center network bandwidth.

From the perspective of **latency sensitivity**, AI inference often requires real-time, low-latency responses, relying on data center networks to provide efficient data transmission and communication capabilities. Real-time virtual interactions in metaverse applications also require that data center networks provide extremely low-latency data transmission services to ensure a good user experience. In addition, applications such as online gaming, stock trading, and real-time communication are susceptible to latency. Even minimal delays can lead to a sharp decline in the user experience and even directly affect the success of the business.

Considering **reliability and stability**, reliability and stability are the most critical factors for applications such as financial services, medical information systems, and government data services. These applications require data center networks to ensure the accuracy and security of data transmission. Any data loss or error could lead to severe consequences. Furthermore, as AI and IoT technologies become widely integrated into daily life, any network failures affecting their reliability and stability can also threaten the convenience and safety of people's daily lives.

Different application scenarios place new and higher demands on the bandwidth, latency, stability, and reliability of data center networks. These metrics affect the performance of applications and user experiences. Therefore, innovation and development in data center network technologies become the essential foundations for supporting the development of these applications.

Early data center networks were primarily based on a simple three-tier architecture design, including the access layer, aggregation layer, and core layer [19], and adopted routing protocols such as Open Shortest Path First (OSPF) [20], Intermediate System to Intermediate System (IS-IS) [21], and Border Gateway Protocol (BGP) [22]. The main advantage of this structure is its relatively simple deployment and management. However, as the scale of data centers and application demands increased, this traditional architecture revealed numerous limitations that need improvement for large-scale data center networks. Problems such as the high operating overhead and slow convergence speed of these routing protocols become more prominent as the network scale increases, making it challenging to meet the needs of current data center networks. Moreover, traditional data center networks also face the following significant challenges [23]:

- **Bandwidth Bottlenecks**: The traditional architecture struggles to provide sufficient internal bandwidth to meet the demands of high-throughput applications.
- **Network Congestion**: Network congestion becomes the norm during peak traffic periods, affecting data transmission efficiency.
- **Poor Scalability**: The traditional architecture has limited scalability and is difficult to adapt to rapidly growing business needs.
- **Weak Fault Recovery**: The recovery time after a network failure occurs takes a long time, affecting the continuity of services.

To overcome these limitations, data center networks have begun to evolve towards more extensive scale, higher performance, and greater intelligence.

Scale is a significant feature of current commercial data center networks. For example, hyperscale data centers now host hundreds of thousands to millions of servers. These servers, interconnected by large-scale networks, form large pools of computing and storage resources. To meet the data interchange needs among these massive nodes, the scale of data center networks has correspondingly expanded; e.g., some well-known companies' data center networks contain over 1,000,000 network ports.

Moreover, intelligence and automation are becoming new trends in commercial data center networks. The application of new technologies such as Software-Defined Networking (SDN) and Network Functions Virtualization (NFV), along with the introduction of new network architectures, has greatly improved the performance and manageability of data center networks [24],

[25]. As a new network paradigm, software-defined networking such as OpenFlow [26] achieves the separation of the control plane and data plane in data center networks, enabling centralized orchestration and automated management of network devices. Additionally, artificial intelligence and big data analytics technologies are gradually being integrated into data center networks, enabling intelligent functionalities such as network fault prediction and intelligent traffic scheduling.

Based on the technologies and architectures mentioned above, current research in the field of data center networks primarily targets the demands of modern data centers for high performance, reliability, and scalability. The main research directions include the following aspects:

- Efficient routing algorithms and network architecture design: With the continuous expansion of data center scales and the increasing complexity of service demands, traditional data center network architectures and routing algorithms take much work to meet the requirements for efficient, low-latency communication. Therefore, designing more efficient routing algorithms and network architectures to support rapid data processing and transmission has become a current research focus.

- Fault detection and self-recovery technologies: The stability and reliability of data center networks are crucial for ensuring service continuity. Thus, effectively detecting and preventing faults within the network and implementing rapid self-recovery to minimize the impact on business operations is another key focus of current research.

- Network virtualization and resource optimization scheduling: Network virtualization technology can enhance the resource utilization rate of data center networks and enable flexible scheduling of resources. Studies on the optimization of resource allocation using network virtualization technology to ensure the QoS of different applications are a promising research direction.

- Intelligent network management: Utilizing technologies such as artificial intelligence and machine learning to achieve intelligent management of data center networks can improve operational efficiency and reduce costs. Investigation of intelligent technologies for real-time monitoring, fault prediction, and automated response is also one of the current research trends.

In the context described above, this thesis focuses on addressing critical technological challenges

in hyperscale data center networks, such as efficient routing, fault detection and awareness, and traffic scheduling. Through in-depth research on advanced routing strategies, innovative fault detection and recovery mechanisms, and efficient fault-aware traffic scheduling algorithms, this thesis aims to propose a comprehensive solution to support the efficient operation of data center networks, overcome limitations in the development of data center networks, and thereby drive data centers towards automation and intelligence. Hence, the research for this thesis holds the following significant implications:

1. **Improving the operational efficiency of data centers.** In data center networks, the transmission of information requires passing through multiple nodes. Designing efficient routing technologies based on the regularity of data center networks can further enhance the efficiency of routing computation and forwarding. An efficient fault detection mechanism can identify issues early on and carry out necessary repairs or adjustments to ensure the normal operation of data centers. Furthermore, by studying traffic scheduling algorithms, data centers can dynamically and reasonably adjust resource allocation when processing large-scale data. Therefore, by addressing critical issues such as efficient routing, fault detection, and traffic scheduling, this thesis can effectively improve the operational efficiency of data center networks, fostering the development of cloud computing and upper-layer applications.

2. **Enhancing the reliability and stability of data centers.** Fault detection mechanisms are one of the essential methods to enhance the reliability and stability of data centers. By efficiently detecting faults, data center network operators can promptly identify failures within the network and take measures to mitigate and improve them. Moreover, fault-aware traffic scheduling algorithms can achieve automatic traffic routing upon fault notification, selecting the best path to bypass faulty links. The research and advancement of these two technologies can significantly improve the reliability and stability of data centers, ensuring smooth operations even in faulty scenarios.

3. **Promoting the development of data center networks towards automation and intelligence.** Automated and intelligent data center networks will become the new trend in data center network management. Solving the core technical problems of data center networks can greatly promote the development of data center networks toward automation and intelligence. In an automated context, data centers can monitor net-

work status in real-time and promptly identify and address issues, avoiding large-scale service interruptions. In an intelligent context, by incorporating machine learning and artificial intelligence algorithms, data center networks can proactively optimize adjustments, further enhancing service quality and operational efficiency. This can improve the operational efficiency of data center networks and ensure fast and stable data processing and transmission to meet the growing user demands.

In summary, the research in this thesis can improve the operational efficiency, reliability, and stability of the data center networks, ensuring the efficient and continuous operation of services. It can also enhance operational efficiency across various industries by further promoting the development of data centers for automation and intelligence. Therefore, this research on data center networks will advance development across various fields, thereby driving the progress of the entire socio-economic landscape.

1.2 Problems and Challenges

Despite extensive research efforts to increase the efficiency and stability of data center networks, several persistent bottlenecks and emerging challenges in this domain still need to be investigated.

Firstly, in the context of Software-Defined Data Center Networks (SDDCN), the issue of control plane bottlenecks remains a pertinent challenge. Even though the centralized control plane in SDN simplifies decision-making for network routing and forwarding, enhancing flexibility and scalability of data center network management, it also necessitates that all network states be reported to the controller via the control plane. Consequently, every decision must originate from the controller and be disseminated to all switches via the control plane, substantially increasing the controller's load. This problem is not particularly evident in typical SDN networks; however, its significance becomes more pronounced due to the larger scale of data center networks. Current solutions only alleviate this issue slightly without fully resolving it. Multi-controller solutions present considerable promise but pose complex optimization problems like controller placement optimization and control plane load balancing. Accordingly, a method must be proposed to address the controller bottleneck issue, facilitating a single controller's ability to support larger-scale networks.

Secondly, in the field of fault detection in DCN, active end-to-end probing matrices still face issues with low detection accuracy and high probing overhead. Among the various methods for fault detection in data center networks, proactive approaches offer notable advantages: they allow for independent selection of detection path combinations and higher real-time detection rates. Thus, proactive or hybrid detection methods have become commonplace in data center networks. Nevertheless, the algorithm constructing the fault detection matrix must efficaciously select an optimal set of detection paths, enabling the fault localization system to achieve a given detection accuracy with minimized detection path overheads. In theory, a greater number of fault detection paths results in more probing results and enhances the ability of the fault localization algorithm to locate faults. However, proactive detection can negatively impact network performance, necessitating a balance between detection overhead and accuracy. Classical methods like PingMesh[27] do not delve into this issue, and although some, like deTector[28], begin to consider fault detection matrix generation, there is still room for further optimization.

Finally, in the field of traffic scheduling, existing methods still face issues with degraded traffic scheduling performance in scenarios involving failures. Given the inevitability of faults in data center networks, the issue of devising effective traffic scheduling strategies under such conditions remains unresolved. While there is extensive research on traffic scheduling to enhance the user experience in data center networks, the specific challenge of traffic scheduling in the face of network faults has received comparatively less attention. Many advanced traffic scheduling strategies employ deep reinforcement learning (DRL) techniques. However, these conventional deep learning methodologies often fall short regarding generalization capabilities. This deficiency becomes particularly evident when a network fault modifies the network topology. If the DRL algorithm fails to adapt to these changes, it may not produce satisfactory scheduling results. Therefore, developing a traffic scheduling strategy with superior generalization abilities, capable of maintaining efficient and stable scheduling amidst recurrent topology changes induced by faults, constitutes a significant area for future investigation.

1.3 Research Objectives and Overview

In addressing the problems mentioned above and challenges, this thesis primarily delves into three crucial issues in data center networks: the routing and overhead complications in software-defined data center networks, the balancing between efficiency and precision in fault detection,

and the problems of fault-aware traffic scheduling.

As shown in Figure 1.1, this thesis's primary research components, which correspond to the problems above, encompass the following three sections:

Firstly, employing the rule-based data center network topology description language (TPDL) as a foundation, we design fundamental routing methods for software-defined data center networks, namely cRetor and sRetor. Secondly, we undertake an analysis and modeling of the end-to-end fault detection mechanism within data center networks, proposing a heuristic probing matrix construction algorithm. The study also considers deep reinforcement learning applications to this issue, devising a DRL-based probing matrix construction method. Finally, regarding the traffic scheduling problem in data center networks, we integrate graph neural networks with deep reinforcement learning to propose a fault-aware data center network traffic scheduling algorithm to resolve the traffic scheduling problem under fault conditions.

This thesis's primary focus is optimizing routing and traffic scheduling within data center networks, aiming to enhance their performance and reliability in response to increasing network traffic and service demands. The investigations throughout this thesis present a comprehensive solution and framework for designing and managing data center networks, thereby providing robust support for their development and application.

The research presented in this thesis has significant potential impacts and applications across various domains:

1. Enhanced Cloud Computing Performance: The proposed routing methods (cRetor and sRetor) can significantly improve the efficiency of large-scale cloud computing infrastructures. By reducing control plane overhead and optimizing routing decisions, these methods can lead to faster response times and increased throughput for cloud-based applications, benefiting sectors such as e-commerce, online gaming, and streaming services.

2. Improved Reliability for Critical Services: The fault detection matrix generation algorithms can enhance the reliability of data center networks supporting critical services. This is particularly crucial for sectors like healthcare (e.g., telemedicine), finance (e.g., high-frequency trading), and government services, where network downtime can have severe consequences.

3. Energy Efficiency in Data Centers: By optimizing routing and traffic scheduling, the proposed methods can contribute to reducing energy consumption in data centers. This aligns

with global efforts to create more sustainable IT infrastructures and can lead to significant cost savings for data center operators.

4. Enhanced AI and Machine Learning Infrastructure: The improved network performance and reliability resulting from this research can provide a more robust infrastructure for AI and machine learning applications. This can accelerate advancements in fields such as natural language processing, computer vision, and large-scale data analytics.

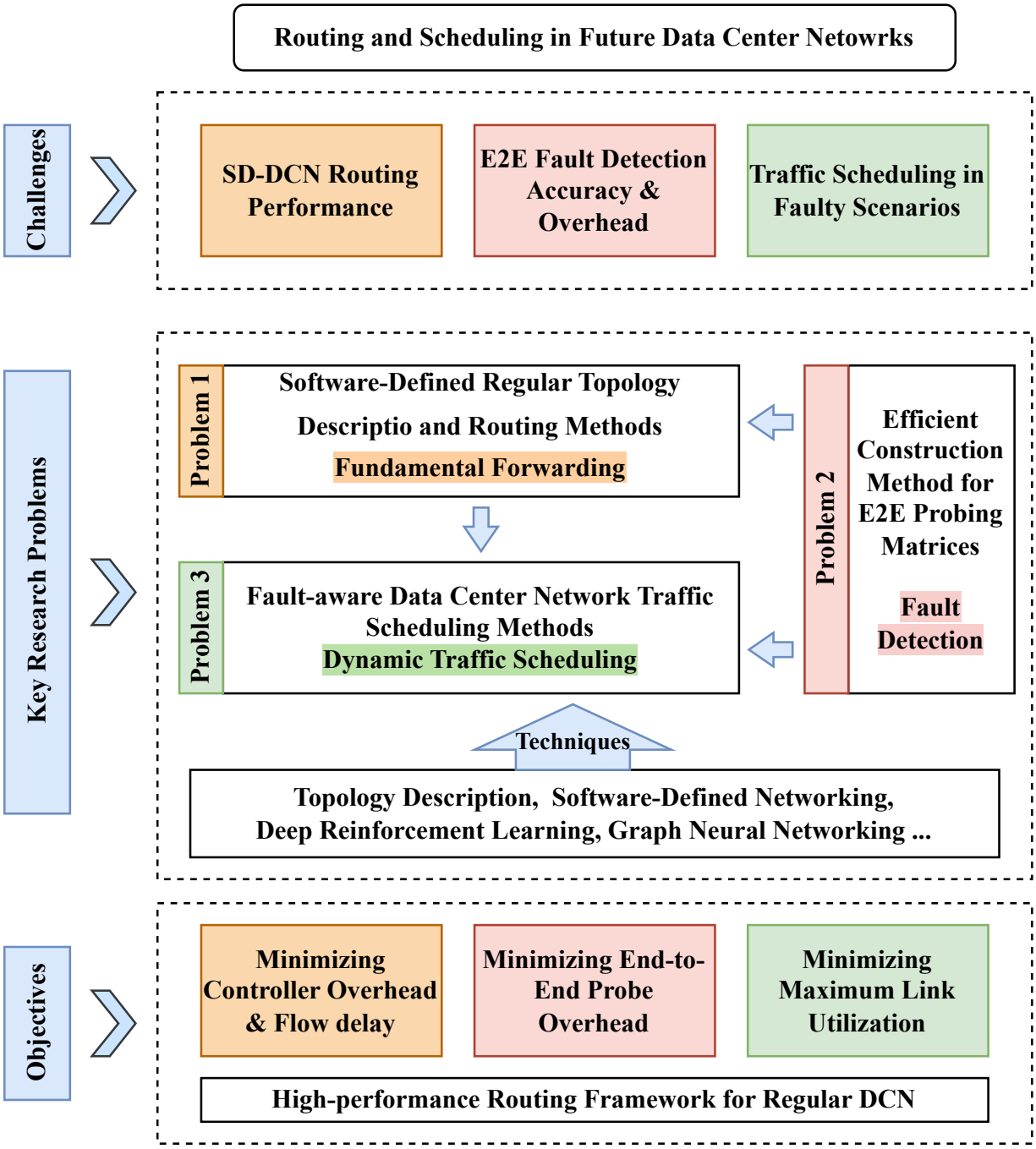


Figure 1.1: The research content and framework of this thesis.

Specifically, this thesis focuses on three main research problems, as follows:

1. Centralized and semi-centralized regular software-defined data center network routing methods based on topology description language

This section's scope of research addresses the challenges associated with managing routing overhead in large-scale data center networks. Due to the massive scale of data center networks, traditional network routing methods make it challenging to meet the needs for efficient routing and forwarding. Therefore, this thesis introduces an SDN-based routing scheme supported by the TPDL. This approach aids in efficient routing and forwarding within data center networks, drastically lessens the load on the controller, and thereby makes SDN technology viable in large-scale data center networks.

2. Heuristic and deep reinforcement learning-based probing matrix construction algorithms in end-to-end fault detection

The fault detection mechanism in data center networks is a vital component to ensure efficient and stable operation. The research goal of this part is to provide precise and rapid fault detection capabilities to achieve efficiency and stability. Specifically, this thesis establishes a mathematical model and corresponding optimization problem for end-to-end fault detection. Based on this model, this thesis first proposes a heuristic generation algorithm for the fault detection matrix to generate a low-overhead matrix. Furthermore, this thesis uses deep reinforcement learning methods to solve the combinatorial optimization problem, obtaining a more optimized fault detection matrix generation algorithm.

3. Fault-aware data center network traffic scheduling algorithm based on graph neural networks and deep reinforcement learning

This section aims to optimize traffic distribution and scheduling in data center networks under fault conditions. Specifically, this thesis proposes a fault-aware data center network traffic scheduling algorithm based on GNN and DRL. This algorithm leverages the message-passing capabilities of GNN to generalize graph structures, enabling message propagation and aggregation across variable fault topologies. As a result, the algorithm can intelligently allocate and dynamically adjust network traffic, even for faults that have never occurred in the network before. This approach aims to maximize flow completion rates while reducing packet loss rates in the network.

In this thesis, the three central research points - routing methods, fault detection, and traffic scheduling - are interconnected and cooperative, forming a comprehensive, adaptive solution for managing data center networks. The routing methods, cRetor and sRetor, serve as the foundation of the network infrastructure. These methods provide basic connectivity and efficient routing capabilities, offering multiple potential paths for traffic forwarding. They establish a low-overhead, high-efficiency routing framework that supports the subsequent layers of fault detection and traffic scheduling.

Building upon this foundation, the fault detection matrix generation layer enhances the routing methods by enabling fault discovery and recovery capabilities. It provides critical real-time fault information to both the routing methods and the traffic scheduling algorithm, ensuring network reliability and adaptability in the face of anomalies.

At the top layer, the fault-aware traffic scheduling algorithm leverages both the routing methods and fault detection to optimize network performance. It dynamically selects optimal paths from those provided by the routing methods, utilizing fault information from the detection layer to make informed scheduling decisions. This allows the system to respond in real-time to network changes, optimizing traffic distribution across the network.

The interdependencies among these components are crucial to the system's effectiveness. The routing methods provide the fundamental network structure and multiple path options. The fault detection layer continuously monitors this structure, identifying any issues that arise. The traffic scheduling algorithm then uses both the routing options and fault information to make optimal traffic management decisions.

This multi-layered approach ensures that the network has a solid, efficient routing foundation through cRetor and sRetor. Faults are quickly identified through the fault detection matrices, and traffic is intelligently managed based on both routing options and network health through fault-aware scheduling. Together, these three components create a robust, adaptive, and efficient system for managing large-scale data center networks, each layer building upon and enhancing the capabilities of the others.

The key contributions of this thesis are as follows:

- Design of routing methods based on the regularity of data center network topology: This thesis introduces two software-defined data center network routing methods, cRetor

and sRetor. These methods leverage the regularity of data center network topology to optimize the routing control process, reducing the load on controllers and enhancing routing efficiency.

- In-depth study of end-to-end fault detection and an innovative algorithm for generating probing matrices: This thesis conducts a theoretical modeling analysis of proactive end-to-end fault detection methods. It proposes heuristic and deep reinforcement learning-based approaches for generating optimized fault detection matrices, effectively balancing fault detection accuracy with overhead.

- Propose a fault-aware traffic scheduling algorithm based on GNN and DRL: This thesis integrates graph neural networks into deep reinforcement learning by modeling and analyzing traffic scheduling issues in fault scenarios. This integration improves the generalization ability of complex network structural changes, leading to more effective traffic scheduling.

1.4 Thesis Organization

This thesis is organized as follows:

Chapter 1, the Introduction, outlines the significance of efficient and stable operations in data centers. It highlights existing challenges and introduces the core research content of this thesis.

Chapter 2 comprehensively analyzes existing literature pertinent to this research. Specifically, it offers an in-depth review of literature related to regular topology routing, fault detection methods, and traffic scheduling techniques in data center environments. This chapter aims to establish a solid foundation for the thesis by critically analyzing previous research and identifying gaps that this study addresses.

Chapter 3 delves into the theoretical foundations and methodologies relevant to this study, including topology description methods, software-defined networking, deep reinforcement learning, and graph neural networks.

Chapter 4 discusses routing methods for regular topology in data center networks. It proposes two routing methods—centralized and semi-centralized—tailored for regular topology data centers, addressing routing efficiency and control plane overhead. This chapter concludes

with experimental validation of these methods, demonstrating their potential to significantly enhance the performance of cloud computing platforms and large-scale web services.

Chapter 5 introduces a novel fault detection matrix generation method for data center networks. It begins with a system model and theoretical analysis for end-to-end fault detection. It is followed by a heuristic fault detection matrix generation algorithm and an advanced approach integrating deep reinforcement learning. This chapter also includes experimental results and discussions, highlighting the method's potential to improve network reliability in critical infrastructure scenarios such as financial trading systems and healthcare networks.

Chapter 6 focuses on a fault-aware traffic scheduling method using graph neural networks and deep reinforcement learning. It defines the problem, establishes a data model, and designs a reinforcement learning method employing graph neural networks and deep reinforcement learning. Experimental results and analyses are provided to validate the approach, demonstrating its potential applications in managing complex, dynamic network environments.

The thesis concludes with Chapter 7, which summarizes the research findings and suggests directions for future work.

Chapter 2

Literature Review

This chapter reviews data center networks, including their regular topologies, routing methods, fault detection techniques, and traffic scheduling strategies. Section 2.1 elaborates on the development and implementation of regular network topologies and their routing methods. Section 2.2 investigates various fault detection methods, emphasizing the importance of stability and reliability in network operations. Section 2.3 discusses the evolution of traffic scheduling methods, particularly within software-defined networking, to address congestion and improve service quality.

2.1 Regular Topologies and Routing Methods in Data Center Networks

2.1.1 Regular Network Topologies

To enhance the forwarding performance of data center networks, researchers have proposed numerous well-known data center network topologies, exemplified by Fat-tree[29] and BCube[30]. These topologies have been gradually adopted and deployed in modern data centers. Most of these innovative data center network architectures are defined and described through recursive and iterative methods, resulting in regular network topologies. This implies that their connection relationships and addressing schemes usually follow fixed patterns[35]. To achieve higher forwarding efficiency and performance, researchers have also designed routing algorithms for these topologies, i.e., topology-aware routing methods. Topology-aware routing methods combine with specific topological structures, utilizing the construction rules of network topologies to achieve more effective routing.

Al-Fares et al.[29] constructed the classic large-scale Fat-tree network topology using ordinary commercial switches and designed a corresponding addressing method for it. This method assigns IP addresses to nodes based on their type, location, and other relevant attributes, allowing network operators and routing algorithms to determine the node's position in the topology through its IP address. They also proposed a novel two-level topology-aware routing method that can make forwarding decisions based on the node's IP address and connection relationships instead of conducting complex network state exchange processes. This algorithm uses suffix matching at the edge and aggregation switches, forwarding packets to different uplink interfaces based on the destination node's IP address. This method fully exploits the multipath characteristics of the Fat-Tree network structure, achieving good load balancing.

BCube[30] represents another data center network architecture, i.e., the server-centric architecture. In this kind of architecture, the forwarding and decision-making processes are performed on the server nodes instead of the switches. The BCube topology can be defined recursively and can generate various scales of network topologies by specifying the number of layers. The authors have also designed a corresponding topology-aware source routing method for BCube, BCube Source Routing (BSR). BSR leverages BCube's topology structure and multipath capability to achieve load balancing and fault tolerance without the need for link state

Table 2.1: Summary of Regular Network Topologies in Data Center Networks

Topology	**Architecture Type**	**Design Principle**	**Advantages**
Fat-Tree [29]	Switch-centric	Multipath routing	Load balancing, Scalability
BCube [30]	Server-centric	Recursive construction	High fault tolerance, Flexible scaling
BCDC [31]	Server-centric	Crossed Cube-based	Supports massive server numbers, Efficient routing
LaScaDa [32]	Switch-centric	Low-connectivity clusters	Scalability, High bisection bandwidth
Criso [33]	Hierarchical	Isomorphic pods	Cost efficiency, High capacity
RibsNet [34]	Dual-centric	Symmetric network	Fault tolerance, Improved network performance

synchronization and distribution.

In addition to the classic data center network architectures like Fat-tree[29], BCube[30], DCell[36], and VL2[37], numerous new regular data center network topologies[31]–[34], [38]–[40] continue to be proposed, expanding the architectural options for building efficient and scalable data centers.

BCDC[31] represents a high-performance, server-centric data center network topology. It is constructed based on the crossed cube, a type of bijective connection network. A n-dimensional BCDC network (B_n) can be recursively defined and supports a larger number of network nodes compared to the traditional Fat-tree. With 16-port commercial switches, a Fat-tree can support up to 1024 servers, whereas BCDC can accommodate up to 524,288 servers. The authors also proposed an efficient topology-aware routing method for supporting one-to-one, one-to-many, and many-to-many traffic patterns within BCDC networks.

The evolution from Fat-Tree to BCDC represents a significant architectural shift in handling scale. While Fat-Tree's design principle focuses on bandwidth optimization through multiple equal-cost paths, BCDC's crossed cube approach prioritizes topological efficiency, achieving its superior node scaling through increased architectural complexity. This trade-off between simplicity and scale presents a fundamental challenge in DCN design.

LaScaDa[32] employs switches with fewer ports to connect nodes within clusters of low connectivity and then interconnects these clusters according to a specific pattern. As a result, LaScaDa exhibits excellent performance in terms of scalability, average path length, and bisection bandwidth. Similarly, its authors have developed a custom hierarchical routing method to support efficient data forwarding within the LaScaDa topology.

Criso[33] proposes a hierarchical and recursive architecture employing two-port servers and commodity switches, forming a network through isomorphic pods. This design not only promises cost efficiency and high network capacity but also achieves superior performance in scalability, power consumption, and fault tolerance when compared to established topologies. Criso's analytical and experimental results affirm its potential to meet the fault-tolerant demands of modern data centers while maintaining balanced throughput and latency.

To handle the increase in data traffic, RibsNet[34] presents a two-layer DCN architecture. It highlights a symmetric, dual-centric network that enhances fault tolerance and network

performance by supporting comprehensive connections. The architecture's design allows for the gradual integration of servers, maintaining topological properties with a stable and low network diameter. RibsNet's theoretical and simulation performance show significant improvements in latency, throughput, cost, and power efficiency, making it a promising DCN architecture.

While these topologies represent different architectural approaches, their comparative advantages emerge in specific deployment scenarios. For instance, Fat-Tree's multipath routing capability makes it particularly effective for large-scale deployments requiring consistent latency, while BCube's recursive construction offers superior flexibility for incremental scaling. BCDC's crossed cube-based design, while supporting massive server numbers, introduces additional complexity in implementation compared to Fat-Tree's more straightforward hierarchical structure. These advancements reflect the ongoing innovation in data center network design, aiming to improve performance, scalability, and efficiency. The development of corresponding routing strategies that leverage the unique characteristics of these topologies is crucial for realizing their full potential in practical deployments.

The various data center network topologies proposed in the aforementioned studies exhibit regularity and predictability, embodying innovative approaches to constructing efficient and scalable data centers. For each specific topology, researchers have designed corresponding addressing methods and topology-aware routing algorithms to facilitate more efficient data forwarding and load balancing. These developments in BCDC, LaScaDa, and other new topologies, along with their bespoke routing strategies, highlight the pivotal role of intelligent design in data center network architecture. By leveraging the unique characteristics of these topologies and implementing efficient routing mechanisms, it is possible to enhance the performance and reliability of data center networks without incurring additional costs. This body of work underscores the significance of regular network topology and routing policy design in optimizing data center operations, offering pathways to advance the frontier of data center network engineering.

2.1.2 Routing Methods for Regular Topologies

In addition to proposing new topological structures, researchers have also leveraged the regularity of existing network topologies to propose novel topology-aware routing methods aimed at improving routing performance within data center networks. These methods often build upon famous network topologies like Fat-Tree, exploiting their unique topological characteristics to

achieve enhanced performance or reliability.

Liu et al.[41] proposed an innovative routing approach named port forwarding load-balanced scheduling algorithm. This method capitalizes on the unique addressing scheme within the Fat-Tree topology, designing a port-based source routing scheme to reduce the complexity of switches. By utilizing the inherent structure of Fat-Tree topologies, the proposed method facilitates efficient data transmission while minimizing the operational overhead on networking devices.

Nepolo et al.[42] leveraged the abundant equivalent path characteristics of the Fat-tree topology to propose a predictive equal-cost multi-path protocol specifically designed for data center networks based on Fat-tree topologies. This protocol makes forwarding decisions based on predicted congestion degree, improving the load balancing in ECMP. The protocol aims a more equitable distribution of traffic across different routes by forecasting congestion states, leading to improved network reliability and effectiveness.

Hu et al.[43] propose an innovative approach to intra-datacenter network management through an SDN scheme tailored for the Fat-tree topology. This method addresses the pressing challenges of cost, maintenance, and quality of service in modern datacenters. By integrating the Fat-Tree architecture with the principles of max-min fairness, the authors aim to refine the allocation of network resources. This approach not only promises enhanced traffic distribution efficiency but also represents a harmonious fusion of traditional network architectures with SDN technology. Their experimental results reveal the potential of this well-crafted Fat-Tree network to elevate traffic management efficacy in datacenters.

In addition, there has been a continuous emergence of customized routing optimization efforts specifically tailored for Fat-tree topologies[47], [48]. These studies focus on enhancing various aspects of network performance, such as efficiency, reliability, and load balancing, by leveraging the inherent features of Fat-Tree structures and introducing innovative routing algorithms and protocols to optimize data flow within these networks.

Similarly, the BCube topology has also inspired numerous customized routing optimization schemes[44]–[46].

Lin et al.[44] have devised fault-tolerant routing methods by constructing completely independent spanning trees within the BCube topology, effectively reducing the average path length

Table 2.2: Summary of Routing Methods for Regular Topologies

Method	**Topology**	**Key Features**	**Improvement Focus**
Liu et al. [41]	Fat-Tree	Load-balanced, Reduced complexity	Efficiency, Simplicity
Nepolo et al. [42]	Fat-Tree	Predictive, Improved load balancing	Load Balancing, Congestion Management
Hu et al. [43]	Fat-Tree	Cost-effective, Quality of service	Cost, Quality of Service
Lin et al. [44]	BCube	Reduced path length, Improved reliability	Reliability, Path Efficiency
Fan et al. [45]	BCube	Enhanced fault tolerance, Virtual links	Fault Tolerance, Network Diameter Reduction
Lv et al. [46]	BCube	Reliability assessment, Simplified approximation	Network Reliability, Fault Diagnosis

and transmission failure rate.

Fan et al.[45] proposed a multi-path routing algorithm that significantly bolsters fault tolerance in BCube topologies. An adaptive path-finding algorithm is introduced, aimed at establishing virtual links between nodes, thus reducing the BCube's diameter.

Lv et al.[46] introduce a novel approach for fault diagnosis in data center networks by focusing on the BCube network topology. It presents the first study on subsystem-based reliability, offering a method to compute and approximate the reliability of BCube. Through numerical simulations, the findings reveal that the approximation effectively reflects the actual reliability, suggesting a simplified yet accurate way to assess network reliability.

The exploration of various regular data center network topologies and the custom routing algorithms designed for each topology's unique characteristics indicate a critical aspect of current research in data center networks. These specialized routing algorithms, tailored to optimize performance for specific topological structures like Fat-tree, BCube, and others, demonstrate the potential for significant enhancements in network efficiency, reliability, and load balancing. However, a notable limitation of these algorithms is their lack of universality; they are optimized for specific topologies and may not be directly applicable to others. This specialization presents challenges as data center network architectures continue to evolve and become more heterogeneous, incorporating multiple types of topologies within a single data center infrastructure. This field still exhibits a gap in the development of universal topology-aware routing methods for regular data center networks that can adapt across different architectures.

In summary, while the advancements in customized routing strategies for specific network topologies have led to notable improvements in data center network performance, the diversity of network architectures and the dynamic nature of data center evolution call for more universal, adaptable routing solutions. Addressing this need will be crucial for ensuring that data center networks can continue to meet the demands of modern computing environments, balancing the requirements for performance optimization with the flexibility to adapt to a range of network topologies and configurations.

2.2 Fault Detection Methods in Data Center Networks

The ability to detect faults within data center networks is a critical capability for ensuring the stability and reliability of these networks. Consequently, researchers have conducted extensive research in the field of fault detection and localization within data center networks, proposing numerous methods tailored to various applications[49]. These methods have evolved significantly over the past decade, driven by the increasing complexity of data center architectures and the growing demands for real-time fault detection and resolution. Fault detection methods employed in data center networks can be categorized into three types: passive, active, and hybrid approaches.

- Passive Fault Detection: This approach relies on monitoring network traffic and analyzing patterns to identify anomalies that may indicate a fault. It does not introduce additional traffic into the network, thereby minimizing the impact on network performance. These methods have evolved significantly over the past decade, driven by the increasing complexity of data center architectures and the growing demands for real-time fault detection and resolution.

- Active Fault Detection: Active methods involve sending specially crafted probe packets through the network to test the path or component's functionality. These methods typically follow specific probing strategies, such as end-to-end probing, hop-by-hop probing, or path-based probing, each with its own trade-offs between detection accuracy and network overhead. By analyzing the response from these probe packets, the system can identify potential issues or failures within the network.

- Hybrid Fault Detection: Hybrid methods combine both passive and active approaches to leverage the advantages of each. The key challenge in hybrid methods lies in determining the optimal balance between passive monitoring and active probing, often requiring sophisticated scheduling algorithms and adaptive mechanisms to maintain efficiency. For instance, a system might primarily use passive monitoring for efficiency and supplement it with active probing when anomalies are detected or when more detailed diagnostics are necessary.

Table 2.3: Overview of Fault Detection Methods in Data Center Networks

Type	Method	Key Contribution	Methodology
Passive	Roy et al.[50], [51]	Designed a passive end-to-end method for accelerating fault detection and localization	Statistical analysis of traffic metrics
	Cociglio et al.[52]	Proposed a non-intrusive monitoring method for accurate packet loss measurement	Multi-point flow monitoring
	Xu et al.[53]	Anomaly detection by passive collection of link state databases	Comparison of routing selection information database with collected data
Active	Pandey et al.[54]	Introduced a network multi-agent system diffusion protocol for resource information propagation	Network structure and node priority-based protocol
	Guo et al.[27]	Developed PingMesh for latency measurement between servers	End-to-end probing agents on edge servers
	Popescu et al.[55]	Proposed PTPmesh for network performance statistics in cloud networks	Precision Time Protocol (PTP)
	Peng et al.[28]	Topology-aware active fault detection with optimized probing paths	Fault detection matrix generation algorithm
	Hao et al.[56]	Introduced an innovative method to enhance fault localization precision using segmentation entropy for probe set selection	Segmentation entropy for optimal probe set selection without a predefined candidate set

Continued

Type	Method	Key Contribution	Methodology
Active	Xie et al.[57]	Introduced a block matrix completion algorithm for link failure detection	Block matrix completion
	Lin et al.[58]	Proposed NetView, a network telemetry framework with efficient probing	Measurement server with on-demand probes
Hybrid	Jia et al.[59]	INT-based detection and localization of gray failures	Programmable switch supported INT
	Xie et al.[60]	Flexible telemetry mechanism leveraging INT for runtime dynamic telemetry tasks setting	INT with runtime configuration
	Su[61]	Gray failure detection and localization in Fat-tree networks using programmable switch technology	Non-overlapping balanced path generation algorithm

2.2.1 Passive Fault Detection Methods

The evolution of passive fault detection methods reflects the growing sophistication of network monitoring technologies and data analysis techniques. Traditional passive fault detection methods often rely on protocols such as SNMP[62] and NetFlow[63] for network data collection and monitoring. These protocols enable sampling and pushing of network traffic data, but they offer relatively coarse precision and sample proportionally, making it difficult to accurately reflect the actual state of the network. The limitations of these traditional methods become particularly apparent in high-speed networks where sampling rates must be carefully balanced against processing capabilities and storage constraints.

In response, researchers have begun to explore more advanced and accurate passive detection methods. For instance, Roy et al.[50], [51] designed a passive end-to-end method in Facebook's front-end data centers to accelerate the detection and localization of faults within parts of the data center network. This method correlates the transport layer traffic metrics and network I/O system call delays of end hosts with the traffic paths through the data center. It determines broken links and switches by employing methods of statistical analysis.

Cociglio et al.[52] proposed a non-intrusive monitoring method to accurately measure packet losses affecting specific flows in different network segments. This method is effective not only for monitoring end-to-end flows but also for multi-point flows that transmit packets along multiple paths within the network.

The method proposed by Xu et al.[53] represents a significant advancement in passive detection by introducing a novel approach to anomaly detection through link state database analysis. Their work demonstrates how careful analysis of routing information can reveal network anomalies without introducing additional traffic. The significance of their contribution lies in the method's ability to detect subtle routing anomalies that might be missed by traditional traffic analysis approaches, though it requires careful consideration of the trade-offs between detection accuracy and computational overhead.

However, these methods still rely on monitoring and analyzing the data generated during normal network operations. Therefore, passive fault detection methods have several non-negligible drawbacks: 1) The collected data may be of low quality, containing errors and noise; 2) There is a lack of real-time detection capability, which may delay the discovery and repair of network faults; 3) There are potential security risks, as traffic data may contain sensitive information,

raising security concerns.

2.2.2 Active Fault Detection Methods

Active fault detection methods have evolved from simple ping-based approaches to sophisticated probing strategies that consider network topology, traffic patterns, and failure characteristics. These methods involve sending specific probe packets into the network to assess its current state. These methods often provide more real-time, accurate, and targeted feedback, making them more efficient for fault localization.

Pandey et al.[54] proposed a novel network multi-agent system diffusion protocol based on network structure and node priority, where nodes interact with neighboring nodes to propagate weighted differential information about available resources. However, this solution still relies on point-to-point probing, thereby increasing the overhead of changing network devices.

PingMesh[27], proposed by Microsoft, is a quintessential active fault detection scheme. It measures latency between servers by deploying end-to-end probing agents on edge servers. Similarly, the PTPmesh[55] network probing tool uses the Precision Time Protocol (PTP) to estimate one-way delays and quantify packet loss, making it a viable tool for obtaining network performance statistics in cloud tenant networks. Like PingMesh, PTPmesh focuses on designing suitable probing methods without considering the selection of an appropriate set of fault detection paths to improve the final accuracy of fault localization.

deTector[28] is a topology-aware active fault detection system for data centers that begins to address optimizing fault localization accuracy by designing a reasonable set of fault detection paths. The authors recognized the significant impact of choosing different probing paths on the detection results and thus designed a fault detection matrix generation algorithm to select suitable detection matrices. This scheme is capable of constructing appropriate fault detection matrices, achieving higher fault localization accuracy.

The work by Hao et al.[56] represents a particularly significant advancement in probe selection methodology. Their innovative use of segmentation entropy not only improves localization precision but also addresses one of the fundamental challenges in active fault detection: determining the optimal set of probing paths without relying on predefined candidates. This approach demonstrates how theoretical advances in information theory can be practically applied to network fault detection, though the computational complexity of their method needs

careful consideration in large-scale deployments.

Xie et al.[57] introduced a new block matrix completion algorithm for detecting link failures. Compared to existing matrix completion algorithms, the proposed block algorithm reduces sampling complexity.

NetView[58] represents a novel network telemetry framework for data center networks. It requires only an additional measurement server to actively send dedicated probes on-demand to detect any device in the network. The proposed probing generation and update algorithms reduce the number of required probes, resulting in lower bandwidth usage and better scalability.

Wang et al.[64] introduce the concept of real-time fault detection as a service, proposing active fault detection in a service-oriented manner. This approach suggests an active method based on a fault detection matrix for probing, which offers certain advantages in terms of detection time, additional bandwidth occupation, and load on the end servers.

Compared to passive methods, active methods introduce additional load on the network. Uncontrolled sending of large volumes of probe packets can interfere with the normal operation of the network, thereby degrading network performance. Hence, efficient fault detection path selection algorithms are crucial. Optimizing active methods to be smarter and more resource-efficient can not only improve detection coverage and accuracy but also reduce unnecessary packet transmission, minimizing the impact on network performance.

2.2.3 Hybrid Fault Detection Methods

The emergence of hybrid methods marks a significant shift in fault detection strategy, particularly with the adoption of In-band Network Telemetry (INT) technology. These methods represent a sophisticated attempt to balance the trade-offs inherent in both passive and active approaches, while leveraging advanced programmable networking capabilities. Such methods[59]–[61], [65], [66] not only reduce the overhead of fault detection in networks but also proactively choose detection paths and frequencies, similar to active methods.

Jia et al.[59] proposed a rapid detection and localization mechanism for gray failures based on INT. This mechanism employs programmable switch technology to perform network-wide telemetry, detecting intermittent failures or those under specific conditions. Unavailable paths are identified and eliminated from each server's path information table when a network failure

occurs.

FINT[60] leverages INT to design a flexible telemetry mechanism that supports the dynamic setting of telemetry tasks and parameters at runtime without the need for redeployment, hence decreasing the influence on network performance and increasing monitoring efficiency.

Su[61] also utilized programmable switch technology to design a gray failure detection and localization technique for Fat-tree networks. This solution combines the characteristics of the Fat-Tree topology to design a non-overlapping balanced path generation algorithm aimed at network fault detection.

Hybrid fault detection methods represent a promising approach to fault detection in data centers. By integrating passive observation of network behavior with the proactive probing of active methods, these hybrid strategies aim to optimize fault detection while minimizing their footprint on network operations. The use of INT technology, in particular, offers a balance between the thoroughness of detection and the necessity of maintaining network performance.

In summary, despite the advancements in fault detection and localization methods within data center networks, there remains a gap in theoretical analysis concerning the accuracy of fault detection. Moreover, the challenge of selecting optimal detection paths to construct more efficient and cost-effective topology detection matrices in active and hybrid fault detection methods persists as a worthwhile avenue for further research.

2.3 Traffic Scheduling Methods in Data Center Network

With the rapid development of the Internet and the increasing complexity of network architectures, ensuring both efficient traffic management and reliable network operation has become increasingly critical. Modern data centers process petabytes of data daily, with traffic patterns becoming increasingly dynamic and unpredictable. This complexity is further compounded by the heterogeneous nature of network flows, ranging from latency-sensitive small flows to bandwidth-intensive large flows, as well as the constant threat of network failures and disruptions.

An effective traffic management strategy in data center networks must address two fundamental challenges: optimal traffic scheduling under normal operations and rapid recovery from network failures. The traffic scheduling aspect focuses on efficiently utilizing network resources and

maintaining quality of service, while failure handling mechanisms ensure network reliability and service continuity during link or node failures. These dual requirements have led to the development of various complementary approaches in modern data center networks.

Recent advances in fast rerouting mechanisms have demonstrated promising results in addressing network failures while maintaining performance. For instance, Bankhamer et al. [67] proposed a randomized approach to local fast rerouting that achieves both high resilience and low congestion, particularly effective in highly connected networks. Their work showed that randomization can effectively handle multiple link failures while maintaining balanced network load distribution. Building on this foundation, Verdi and Luz proposed InFaRR[68], a significant advancement in programmable data plane fast rerouting. Implemented in P4, this solution provides essential features such as loop prevention and efficient failure bypassing without requiring additional headers or network state heartbeats. Furthermore, Foerster et al.[69] introduced an innovative postprocessing framework for improving Fast Rerouting (FRR) mechanisms, enhancing existing arborescence-based network decompositions through iterative arc swapping strategies to improve route quality while maintaining resilience against multiple failures.

While these fast rerouting mechanisms provide crucial failure recovery capabilities, they work in concert with broader traffic scheduling strategies to ensure optimal network performance. In recent years, software-defined networking has emerged as a key enabler for more sophisticated traffic management, separating network control from data forwarding to enable more flexible and precise handling of complex tasks. This separation has led to significant advances in traffic scheduling methods, particularly in addressing network congestion and uneven link load distribution in data center networks.

Based on the algorithms they employ, we can categorize traffic scheduling methods into two main approaches: *heuristic traffic scheduling algorithms* and *reinforcement learning-based traffic scheduling algorithms*. These approaches, combined with modern fast rerouting mechanisms, form a comprehensive framework for managing traffic in modern data center networks, ensuring both efficient resource utilization and robust failure recovery.

2.3.1 Heuristic Traffic Scheduling Methods

In recent years, the academic and industrial communities have achieved a series of results in the study of heuristic traffic scheduling algorithms. These methods primarily rely on human expe-

Table 2.4: Summary of Traffic Scheduling Methods in Data Center Networks

Method	**Type**	**Main Focus**	**Key Outcomes**
Reddy et al.[70]	Heuristic	Energy efficiency	Reduced energy costs
Guo et al.[71]	Heuristic	Scheduling efficiency	Reduced power consumption
BiTE[72]	Heuristic	Load balancing	Energy efficiency and load distribution
Kamboj et al.[73]	Heuristic	Energy consumption	Improved delay, throughput, energy savings
Dai et al.[74]	Heuristic	Network load balancing	Enhanced bandwidth utilization
RSIR[75]	RL-Based	Routing efficiency	Improved packet loss, delay
CFR-RL[76]	RL-Based	Traffic rerouting	Balanced link utilization
Jin et al.[77]	RL-Based	Congestion control	Maximized link utilization
SmartFCT[78]	DRL-Based	Energy efficiency	Enhanced energy efficiency
DRSIR[79]	DRL-Based	Intelligent routing	Efficient routing paths
DRL-PLink[80]	DRL-Based	Hybrid flow scheduling	Reduced system overhead
EfficientTE[81]	DRL-Based	Dynamic traffic routing	Improved load-balancing efficiency

rience and understanding of the problem to find feasible solutions, effectively solving complex computational issues and enhancing resource utilization and throughput in networks.

Reddy et al.[70] proposed a scheme using SDN to manage resources in an energy-efficient manner. This scheme selectively activates a subset of switches, provides multipath routing for all predetermined flows, and installs corresponding forwarding rules on the switches. To solve the associated integer linear programming problem, the authors applied a Particle Swarm Optimization (PSO) heuristic algorithm for feature selection to minimize energy costs. Simulation results demonstrated the effectiveness of the proposed algorithm. While this approach effectively reduces energy costs, it faces potential scalability challenges in large-scale data centers due to the computational complexity of PSO in real-time scenarios. Additionally, the trade-off between energy efficiency and network performance needs a more thorough investigation.

Guo et al.[71] introduced a dynamic flow scheduling scheme named AggreFlow, which enhances scheduling efficiency and service quality. Experimental results indicated AggreFlow improved service quality, enabled load balancing, but consumed less electricity.

BiTE[72] represents a dynamic two-tier traffic engineering model in software-defined data center networks, aiming to balance load distribution and energy efficiency. Because of the complexity of the two-tier planning problem, the authors proposed a co-evolutionary meta-heuristic algorithm to tackle the BiTE challenge. The results showed that BiTE maintained energy efficiency while exhibiting good traffic load balancing performance. Although BiTE demonstrates promising results in balancing load distribution and energy efficiency, the co-evolutionary meta-heuristic algorithm may require significant computational resources for large-scale networks.

Kamboj et al.[73] presented a method to adjust energy consumption according to traffic by migrating traffic from less loaded switches to other devices and then turning them off to reduce energy consumption. The authors presented a heuristic method for traffic scheduling and developed an integer linear programming problem to reduce the total number of active switches. Compared to baseline approaches, this method significantly reduced average latency and improved the throughput.

Dai et al.[74] proposed a dynamic traffic scheduling mechanism based on differential evolution fused with an ant colony algorithm for elephant flows in software-defined networking. This method uses the candidate paths as the algorithm's initial global pheromone, combined with the global network status, to find the optimal global path. Experimental results show that,

compared to existing methods, this approach can significantly improve bandwidth utilization and optimize the network's maximum link utilization rate, thereby achieving better network load balancing.

While heuristic methods have made some progress in traffic scheduling, they still exhibit certain shortcomings in fault scenarios compared to methods based on reinforcement learning. Firstly, heuristic approaches usually generate solutions based on preset rules or an understanding of the problem, which may not adapt well to unknown or dynamically changing environments, especially in fault conditions. Secondly, due to a lack of self-learning and optimization capabilities, heuristic methods may fail to find the optimal solution even after observing a large number of samples during the training process. Therefore, although heuristic methods have achieved certain success in solving traffic scheduling problems, there is a need to integrate them with other machine learning techniques, such as reinforcement learning and deep reinforcement learning, to overcome their limitations in dealing with complex and dynamic network environments, particularly in fault scenarios.

2.3.2 Reinforcement Learning-Based Traffic Scheduling Methods

In recent years, reinforcement learning has shown exceptional problem-solving effectiveness and adaptability, particularly in handling highly dynamic and uncertain scenarios. This has attracted attention from both academia and industry towards RL-based traffic scheduling methods. These approaches combine the flexibility of SDN with the adaptive learning capabilities of reinforcement learning, aiming for efficient and stable network traffic scheduling while ensuring quality of service.

RSIR[75] is an approach that employs the Q-learning reinforcement learning method to calculate routes in SDN. By incorporating a knowledge plane into SDN, it allows forwarding decisions to take link state into account. By utilizing the environment and reinforcement learning's intelligence, this approach performs well regarding stretch and packet loss.

To address the limitations of heuristic algorithms in adapting to traffic matrix and network dynamics, CFR-RL[76] employs the REINFORCE method for traffic rerouting. In order to reroute these chosen key flows and maintain network link utilization balance, it proactively trains a policy that determines critical flows.

Jin et al.[77] proposed congestion control approaches based on Q-learning and Sarsa algorithms.

By introducing reinforcement learning to software-defined data center networking, this method implements flow-based congestion control, intelligently avoiding network congestion while maximizing overall network link utilization.

Compared to heuristic approaches, RL-based methods like RSIR and CFR-RL demonstrate superior adaptability to dynamic network conditions. However, they face challenges in training stability and convergence time, particularly in large-scale deployments. The trade-off between exploration and exploitation in these methods deserves deeper investigation.

Deep reinforcement learning integrates the perceptual strengths of deep learning with the strategic prowess of reinforcement learning, markedly enhancing the management of intricate problems within high-dimensional continuous state and action domains. Some researchers believe that DRL-based traffic scheduling methods hold promising research prospects in traffic engineering[82]. As a result, many researchers have begun to explore DRL-based traffic scheduling methods in data center networks[78]–[80], [83]–[87].

SmartFCT[78] merges DRL and SDN into a traffic scheduling strategy designed to reduce the energy consumption while maintaining flow completion times. This approach dynamically train the DRL model by collected data flow information, which will identify the classifications of flows and send SDN instructions to the switches.

Casas et al.[79] proposed DRSIR, a smart routing algorithm that combines DRL with software-defined networking. DRSIR overcomes the limitations of solutions based on traditional reinforcement learning by generating efficient and intelligent routing paths based on path state indicators to adapt to dynamic traffic changes.

DRL-PLink[80] integrates SDN with the DDPG deep reinforcement learning algorithm for scheduling hybrid flows. It partitions link bandwidth and build dedicated private links for flows to reduce their competition, effectively scheduling hybrid flows with minimal system overhead and achieving good load balancing between paths.

EfficientTE[81] is a novel TE solution that leverages deep reinforcement learning to dynamically adjust traffic routing in response to real-time network demands. This approach enables the DRL algorithm to selectively reroute critical traffic flows based on an analysis of network topology and traffic patterns, utilizing a weighted K-shortest path algorithm for minimal disruption. Experimental results demonstrate that EfficientTE significantly enhances network

performance, achieving substantial improvements in load-balancing efficiency across various network topologies when compared to existing TE methods.

The evolution from traditional RL to DRL-based approaches reveals a clear trend toward more sophisticated decision-making capabilities. While methods like SmartFCT and EfficientTE show promising results in controlled environments, their performance in production networks with heterogeneous traffic patterns and frequent topology changes requires further investigation.

Although the aforementioned methods have made progress in the fields of deep reinforcement learning and data center network traffic scheduling, many challenges remain to be addressed. A significant issue is the vast scale of data center networks, which are constantly subject to numerous link failures, device malfunctions, and software configuration issues. However, current traffic scheduling algorithms usually do not take fault conditions into account. Conventional DRL methods based on neural networks generally lack versatility. When faults cause topological changes, existing methods may not achieve the expected outcomes. Therefore, designing a traffic scheduling method for data center networks with strong generalization capabilities and fault awareness is necessary.

2.4 Summary

This chapter succinctly reviews the essential components and developments in data center networks, covering network topologies, routing methods, fault detection techniques, and traffic scheduling strategies. It highlights the significance of these technologies in maintaining network stability and reliability, mitigating congestion, improving service quality, and driving the advancement of data center network technologies.

Chapter 3

Foundations and Techniques

This chapter introduces the foundational concepts and methodologies supporting our thesis, focusing on the formal description of regular network topology, the principles of software-defined networking, the mechanisms of deep reinforcement learning, and the innovations of graph neural networks. Section 3.1 illustrates the structured architecture of data center networks, which allows for optimization and efficiency in network operations through recursive or iterative construction methods. Section 3.2 delves into the transformative potential of software-defined networking, emphasizing its role in decoupling control and data planes to achieve enhanced network manageability and adaptability. Section 3.3 explores the domain of deep reinforcement learning, showing its significance in optimizing decision-making processes by learning policies that maximize cumulative rewards. Finally, Section 3.4 introduces the cutting-edge developments in Graph Neural Networks (GNNs), which are pivotal for processing and analyzing graph-structured data.

3.1 Formal Description of Regular Network Topology

Regular data center networks, characterized by their predictable and structured architecture, can be effectively constructed using recursive or iterative methods. These networks exhibit distinct patterns in various aspects, such as node positioning, IP address encoding, and the configuration of inter-node connections. These patterns are not just incidental but fundamental to the network's design, offering opportunities for optimization and efficiency in network operations. To fully exploit these inherent patterns, especially for developing advanced routing algorithms, a robust and precise formalism is essential for describing regular data center network topologies.

Our prior work [88] addressed this need by proposing a formal method specifically designed to capture the inherent regularity in network topologies. Alongside this method, we introduced a domain-specific language named Topology Description Language (TPDL). TPDL is tailored to provide an intuitive description of commonly encountered regular network topologies. Its design allows for easy parsing by various software tools, enabling seamless integration into diverse routing algorithms and network programs. This dual approach of a formal method and TPDL significantly enhances the adaptability and efficiency of routing processes in regular topology data center networks.

3.1.1 Formal Description Method for Regular Network Topology

Computer networks comprise numerous network elements, broadly classified into network devices (servers, switches, and routers) and the links connecting these devices. Conventionally, a network topology is depicted as an undirected graph, as follows:

$$G = (V, E) \tag{3.1}$$

In this representation, V denotes the set of all nodes in the graph, and E represents the set of all edges, which are the links between these nodes.

However, this traditional representation method falls short of effectively capturing the regularity inherent in network topologies. Recognizing this limitation, it becomes imperative to refine this approach. Classical regular data center networks, such as Fat-tree and BCube networks, inherently categorize nodes based on their functional roles and physical locations. For example, Fat-tree networks segregate nodes into four distinct classes, while BCube networks feature a

layered node hierarchy. By grouping nodes into categories, each group's nodes exhibit similar functional attributes and connection patterns. This categorization simplifies the representation and allows for a more precise and formal definition of node groups, which is crucial for optimizing network management and routing strategies.

To better formalize the grouping characteristic of regular networks, an undirected graph with multiple types of nodes can be utilized, as expressed in the following equation:

$$G = \left(\cup_{t=1}^{k} V_t,\ \cup_{t=1}^{k} \cup_{t'=t}^{k} E_{tt'}\right) \tag{3.2}$$

In this model, k signifies the number of distinct node sets into which the network's nodes are partitioned, with each set V_t comprising nodes that share similar locations or connection patterns. Here, $\cup_{t=1}^{k} V_t$ represents the union of all node sets, encompassing every node in the network. Similarly, $\cup_{t=1}^{k} \cup_{t'=t}^{k} E_{tt'}$ denotes the comprehensive set of all links, where $E_{tt'}$ specifically indicates the links connecting nodes between any two sets V_t and $V_{t'}$.

The distance formula serves as a fundamental tool for calculating the distance between any two nodes in a network topology. It comprises a set of distance rules, each defining the distance between a pair of source and destination nodes under certain conditions. A singular distance rule can be succinctly represented as a quadruplet: $(type_{src}, type_{dst}, condition, distance)$. Here, $type_{src}$ and $type_{dst}$ indicate the types of the source and destination nodes, respectively. *condition* is a Boolean expression that stipulates the prerequisites for the rule to apply, and *distance* specifies the distance value between the nodes when these conditions are met.

This formula is particularly adept at intuitively capturing how the types and properties of nodes influence their inter-node distances. It enables researchers and network designers to quickly discern distance relationships based on the nodes' characteristics. A significant advantage of this distance formula is its intrinsic design consideration of the regularity inherent in data center networks. As a result, the complexity of the formula remains manageable, irrespective of the network's scale. This feature ensures that the distance formula remains efficient and practical even when applied to describe sprawling, large-scale data center networks.

3.1.2 Topology Description Language

The TPDL is a domain-specific language tailored for the declarative description of data center network topologies. Designed for efficiency, TPDL, in conjunction with its parser, directly supplies topology data to routing programs. This language comprehensively encapsulates various network components, such as nodes and links, along with vital metadata like IP address allocation schemes and the distance formulas within the network.

TPDL's structure is organized into three primary block types:

1) Network device definition: This block specifies the various network devices in the data center, detailing their types and characteristics.

2) Link Definition Block: This block defines interconnectivity and relationships between the different network devices, establishing how these devices are linked within the network.

3) Distance Formula Definition Block: This block contains the rules for calculating the distance between nodes per the specific network topology.

These blocks are demonstrated from Code 3.1 to Code 3.3, respectively. In a TPDL file, users can declare multiple instances of these blocks to describe a range of network devices, their interconnections, and the pertinent distance formulas. The declarative nature of TPDL facilitates network administrators in efficiently describing the data center network topology, making this information readily accessible and utilizable by various network management programs and algorithms.

Code 3.1: A Typical Device Block Definition.

```
1  device AggSwitch {
2      num: 8
3      port: 4
4      address: 0xC0000000
5      attrs: {
6          pod = [1..4], 0x00FF0000
7          index = [1..2], 0x000000FF
8      }
9  }
```

Code 3.2: A Typical Iterate Link Block Definition.

```
link: {
    for i = 1..2,j = 1..4,z = 2..3 {
        EdgeSwitch[${j}][${i}] <--> server[${j}][${i}][${z}]
    }
}
```

Code 3.3: A Typical Distance Formula Block Definition.

```
distance server:s1, server:s2 {
// When s1 and s2 are connected to the same edge switch, their distance is 2 hops
condition: ${s1.pod} == ${s2.pod} && ${s1.edge} == ${s2.edge} => value: 2;

// When s1 and s2 are located in the same pod but connected to different edge
    switches, their distance is 4 hops
condition: ${s1.pod} == ${s2.pod} && ${s1.edge} != ${s2.edge} => value: 4

// When s1 and s2 are located in different pods, their distance is 6 hops
condition: ${s1.pod} != ${s2.pod} => value: 6;

}
```

The TPDL aggregates devices, connection relationships, and distance formulas of regular data center networks, providing a comprehensive view of the entire topology for the control plane. Remarkably, the corresponding TPDL file can be succinct even for large-scale data center networks. For instance, a TPDL file for a 16-server Fat-Tree topology comprises only 140 lines. This brevity remains consistent even when expanding the network topology to accommodate 1024 or more servers. As long as the Fat-Tree network structure is maintained, the network topology can still be described using a 140-line TPDL file. Therefore, in the context of data center networks, TPDL serves as an efficient and concise method for topology description.

3.2 Software Defined Networking

Software-defined networking is a revolutionary concept that redefines the fundamentals of networking infrastructure, resulting in more centralized and flexible network management. Traditionally, network devices' control and data forwarding functions were tightly integrated, leading

to complex and inflexible network configurations and management. SDN separates control functions from data transmission functions in traditional hardware-driven network devices. This technological evolution is grounded in the growing need for network flexibility and programmability in domains such as cloud computing and big data. SDN addresses inherent challenges in traditional network design, such as complexity, lack of flexibility, and scalability. Furthermore, owing to its powerful customizability and scalability, SDN has begun to see widespread adoption in data centers, service provider environments, and enterprise networks.

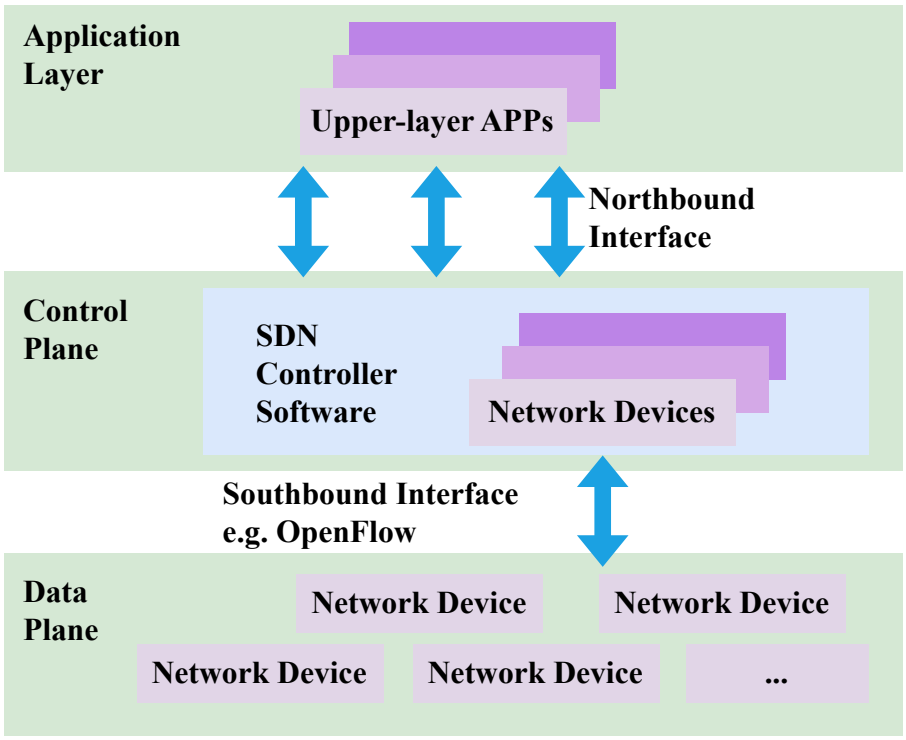


Figure 3.1: A typical architecture of the software-defined networking.

As illustrated in Figure 3.1, a typical SDN architecture is fundamentally composed of three layers: the application plane, the control plane, and the data plane [89]. The SDN controller is central to the control plane, orchestrating the network's operations. It interacts with network devices via the southbound interface, facilitating network status information collection and forwarding instructions to switches. Additionally, the controller offers a northbound interface, enabling applications and services to access network services programmatically.

In the data plane, switches, often referred to as "dumb" switches in this context, relinquish autonomous forwarding decision-making. Instead, they adhere to the flow table rules distributed by the SDN controller. OpenFlow [26], a prominent protocol within SDN, standardizes the communication between the control and data planes. While some companies opt for propri-

etary southbound protocols, OpenFlow stands out for its openness and widespread adoption. It empowers SDN controllers to directly manipulate the flow tables on OpenFlow-compatible devices, dictating the flow of data across the network.

SDN architecture also champions network virtualization, enhancing resource utilization and network flexibility by facilitating the creation and management of virtual networks atop the physical infrastructure. The architecture's inherent automation and programmability significantly streamline network management.

The principal benefits of SDN encompass simplified network management, improved resource utilization, increased flexibility, and scalability [90]. However, SDN also faces new challenges that require further optimization. While simplifying control logic, the SDN controller's centralized nature can pose a bottleneck risk [91]. In large-scale data center networks, the proliferation of switches can overwhelm the controller's capacity, potentially leading to service degradation or outright failure.

3.3 Deep Reinforcement Learning

Reinforcement learning involves training intelligent agents to optimize their decision-making strategies through iterative interactions with their environment. This process aims at maximizing cumulative long-term rewards [92]. One of the most important mechanisms of RL is trial-and-error, where agents gradually refine their actions based on the rewards or penalties they receive. The versatility and broad applicability of reinforcement learning in a variety of fields, including gaming, autonomous driving, resource management, and power system optimization, highlight its usefulness.

The underlying architecture of reinforcement learning, as depicted in Figure 3.2, includes several fundamental components. These are:

1. **Agent**: The learner or decision-maker in the RL process.
2. **Environment**: The external system with which the agent interacts.
3. **States**: Different situations or configurations of the environment.
4. **Actions**: The set of possible decisions or moves the agent can make.
5. **Rewards**: Feedback from the environment in response to the agent's actions, guiding

the learning process.

This architecture facilitates a dynamic learning process where the agent continuously learns from the consequences of its actions, adjusting its strategy to achieve the most favorable outcomes.

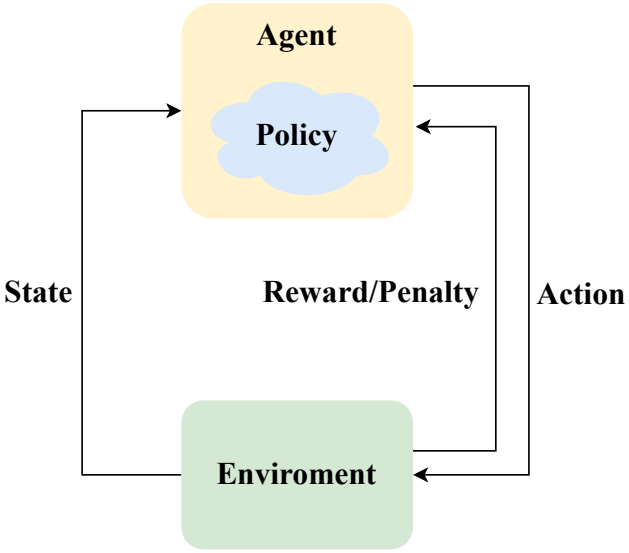


Figure 3.2: The basic architecture of the reinforcement learning.

The primary objective of RL is to empower an intelligent agent to discern optimal actions based on environmental states, aiming to maximize cumulative rewards through its interactions with the environment. This process involves the agent engaging in three key interfaces with its surroundings:

1. **Receiving States**: The agent receives current state of the environment.
2. **Executing Actions**: It sends out actions to be executed in the environment.
3. **Receiving Rewards or Penalties**: The agent obtains rewards or penalties from the environment according to its last actions.

At the heart of these interactions lies the agent's policy, a set of rules or strategies determining the actions it selects in response to specific states. Markov Decision Processes (MDPs) [93], often utilized in RL, provide a structured framework for formulating these policies. An MDP is characterized by a quadruple $(\mathcal{S}, \mathcal{A}, \mathcal{P}_{sa}, \mathcal{R})$, where:

- $\mathcal{S}$ represents the set of all possible states, with $s_t \in \mathcal{S}$ denoting the state at time t.
- $\mathcal{A}$ signifies the set of all candidate actions, with $a_t \in \mathcal{A}$ dicating the action taken at time

t.

- $\mathcal{P}_{sa}$ denotes the state transition probability matrix, which describes the likelihood of transitioning to a new state s_{t+1} after executing action a_t in state s_t.
- $\mathcal{R}$ represents the reward function, where $r_t \in \mathcal{R}$ is the reward received upon transitioning from state s_t to state s_{t+1} by executing action a_t.

The Markov property is central to MDPs and is expressed as:

$$p(s_{t+1}|s_t) = p(s_{t+1}|h_t) \tag{3.3}$$

This suggests that the current state is the only factor that determines the likelihood of changing to a future state, independent of the historical sequence of states $h_t = \{s_1, s_2, \ldots, s_t\}$. RL strategies explore various actions to gain feedback from the environment. By adjusting their state transition probabilities based on this feedback, agents can learn sequences of actions that yield higher rewards, thus achieving predefined goals or tasks.

Conventional reinforcement learning techniques, including tabular Q-Learning [94], have performance limitations in complex scenarios due to state space size they're able to handle. However, a notable shift has been observed with the rapid advancement of deep learning. Researchers have found that deep neural networks are capable of approximating complex state transition probabilities, thereby enabling solutions to more intricate problems. This integration of deep learning with traditional reinforcement learning algorithms has led to the emergence of deep reinforcement learning. Studies indicate that DRL approaches significantly enhance performance in complex environments [95], showcasing their versatility across diverse applications [96].

A prominent example of DRL's success is the Proximal Policy Optimization (PPO) algorithm [97]. Evolving from earlier policy gradient methods, PPO has risen to prominence, particularly at leading AI research entities like OpenAI. PPO distinguishes itself through its training stability and efficient sample utilization. It achieves this by judiciously constraining policy updates during each learning iteration, circumventing drastic, potentially destabilizing changes. Moreover, PPO's ability to recycle past experiences enhances its sample efficiency. In essence, the algorithm's controlled policy updates and reuse of experience effectively mitigate common challenges in reinforcement learning, such as instability and inefficiency in sample use.

The PPO algorithm achieves policy updates by optimizing the following objective function:

$$\mathcal{L}\text{PPO}(\theta) = \frac{1}{T}\sum_{t=1}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} A_t^{\text{clip}}, \right.$$
$$\left. \text{clip}\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon\right) A_t^{\text{clip}}\right) - \lambda\mathcal{H}(\pi_\theta(a_t|s_t)) \tag{3.4}$$

$$\theta \leftarrow \theta - \alpha\nabla_\theta\mathcal{L}_{\text{PPO}}(\theta) \tag{3.5}$$

The term $\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ signifies the ratio of the new policy probability to that of the old policy. Optimizing the objective function, which includes this ratio, facilitates policy improvement. The 'clip' function within PPO plays a crucial role by limiting the extent of each policy update, thereby avoiding large and potentially destabilizing changes. The parameter ϵ, typically set to a small constant like 0.1 or 0.2, defines the acceptable bounds for these policy updates. This approach strikes a balance between exploration and utilization, ensuring both stability of training and the efficient use of samples.

PPO stands out for several reasons compared to other reinforcement learning algorithms:

- Stable Policy Updates: PPO maintains stability in policy evolution by controlling the magnitude of each update. This cautious approach prevents the policy from undergoing drastic, potentially harmful changes.

- Improved Sample Efficiency: PPO extracts more value from historical information by reusing multiple epochs of batch data, thus enhancing its sample efficiency.

- Simplicity and Accessibility: One of the most appealing aspects of PPO is its straightforward implementation. Unlike some other algorithms, PPO does not necessitate intricate additional loss terms, making it easier to understand and modify.

PPO has demonstrated its versatility and effectiveness in tackling a range of complex tasks, from game AI and robotic control to applications in natural language processing. Its blend of simplicity, intuitive implementation, and robust performance has led to its widespread adoption in the research and practical applications of reinforcement learning.

3.4 Graph Neural Networks

The field of Graph Neural Networks emerged from the intersection of deep learning and graph theory, with its foundations traced back to early works in the early 2000s. The seminal paper by Gori et al. [98] introduced the first Graph Neural Network model, marking a significant milestone in processing graph-structured data using neural networks. This innovation was further developed by Scarselli et al. [99], who formalized the GNN framework and established its theoretical foundations.

Graphs are intricate data structures consisting of nodes and edges commonly used to model complex systems such as social networks, protein-protein interactions, brain networks, road systems, physical interaction networks, and knowledge graphs. The representation of data in graph form offers significant advantages, particularly in modeling relationships within a system and simplifying complex problem representations. However, traditional deep neural network-based methods often struggle with graph-structured data. This challenge arises due to the inherent irregularities of graphs, including their non-uniform and unordered node structures and dynamically changing neighborhoods. These characteristics make it challenging to apply standard mathematical operations, like convolution, directly to graph data. With the advancement of deep learning, researchers have increasingly turned to graph neural network architectures to handle graph-structured data effectively.

The breakthrough in applying deep learning to graphs came with the development of spectral graph convolutions by Bruna et al. [100], which laid the groundwork for modern Graph Convolutional Networks. This was followed by several key innovations, including the introduction of spatial-based graph convolutions and the development of scalable approaches for large-scale graph processing.

In the realm of graph neural networks, several key methodologies stand out for their unique approaches and contributions:

(1) **Message Passing Neural Network (MPNN)**: The MPNN framework unifies various GNN types, offering an efficient model for capturing complex interactions between nodes and edges in a graph [101]. This framework operates on the principle of message passing, wherein nodes exchange messages based on their adjacent nodes and edges [102]. The essence of MPNN lies in nodes developing their representations by communicating with their neighbors, enabling

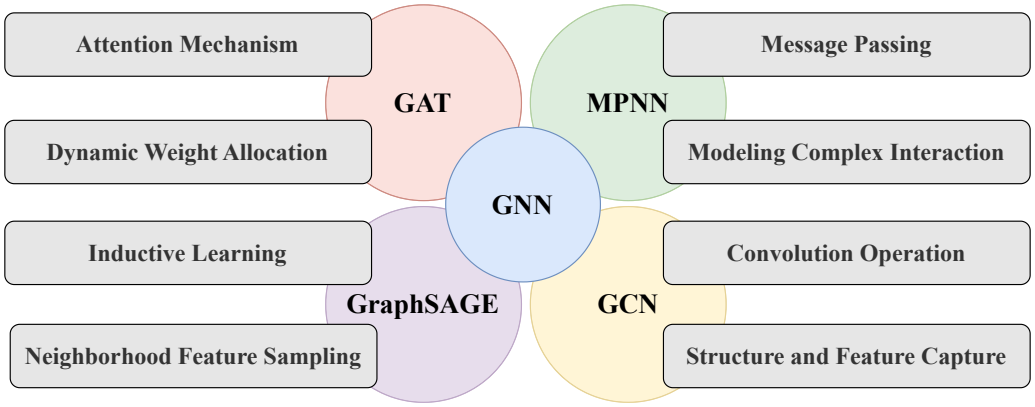


Figure 3.3: Key improvement methods for Graph Neural Networks.

both local and global representation computations across the graph.

(2) **GraphSAGE**: An inductive approach to node embedding, GraphSAGE leverages node attributes to learn a function for embedding. This method allows for simultaneous learning of graph topology and node feature distribution, providing a comprehensive understanding of graph data. GraphSAGE begins by sampling node features from local neighborhoods, balancing computational efficiency and neighborhood context incorporation. It aggregates information from neighboring nodes using a learned function mapping, ensuring effective information propagation throughout the network layers [103].

(3) **Graph Convolutional Network (GCN)**: GCNs apply the concept of convolution to graph data, enhancing node representations through multiple convolution layers. This stacking approach enriches the structural and feature information representations within the graph. GCNs have shown substantial promise in complex graph-structured problems, demonstrating versatility in various application domains [104]. Notably, GCNs can model unstructured data by constructing relationships between data points, thus rendering non-graph data compatible with GNN analysis.

(4) **Graph Attention Network (GAT)**: GAT introduces an attention mechanism for aggregating features from neighboring nodes. Contrary to traditional methods like GCN and GraphSAGE, which assign fixed or equal weights to adjacent nodes, GAT dynamically learns the weights between node pairs. This allows nodes to modulate their influence based on their relative significance and relationship to the target node, enabling a more nuanced capture of dependencies and variations in node importance [105].

Graph neural networks are renowned for their ability to effectively capture the positions and

relationships of nodes within a graph. This capability significantly enhances the network's expressive power, making GNNs a potent tool for a variety of graph analysis tasks.

In this thesis, we propose an innovative approach that merges the strengths of message-passing neural networks with deep reinforcement learning. This combination is tailored to tackle the complexities of data center network traffic scheduling, especially under conditions of fault tolerance. The choice of MPNNs as a central component is driven by several considerations. Firstly, the inherent flexibility of MPNNs positions them as an ideal fit for data center networks, where the topology is subject to continual changes due to failures. MPNNs demonstrate remarkable adaptability to a range of graph structures, including heterogeneous and dynamic ones. Moreover, the strength of MPNNs lies in their ability to aggregate and propagate node features efficiently. Their message-passing mechanisms ensure that information across the network is fully leveraged, which is crucial for effective network management. Further enhancing this approach is integrating with deep reinforcement learning. This combination is particularly suited for the dynamic nature of network scheduling problems, which inherently involve elements of reinforcement learning. While other neural network architectures like GCN, GAT, and GraphSAGE can also be integrated with DRL, they primarily cater to supervised learning contexts. Such architectures may only partially meet the continuous decision-making and optimization demands of network scheduling. Conversely, MPNNs demonstrate a superior capacity to process and utilize environmental feedback. This ability enables them to iteratively learn and derive optimal strategies for traffic scheduling through a process of trial and error.

3.5 Summary

This chapter delves into the core theories and methodologies that are integral to the research presented in this thesis. These foundational elements are pivotal in underpinning the research work discussed herein. The regular network topology description language provides a foundation for describing data center network topologies and utilizing their regularity in this thesis. Software-defined networking is the underlying architecture of modern data center networks, and much of the research work in this thesis leverages the software-defined networking architecture. Deep reinforcement learning and graph neural networks are effective optimization tools for addressing failure-probing matrix and traffic scheduling problems in networks.

Chapter 4

Routing Methods for Data Center Networks with Regular Topologies

The increasing prevalence of cloud computing and big data technologies has led to the expansion of data centers on a large scale. Traditional routing methods, such as BGP[22] and OSPF[20], often struggle with high overhead and slow convergence rates in large-scale data center networks. To meet the evolving communication and management demands of these expansive networks, researchers have proposed a range of innovative network topologies and routing methods designed to enhance performance. Notable examples include Fat-Tree[29] and BCube[30], among others.

An important characteristic of these emerging data center network topologies is their regular network structure design, which aims to improve routing efficiency. Regular network topologies are marked by a uniform pattern in their structure, defined through iterative and recursive methods. These topologies can be broadly classified into two categories: switch-centric networks (e.g., Fat-tree[29], VL2[37]) and server-centric networks (e.g., BCube[30], DCell[36]). Such regular data center network topologies are definable and constructible using parameters associated with their topological features, as outlined in recent studies[35].

From the perspective of routing methodologies, many routing methods have adopted topology-aware techniques. These techniques leverage the regularity of the network's topological structure to enhance routing efficiency. Unlike traditional link-state routing methods such as OSPF, topology-aware routing approaches incorporate information about the network's physical topol-

ogy and addressing patterns during the design phase, optimizing the performance of path selection algorithms.

Although emerging network topologies and corresponding routing methods can provide higher forwarding efficiency for large-scale data center networks, there are still numerous issues that hinder their practical deployment. Firstly, these topology-aware routing methods are typically designed for specific network topologies, lacking universality. During their design, many routing methods have already internalized the characteristics of the network topology into the algorithm. As a result, while these methods are highly efficient, they are tightly coupled with the corresponding topology. If the network topology changes, these algorithms will become ineffective or invalid. Secondly, these routing methods did not consider interoperability in their design, lacking a unified interaction interface between different routing approaches. In modern data center networks, heterogeneity is also a key feature. A data center may consist of multiple heterogeneous network topologies [106], and communication between these topologies cannot be ignored. However, current topology-aware routing methods cannot be collaboratively deployed and operate within data centers. Topology-aware routing techniques still need to be improved to implement and use in data center networks due to these drawbacks.

On the other hand, while software-defined data center networks offer centralized dynamic network management and traffic scheduling, they also introduce new challenges. As the network scale continues to expand, the overhead of the OpenFlow communication channel between switches and controllers rapidly increases. Even though the OpenFlow traffic generated by a single switch is modest, the cumulative overhead from potentially hundreds of thousands of switches in a large-scale data center can significantly burden the controller, making the controller's processing capacity a potential bottleneck in large-scale SDNs [107]. Furthermore, software-defined networks typically rely on OpenFlow's Packet-In mechanism to initialize flow paths. Controllers must handle Packet-In messages for each new flow, leading to a rapid increase in controller load as traffic grows. This on-demand computation can also prolong the initial packet delay of a flow, potentially violating the ultra-low latency requirements for time-sensitive applications.

This chapter introduces two topology-aware routing algorithms, centralized and semi-centralized, based on the TPDL. These algorithms utilize TPDL to understand the regularities in data center network topologies. By leveraging these regularities, they achieve efficient routing and

forwarding capabilities. Additionally, they address issues such as control plane bottlenecks in software-defined data center networks.

4.1 TPDL-based Centralized Routing Method for Data Center Networks

The introduction of the TPDL offers a standardized approach to describing regular data center network topologies, paving the way for a universally applicable, highly efficient routing method tailored for data center environments. This section introduces a topology-aware routing methodology named cRetor, specifically designed for large-scale data center networks within the SDN framework. cRetor harnesses the inherent regularity of data center network topologies for efficient topology discovery and path computation. It seamlessly integrates TPDL into the SDN controller's architecture, significantly reducing the controller's workload and enhancing its capacity to manage a larger number of switches. This scalability is a vital attribute for expansive network infrastructures.

4.1.1 System Overview

cRetor is a routing framework fundamentally based on the TPDL and tailored for software-defined data center networks. It inherits a centralized control architecture from the principles of SDN. The primary objectives of cRetor include minimizing topology discovery overhead in software-defined networks and introducing an effective routing calculation strategy that supersedes the conventional shortest path algorithm, such as Dijkstra's algorithm.

The main design ideas of cRetor are as follows:

1. Introduce a TPDL module on the cRetor controller for topology establishment and fault detection, reducing the overhead of topology discovery.

2. Replace the original shortest path calculation algorithm on the cRetor controller with a more efficient A^* algorithm based on TPDL to improve the path computation efficiency.

3. Introduce an active fault probing and reporting mechanism on cRetor switches to enhance cRetor's dynamic topology awareness capabilities.

A central aspect of the cRetor framework is the employment of the TPDL as foundational knowledge for the SDN controller. By leveraging TPDL's topology data, the cRetor controller can form a foundational understanding of the network's overall state. Additional controller components are tasked with detecting dynamic topology changes and managing faults. Reflect-

ing the inherent stability and limited variability of data center networks [108], cRetor integrates a substantial volume of static topology information into the controller at the system's initiation. This approach allows for minimal dynamic topology data processing during operation, primarily focusing on scenarios like link failures. This strategy, when contrasted with the conventional topology detection mechanism that relies on the Link Layer Discovery Protocol (LLDP) in standard SDN setups, enables cRetor to significantly alleviate the control plane's workload and reduce extraneous overhead.

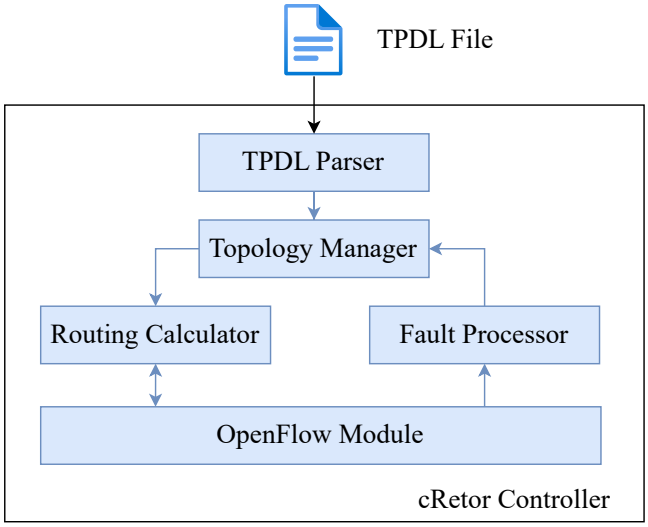


Figure 4.1: The architecture of cRetor controllers.

Figure 4.1 illustrates the architecture of the cRetor controller. A critical component of this structure is the TPDL parser, tasked with interpreting input TPDL files and sending the parsed data to the topology manager. Utilizing the TPDL-derived data, the topology manager constructs and maintains an in-memory network topology, periodically updating it to reflect the real-time network state during the controller's operation.

The routing calculator module employs an A^* algorithm informed by distance metrics and is responsible for determining the most efficient forwarding paths based on the available topology information. Meanwhile, the OpenFlow module, adhering to the OpenFlow protocol, manages message exchanges through a secure channel between the switch and the controller. Upon receiving a Packet-In message from a switch, this module prompts the routing calculator to determine the optimal route for the respective flow. Additionally, the fault processor module, receiving topology change notifications from the OpenFlow module, alerts the topology manager to update the network topology accordingly.

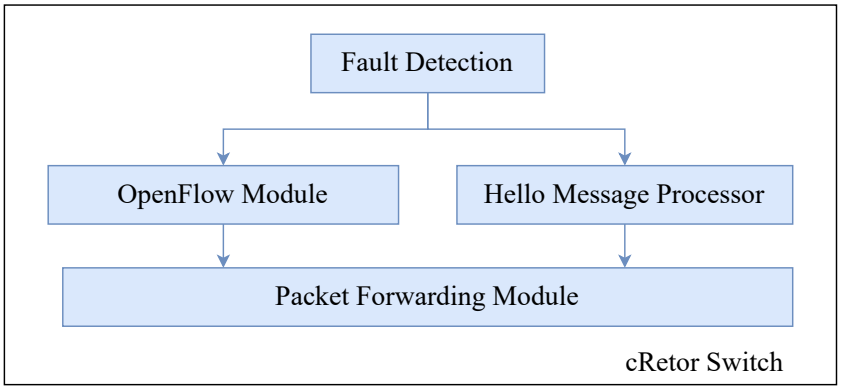


Figure 4.2: The architecture of cRetor switches.

The cRetor switch, illustrated in Figure 4.2, represents an extension to the standard SDN switch architecture. It is specifically designed to integrate seamlessly with existing SDN infrastructures while introducing additional functionality for enhanced fault tolerance. Central to its design are two key components: a Fault Detection Module and a Hello Message Processor (HMP), both dedicated to proactive fault detection. The Fault Detection Module is responsible for monitoring the health of connections with neighboring switches, utilizing a continuous exchange of Hello messages processed by the HMP. This module's effectiveness hinges on its ability to detect anomalies in message transmission, indicating potential link failures.

The HMP operates as a non-interactive unit, only transmitting and receiving Hello messages without requiring acknowledgments from peer switches. It periodically broadcasts these messages across all ports, thereby broadcasting the switch's identifier to neighboring nodes. Upon receipt of a Hello message, a peer switch's HMP logs the sender's details, enabling the Fault Detection Module to assess the operational status of each link. A failure is presumed if a Hello message from a neighboring node is not received within a predefined time frame, prompting an alert to the network controller. The details of this fault handling mechanism are further explored in Section 4.1.2.

Our implementation efforts focused on enhancing the standard Open vSwitch to embody the cRetor switch design. This upgraded switch maintains compatibility with conventional SDN controllers and switches, allowing for a smooth integration of cRetor into existing networks. Consequently, cRetor benefits from established SDN algorithms and infrastructure, including traffic engineering and Quality of Service (QoS) algorithms. Furthermore, these existing mechanisms could potentially benefit from the topology awareness facilitated by the TPDL.

4.1.2 Algorithm Design

1. Path Calculation Algorithm

In traditional SDN controllers, path selection relies on algorithms such as Shortest Path First (SPF) and Constrained Shortest Path First (CSPF). Controllers, like Floodlight, typically use caching to reduce shortest-path calculations. A notable improvement in this area is Yen's k-shortest path algorithm, which enhances the traditional Dijkstra algorithm. However, SDN controllers still need to recompute these paths for all nodes whenever there is a change in network topology.

Contrastingly, the cRetor framework leverages the TPDL to obtain a comprehensive understanding of the data center network topology. This approach eliminates the need for the intensive LLDP topology discovery process. Initially, the controller constructs a base network topology at startup. It then maintains and updates this topology in real time, informed by fault data from the switches.

Moreover, with TPDL integrated into the controller, routing computations between two network nodes can be performed with significantly reduced overhead. This efficiency opens the door for developing customized routing algorithms that are more efficient than traditional methods. Consequently, cRetor departs from using Dijkstra's algorithm for shortest-path calculations. Instead, it adopts the A^* algorithm, utilizing the distance function provided by TPDL as its heuristic. This method enhances path computation efficiency by leveraging the topology-awareness inherent in TPDL.

The A^* algorithm distinguishes itself from traditional search algorithms through the introduction of a heuristic function. This function estimates the distance from the current node to the destination node. The algorithm guarantees the identification of the shortest path between the source and destination nodes, provided that the heuristic estimate is less than or equal to the actual distance. In cases where the estimated value precisely matches the actual distance, the A^* algorithm can efficiently trace the shortest path directly from the source to the destination. When the heuristic function consistently outputs zero, the algorithm will downgrade to be equivalent to the Dijkstra algorithm [109].

A critical aspect of the A^* algorithm is its search order, which is directed by the cost function $f(n)$. This function is defined as shown in Eq. (4.1):

$$f(n) = g(n) + h(n) \tag{4.1}$$

In this equation, n represents the current node. The term $g(n)$ denotes the actual cost of traveling from the source node to n, and $h(n)$ is the estimated cost from n to the destination node. In cRetor, the heuristic function $h(n)$ employs the distance formula provided by TPDL, as detailed in Algorithm 1. Thus, $f(n)$ represents the total estimated cost from the source node to the destination via node n. The A^* algorithm iteratively minimizes $f(n)$ to navigate towards the destination node. Therefore, the accuracy of the heuristic function $h(n)$ directly impacts the algorithm's efficiency in finding the optimal path.

Algorithm 1: TPDL Distance Calculation Algorithm

Input:

n_s: Source node

n_d: Destination node

list: List of distance formula rules in the TPDL file

Output: Distance between nodes n_s and n_d

```
1  distance ← ∞;
2  for each rule in list do
3      if Node types of n_s and n_d match those in rule then
4          cond ← Calculate the value of the condition expression in rule;
5          if cond = true then
6              distance ← rule.distance;
7              break;
8          end
9      end
10 end
11 return distance;
```

We analyze the performance of our path calculation algorithm, which combines the A^* algorithm with the TPDL distance formula, in different network conditions: fault-free and partial fault scenarios. We define $d(n)$ as the actual distance from the current intermediate node n to the destination node.

- **Fault-Free Network Topology**: In a network without faults, the TPDL distance formula precisely gives the actual distances between nodes, meaning $h(n) = d(n)$ in these scenarios. Consequently, this accuracy makes the TPDL distance formula an optimal heuristic, enabling the A^* algorithm to identify the shortest path with 100% accuracy rapidly.

- **Partially Faulty Network Topology**: When partial faults are present, the TPDL distance formula still indicates the shortest path, but actual distances might be longer due to necessary detours around faults ($h(n) \leq d(n)$). The algorithm might take longer to find the shortest path in such scenarios. However, since faulty links usually represent a minor portion of a data center network, the impact on the original path is often limited to one or two link failures. In these cases, the A^* algorithm initially follows the shortest path toward the fault. Upon encountering a fault, it backtracks to identify an alternative route. It then continues along the updated shortest path recalculated by the TPDL formula to reach the destination.

2. Fault Recovery Mechanism

Unlike general-purpose networks, data center networks require exceptionally high network availability and reliability standards. This heightened demand makes robust fault detection and recovery mechanisms indispensable. In standard SDN solutions, the LLDP mechanism serves a dual purpose: it enables controllers to discover network topology and detect network faults. However, in cRetor, the traditional LLDP-based topology discovery approach is replaced by the TPDL, necessitating an alternative, more efficient method for fault detection.

cRetor addresses this need through a lightweight fault-awareness mechanism that revolves around the periodic transmission of Hello messages. Each switch in the network broadcasts these messages across all its ports at regular intervals. A switch receiving a Hello message from a neighboring switch can infer that the link to that neighbor is operational. cRetor adopts a threshold of three times the Hello message interval to detect link failures. If a switch fails to receive a Hello message from a neighbor within this timeframe, it identifies the link as faulty. This fault status is then reported to the controller via a secure tunnel, ensuring rapid and secure communication of network status.

cRetor's fault-tolerant recovery mechanism is a blend of proactive and passive strategies. When

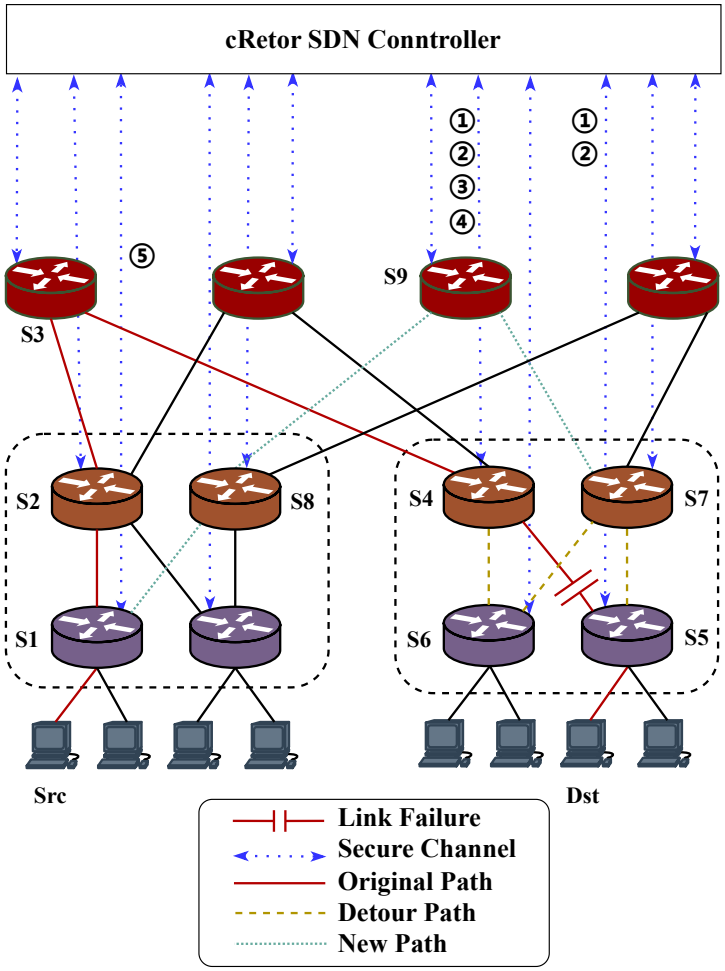


Figure 4.3: A 4-pod Fat-Tree topology (only two pods are shown). The original path from Src to Dst is (Src, S1, S2, S3, S4, S5, Dst). After the link failure between S4 and S5 occurring, the detour path for packets on the way turns into (S4, S6, S7, S5). New path from Src to Dst becomes (Src, S1, S8, S9, S7, S5, Dst).

a fault is detected, the affected switch notifies the controller by sending a Port-Status message. The controller then proactively updates its real-time topology and instructs the affected switches to delete flow table entries related to the fault. This proactive approach ensures a rapid response to network changes. New packets arriving at the switch trigger a Table-Miss event during a network fault as they find no matching entries in the flow table. This results in the switch sending Packet-In messages to the controller. The controller, equipped with the latest network information, recalculates the forwarding path using the advanced routing algorithm discussed earlier. This recalculated path is comprehensive, containing the shortest route both from the current switch to the destination and from the source to the destination, thus avoiding inefficient detours.

An illustrative example of this process is depicted in Figure 4.3, which shows a Fat-tree topology with $k = 4$ Pods. In a fault-free scenario, the standard path from a source node to a destination node is (Src, S1, S2, S3, S4, S5, Dst). When a fault occurs between switches S4 and S5, cRetor manages it as follows:

1. Switches S4 and S5 identify the fault via Hello messages and report it to the controller.
2. The controller updates the real-time topology and sends Flow-Mod messages to S4 and S5 to remove flow table entries related to the S4-S5 link.
3. Subsequent packets from the source to the destination arriving at S4 cause a Table-Miss. S4 then informs the controller.
4. The controller determines the new optimal path (S4, S6, S7, S5) and guides S4 to forward packets to S6.
5. Simultaneously, it recalculates and updates the entire path to (Src, S1, S8, S9, S7, S5, Dst), directing S1 to route packets to S8 instead of S2.

This updated path ensures that packets already in transit continue toward the destination via the new detour while new packets follow the revised optimal path. Without step 5, packets would follow a suboptimal path (Src, S1, S2, S3, S4, S6, S7, S5, Dst), leading to increased latency and costs. cRetor's approach, therefore, not only adapts quickly to faults but also maintains optimal network performance.

4.1.3 Experiments and Evaluation

To assess the performance of cRetor, we conducted a comparative analysis against several established routing schemes, each selected for its relevance to cRetor's intended application in data centers. First, we compared cRetor with the OSPF protocol, a typical link-state routing protocol extensively used in data centers [110]. Additionally, we evaluated cRetor against Floodlight, which exemplifies traditional SDN controllers. PEMA[111] is a routing algorithm designed for software-defined data center networks, which improves traditional SD-DCN architectures. Lastly, we included DCell [36] in our comparison due to its topology-aware routing approach, which aligns closely with the architectures of Fat-Tree and BCube.

For these experiments, we built the cRetor controller on the Ryu SDN controller framework [112] and developed the cRetor switch using Open vSwitch [113]. A TPDL parser, created with

ANTLR [114], was integrated into the cRetor controller. We utilized Mininet [115] to set up our basic experimental environment, creating various topology scales on the simulation platform. For comparison, standard SDN and OSPF environments were also established using Mininet. The OSPF environment employed Quagga software for routing [116], while the SDN-based DCell setup [117] was constructed to reproduce the performance characteristics reported in the original DCell paper [36]. The configuration details for each scheme are summarized in Table 4.1.

Table 4.1: Experimental settings for the four routing schemes.

Scheme	**Controller**	**Switch**	**Routing Method**
cRetor	Ryu-based cRetor controller	Enhanced OVS switch	A^* algorithm based on TPDL
Floodlight	Floodlight controller	OVS switch	Dijkstra's algorithm
OSPF	–	Linux virtual switch	OSPF implementation by Quagga
DCell	Pox-based DCell controller	OVS switch	DCell routing algorithm
PEMA	PEMA Controller	OVS switch	PEMA routing algorithm

We evaluated cRetor's performance in five key areas: 1) path computation time, 2) network convergence time, 3) first packet delay, 4) control message overhead, and 5) fault recovery time. Each metric was chosen for its significance in measuring the efficiency and effectiveness of network routing protocols.

1. Path Calculation Time

In this experiment, we compare the path calculation performance of the A^* algorithm, utilizing TPDL's distance formula as a heuristic, against the widely-used Dijkstra algorithm. We conducted experiments across Fat-tree network topologies of various scales, defined by the scale parameters k of 4, 8, 12, and 16. Both algorithms were implemented using the NetworkX framework [118], and the experiments were run on an Intel Core i7 3.41GHz workstation with Python 3.7.

The evaluation generated 1,000 random pairs of source and destination nodes for both algorithms and recorded their computation times. As depicted in Figure 4.4, we observed that at smaller network scales, the computation times of the two algorithms were comparable, with the

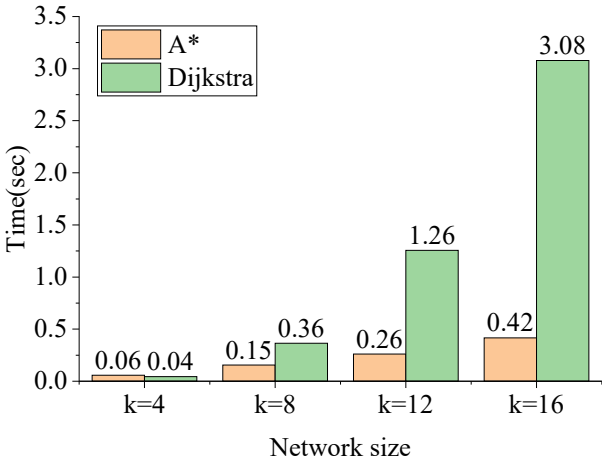


Figure 4.4: Comparison of routing calculation time of the Dijkstra's algorithm and cRetor's A^* algorithm in different network scales.

Dijkstra algorithm occasionally outperforming the A^* algorithm. This slight advantage for Dijkstra can be attributed to the additional distance calculations required by A^*. However, as the network scale increases, the A^* algorithm demonstrates a significant performance advantage, exhibiting near-linear growth in computation time compared to the exponential growth of the Dijkstra algorithm. This efficiency makes the A^* algorithm particularly suitable for large-scale data center networks.

Further, we also assessed the performance in fault-aware Fat-tree networks with k values of 12 and 16. The evaluation program randomly selects links in the network based on different link failure rates, removing them from the standard Fat-Tree topology. Both algorithms then perform the path calculations from the previous experiment on the modified network topologies and record the time the two algorithms took.

Figure 4.5 reveals that the computation time of the A^* algorithm increases with higher link failure rates due to the reduced accuracy of the heuristic function and consequent backtracking. Conversely, the Dijkstra algorithm's computation time decreases as failure rates rise, owing to fewer edges in the altered topologies. Notably, even with a 20% link failure rate, the A^* algorithm maintained a faster computation time than Dijkstra. This finding is especially relevant given that such high failure rates are rare in real-world data center networks [119], reinforcing the practicality of the A^* algorithm in typical network conditions.

The superior performance of the proposed method has significant implications for data center networks. As cRetor can remain its performance advantage even as networks scale up, which is

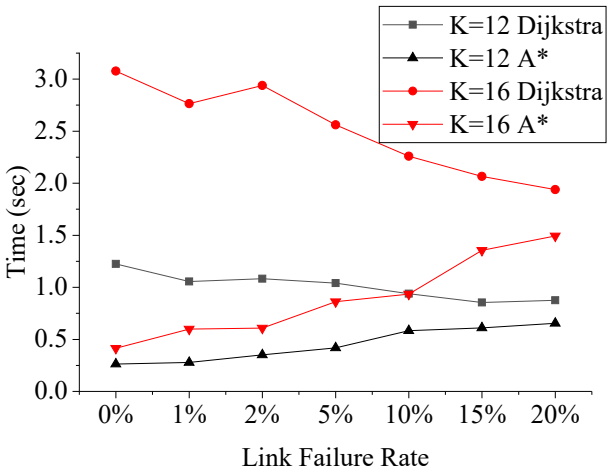


Figure 4.5: Comparison of routing calculation time of the Dijkstra's algorithm and the cRetor's A^* algorithm in different link failure rates.

a critical factor in future-proofing routing solutions for evolving data center architectures. In addition, the ability to maintain efficient path calculation even under such adverse conditions ensures that cRetor can provide reliable routing in dynamic and fault-prone environments, contributing to overall network stability and performance.

2. Network Convergence Time

In this section, we compare the network convergence time of cRetor with three alternative routing schemes. Network convergence time is a critical performance metric, indicating how quickly a network stabilizes after changes or initialization, allowing all nodes to communicate effectively. To conduct this comparison, we utilized Mininet to build test topologies of varying scales, tailoring each to the specific requirements of the different routing schemes.

Due to inherent differences in the structural designs of DCell and Fat-Tree topologies, achieving identical node scales was not feasible. We selected the closest matching scales for a fair comparison: 16 and 128 server nodes for Fat-tree and 20 and 132 server nodes for DCell. The configurations for these experimental scenarios are listed in Table 4.1.

It is important to note that the varying architecture paradigms of the routing schemes naturally lead to differences in how network convergence time is measured. In this experiment, we defined network convergence time as the duration from the initiation of the simulated network to the point where any two nodes within the network can successfully communicate with each other. This metric was chosen as it provides a practical and direct measure of the network's operational

readiness post-deployment or post-modification. The measurement methodology was carefully designed to ensure an objective and comparable evaluation across different network structures and routing strategies.

In measuring the network convergence time for various routing schemes, the following methodologies were adopted:

- **OSPF**: For this distributed algorithm, convergence time is defined as the duration from network startup to the entire establishment of routing table entries in all switches. Completing routing table entries signifies that all switches have a uniform understanding of the entire network topology, marking the point of convergence.
- **Standard SDN architecture for Floodlight and PEMA**: Here, convergence time is measured from network initiation until the controller has detected all network links via LLDP. The moment the controller has a complete view of the network topology through LLDP, it is considered the point of network convergence.
- **cRetor and DCell (Topology-Aware Routing)**: Since these algorithms lack explicit convergence indicators, the experiment employs a practical approach. We selected two nodes in different pods as source and destination and measured the time from network startup until the first data packet from the source was received at the destination node. This method provides a tangible measure of when the network effectively becomes operational for routing data.

The experimental results, as shown in Figure 4.6, reveal a significant advantage of cRetor in terms of network convergence time compared to traditional link-state protocols like OSPF and LLDP-based SDN solutions (Floodlight, DCell and PEMA). Notably, cRetor achieved the transmission of the first packet within just 1 second, whereas OSPF and Floodlight required several tens of seconds for topology discovery and synchronization. Although the DCell approach shows quick convergence in smaller networks, its performance becomes comparable to Floodlight in larger network settings.

When examining scalability, cRetor demonstrates remarkable stability in convergence times, even as the network size increases from $k = 4$ to $k = 8$. This stability is due to the minimal overhead involved in establishing secure tunnels between controllers and switches in cRetor. In contrast, Floodlight's convergence time increases with network size due to the additional

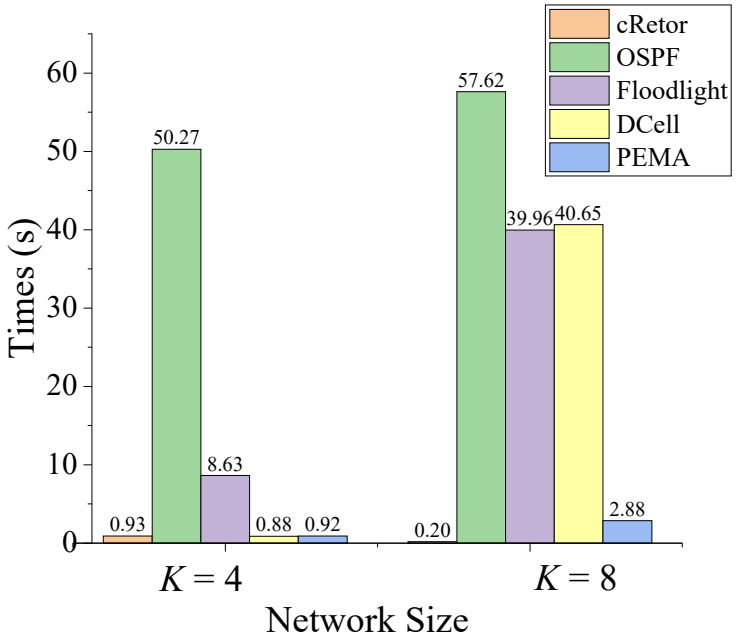


Figure 4.6: Comparison of network convergence time of cRetor, OSPF, Floodlight, PEMA and DCell in different network sizes.

steps of broadcasting LLDP messages and relaying them back to the controller. This process consumes significant control plane bandwidth, making the controller a bottleneck in large-scale networks. The PEMA algorithm, which also utilizes the standard SDN architecture, exhibits a smaller increase in convergence time compared to Floodlight. However, it also demonstrates a significant improvement due to its topology discovery mechanism. OSPF also shows increasing convergence times with network growth, owing to the necessity for switch-to-switch message exchanges, like Link State Update (LSU) and Database Description (DD) messages, during its convergence process. Therefore, compared to traditional SDN and OSPF, cRetor's ability to maintain fast convergence times regardless of network size is a significant advantage. This characteristic ensures that as data centers grow, they can maintain their agility in responding to network changes, supporting dynamic workloads and maintaining service quality.

3. First Packet Delay

This experiment focuses on the first packet delay to evaluate the response time and processing capability of SDN controllers in different routing schemes: cRetor, Floodlight, PEMA, and DCell. The first packet delay is crucial as it reflects how quickly a controller can compute a forwarding path and install flow table rules upon the arrival of a flow's first packet, with subsequent packets relying solely on the forwarding efficiency of the switches and the path

length.

We utilized three distinct traffic models—one-to-one, one-to-many, and many-to-many—to represent common intra-data center traffic scenarios. For cRetor, Floodlight and PEMA, we employed a 4-ary Fat-Tree topology with 16 server nodes. The number of flows in the one-to-one, one-to-many, and many-to-many traffic modes was set to 1, 15, and 240, respectively. In contrast, the DCell topology used for DCell experiments consisted of 20 server nodes, with 1, 19, and 380 flows for the corresponding traffic modes.

After ensuring network convergence in each setup, nodes in all networks initiated data packet transmissions simultaneously. The round-trip time (RTT) measured from Ping messages was halved to determine the one-way end-to-end delay. This measurement method clearly indicates each routing scheme's efficiency in handling initial data packet transmission, a critical factor in real-world data center network operations.

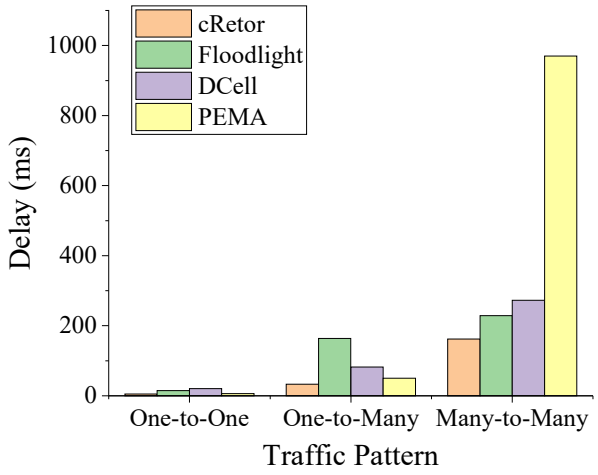


Figure 4.7: Comparison of first packet delay of cRetor, Floodlight and DCell in different traffic patterns.

The experimental results, as illustrated in Figure 4.7, clearly show that cRetor exhibits lower first packet delays across all traffic patterns when compared to other methods. The primary reason for this enhanced performance is the efficiency of the A^* algorithm, which, powered by the TPDL distance formula, calculates forwarding paths more rapidly than the traditional Dijkstra's algorithm. Additionally, cRetor's controller is relieved of the burden of topology discovery, translating to a lower baseline workload and higher throughput for processing Packet-In messages, further contributing to reduced first packet delays.

However, it is noteworthy that cRetor, like other SDN architectures, still experiences higher

first packet latency than traditional non-SDN architectures such as OSPF, BSR, and Fat-tree dual-layer routing algorithms. This is a consequence of the SDN design philosophy that separates control and data planes, inherently introducing more latency due to the interaction between the controller and switches. To address this latency inherent in SDN architectures, we propose a semi-centralized improvement scheme, which will be discussed in subsequent sections. This scheme aims to balance centralized control and distributed data forwarding, potentially reducing the latency associated with SDN architectures.

The lower first packet delays in cRetor across all traffic patterns have significant implications for data center performance, especially for latency-sensitive applications. In modern data centers, where microseconds can make a difference in application responsiveness, cRetor's ability to reduce first packet delay could translate to improved user experience and better overall application performance.

4. Control Message Overhead

We utilized the TCPDump program to capture and analyze the control message overhead for the four routing schemes. TCPDump is a network packet analyzer that records packets transmitted over a network interface, making it ideal for our analysis. We captured all control messages from the initiation of the network simulation until 80 seconds later, encompassing both the network convergence and operation phases, for a thorough assessment.

For the OSPF-based setup, we included all control messages as defined in the OSPF standard in our analysis. In contrast, for the four SDN-based schemes (cRetor, Floodlight, PEMA, and DCell), we focused on capturing all OpenFlow messages, LLDP messages, and specifically for cRetor, its Hello messages.

The results of this analysis are presented in Figures 4.8 to 4.9. These figures show the control message overhead in terms of both the number and size of packets. This offers a comprehensive view of the control plane traffic generated by each routing scheme, providing valuable insights into their efficiency and scalability.

The control message overhead analysis reveals significant differences in the messaging strategies of the routing schemes studied. DCell, for instance, consistently sends more messages than the other schemes, with a peak rate of 3000 packets per second (pps) during both the convergence and operational phases, as shown in Figure 4.8. This high message rate, primarily due to

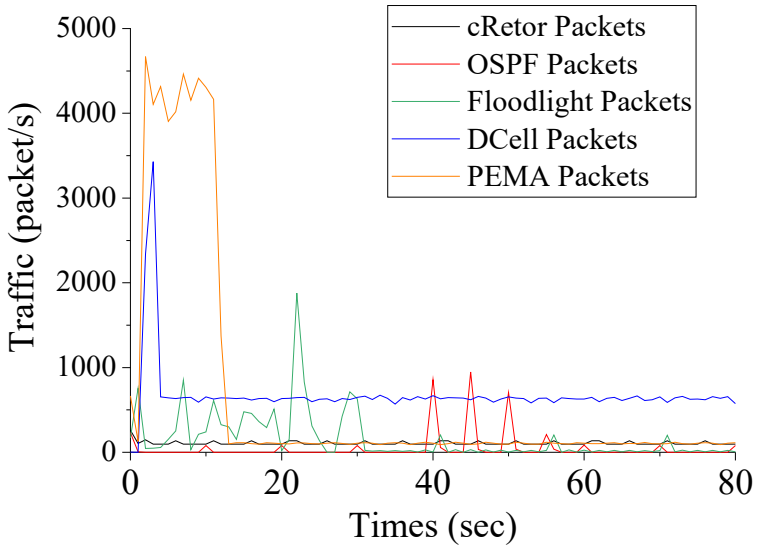


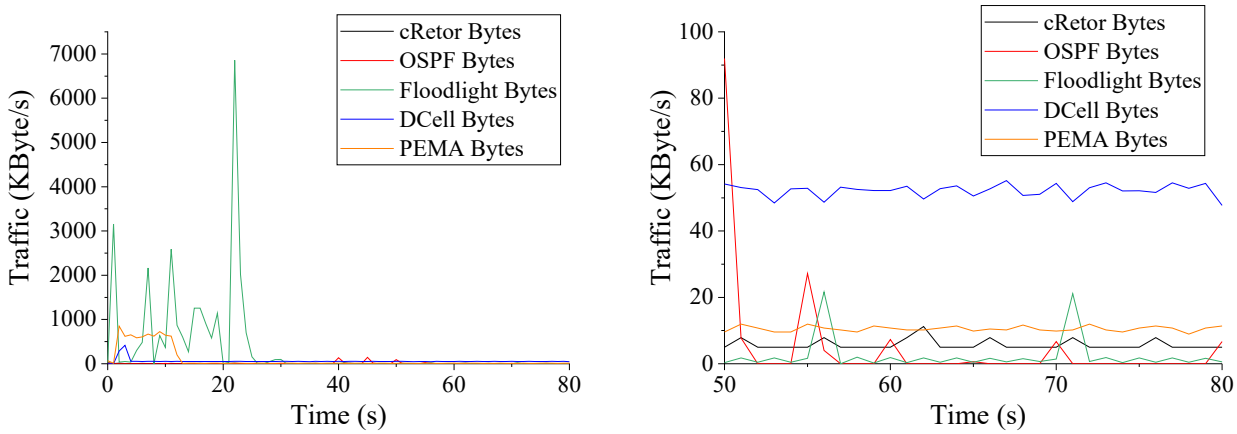
Figure 4.8: Comparison of control messages overhead in packet number of cRetor, OSPF, Floodlight and DCell.

its aggressive LLDP detection settings (1-second intervals), is designed to speed up topology discovery and fault recovery. However, it also implies a higher network load compared to other schemes, such as Floodlight, which sets a longer 15-second LLDP detection interval. The PEMA algorithm adopts a more aggressive topology discovery strategy at the start, leading to the generation of a significant number of topology discovery packets. However, once it enters the stable phase, the interval between discoveries becomes longer, resulting in fewer packets.

Both OSPF and Floodlight experience a surge in message quantity during convergence, taking several tens of seconds to reach peak rates. After convergence, they enter a stable operational state with regular topology detection messages. OSPF sends out Hello messages every 10 seconds, while Floodlight's LLDP messages are on a 15-second cycle. The control traffic generated by these detection intervals, especially in larger networks, can lead to significant fluctuations in network traffic.

cRetor's control message strategy stands out for its relative stability and efficiency. Unlike OSPF and Floodlight, cRetor maintains a consistent message rate from network initialization, primarily due to its non-convergent nature. The bulk of cRetor's messages are Hello messages, sent at a 1-second interval between switches. Although this results in a higher total number of Hello messages, the simplicity of these one-way communications and their restriction to the data plane make them less taxing on the network. They do not require replies from the receiving

peer and do not burden the control plane or the controller, thus minimizing overall network overhead.



(a) Complete Process: 0~80s.

(b) Stable Phase: Zoom to 50s~80s.

Figure 4.9: Comparison of control messages overhead in bytes of cRetor, OSPF, Floodlight, DCell and PEMA.

An examination of the control message bytes, as depicted in Figure 4.9, reveals distinct trends among the routing schemes. During the convergence phase, Floodlight's control message exchange is notably higher in bytes than the other algorithms. To facilitate a clearer comparison, Figure 4.9b zooms to 50 ~80s for a clearer view.

DCell has the highest overall overhead, primarily due to its frequent LLDP topology discovery activities. In contrast, despite also conducting high-frequency discovery, cRetor incurs significantly lower overhead compared to DCell and PEMA. This suggests that cRetor's topological discovery and fault detection approach is more byte-efficient than traditional SDN solutions like DCell. While having a lower discovery overhead overall, OSPF exhibits spikes in transmission during its link-state data exchange in the convergence process.

In summary, although cRetor generates a high number of control packets and bytes due to its frequent topology detection, it maintains a smoother and more predictable control overhead profile. This contrasts with the more variable and abrupt fluctuations observed in the control traffic of the other schemes, which can potentially introduce greater instability into the network. cRetor's ability to efficiently manage control message overhead contributes to its stability and reliability in network operations, especially in dynamic environments typical of data center networks. By reducing the control plane bandwidth consumption and eliminating the controller

bottleneck in large-scale networks, cRetor paves the way for more efficient and responsive data center networks. This could potentially lead to improved application performance, reduced latency, and better utilization of network resources in large-scale data center environments.

5. Fault Recovery Time

In this experiment, we assessed cRetor's fault recovery performance by measuring the time taken for fault recovery and the number of packets lost during the process. We selected a pair of cross-Pod nodes in the test network as the test nodes, running a client program on the source node to send UDP packets at 1-millisecond intervals, each with a unique sequence number. The destination node ran a server program, recording the arrival times and sequence numbers of received packets. We initially disconnected all but one shortest path between the source and destination to evaluate the fault recovery capability. Thus, packets initially traveled along this unique path. Upon inducing a fault, the routing algorithm had to identify an alternative, albeit longer, path.

In the Floodlight environment, we encountered a failure in fault recovery. We identified that the default 5-second idle timeout for Floodlight's flow table entries hindered the switch's ability to generate a table miss for redirecting packets along a new path when a fault occurred. This was exacerbated by the continuous transmission of UDP packets, resulting in the persistence of invalid flow table entries. However, PEMA, which also utilizes the standard SDN structure, addresses this issue by setting a mandatory expiration time for flow table entries, allowing it to recover from failures.

For OSPF, we noted a significant influence of its Fast Convergence feature on recovery times. This feature, when enabled, reduces the SPF timer from the default 5 seconds to 50 milliseconds, significantly impacting fault recovery speed. Based on Cisco's documentation [120], we conducted experiments with both settings—Fast Convergence enabled and disabled—to understand its impact on fault recovery.

The comparative analysis highlights the varying capabilities of different routing schemes in fault recovery, with particular attention to the speed and efficiency of each scheme in adapting to network changes.

As illustrated in Figure 4.10, the experimental results underscore the substantial differences in fault recovery capabilities among the routing schemes. Without fast convergence, OSPF's

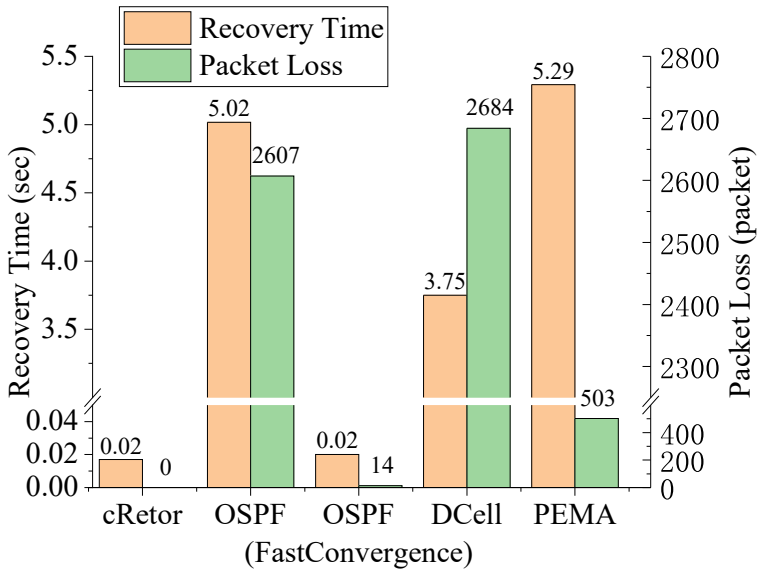


Figure 4.10: Comparison of recovery time and loss packet number of cRetor, OSPF and DCell.

recovery time exceeds 5 seconds, resulting in a significant loss of over 2,600 packets. However, enabling fast convergence dramatically enhances OSPF's performance, reducing recovery time to approximately 20 milliseconds and limiting packet loss to just 14 packets. This illustrates the profound impact of OSPF's fast convergence feature on its fault recovery efficiency.

DCell, on the other hand, takes about 3.75 seconds to reconfigure switches and establish new data paths, resulting in the loss of over 2,600 packets. In the PEMA method, data flow transmission was interrupted for about 5.29 seconds, resulting in 503 packets being lost. In contrast, cRetor stands out with its exceptional recovery time of only 17 milliseconds and no packet loss during the entire process. This rapid and seamless recovery showcases cRetor's advanced fault detection and link-switching capabilities.

These findings are particularly significant for data center networks, where stability and uninterrupted data transmission are paramount. cRetor's ability to quickly recover from faults without packet loss aligns closely with the core requirements of data center networks, offering a more stable and reliable transmission capacity compared to traditional methods. Overall, cRetor's performance in this aspect highlights its suitability for deployment in environments where rapid adaptation to network changes is critical.

Therefore, the rapid recovery capability could be particularly valuable in supporting critical applications with stringent uptime requirements. It suggests that cRetor could significantly enhance the resilience of data center networks, potentially reducing the need for redundant

hardware and complex failover mechanisms. The ability to maintain uninterrupted data transmission even in the face of network faults aligns perfectly with the core objectives of modern data centers, where continuous availability and consistent performance are paramount.

4.2 TPDL-based Semi-Centralized Data Center Network Routing Method

In the previous section, we introduced the centralized cRetor routing scheme, leveraging TPDL to notably diminish the topology discovery overhead for controllers in software-defined data center networks and enhance path calculation efficiency. However, two significant challenges remain to be addressed in this scheme. Firstly, the issue of flow establishment time or initial packet delay remains a concern. Compared to traditional routing methods like OSPF or Fat-Tree two-layer routing, cRetor, constrained by the structural limitations of SDN, exhibits higher initial packet delays. This delay is critical in data center networks, especially for latency-sensitive applications. We aim to optimize this metric, striving to reduce initial packet delays while preserving the inherent flexibility offered by SDN. The second challenge is to reduce the controller overhead. In SDN environments, the controller's role in calculating and establishing paths for each flow becomes increasingly heavy as the network scale expands. Despite the negligible per-switch overhead, the aggregated impact on the control plane is substantial. This increased workload can lead to higher processing delays, as evidenced in [107]. Consequently, traditional SDN approaches might fall short of meeting the stringent latency requirements of some latency-sensitive applications in data centers.

To address the challenges of reducing flow setup time and minimizing controller workload in data center networks, we propose an enhanced version of the cRetor routing method, sRetor. sRetor is a semi-centralized routing scheme, distinguished from its predecessor, cRetor, primarily by deploying TPDL not just on the controller but also on individual switches. This key innovation enables switches in sRetor networks to independently perceive the entire network topology and perform local distance calculations using the TPDL information. These switches are configured to acquire TPDL information during their startup phase, allowing them to subsequently operate autonomously. Given the stable fundamental architecture of data center networks, it is not necessary for sRetor switches to update TPDL when topology changes temporarily, such as those due to link failures.

sRetor's semi-centralized nature strikes a balance between centralized control and decentralized decision-making. While the switches are equipped with TPDL, enabling them to forward packets independently, they remain fundamentally OpenFlow switches. They still communicate with the sRetor controller via the OpenFlow protocol, allowing the controller to retain

ultimate control over switch forwarding behaviors. This design not only reduces the workload on the controller by offloading basic routing functions to the switches but also preserves the flexibility inherent in standard SDN architectures. Thus, sRetor effectively merges the benefits of centralized and decentralized approaches, optimizing both network efficiency and controller resource utilization.

4.2.1 Problem Model

In this section, we delve into the theoretical modeling and analysis of two critical aspects of SDN networks: packet delay and controller workload. Our objective is to theoretically substantiate the benefits of executing packet forwarding directly on switches rather than relying on controller-based routing. We will demonstrate how this approach reduces network latency and alleviates the controller's processing burden by distributing tasks to the data plane.

1. Delay Model

The delay model is crucial for understanding the latency dynamics in software-defined data center networks, particularly in the context of the cRetor architecture. Figure 4.11 illustrates the basic architecture where switches, devoid of decision-making capabilities, rely on the SDN controller for action instructions. The switches in this architecture are dummy switches without forwarding decision capabilities. They are only responsible for executing action instructions in flow table entries. The SDN controller establishes connections with each switch via in-band or out-of-band channels and makes forwarding decisions. This reliance forms the basis of our model, focusing on the delays incurred in this process.

In the delay model, the point-to-point delay $\tau(n)$ for a packet n in a flow is represented by Equation (4.2) [121]:

$$\tau(n) = t_{queue}(n) + t_{proc}(n) + t_{trans}(n) + t_{prop}(n) \tag{4.2}$$

Here, $t_{queue}(n)$ is the queuing delay at the switch, influenced by factors like network congestion and buffer strategies. The transmission delay, $t_{trans}(n)$, depends on packet size and data rate, while the propagation delay, $t_{prop}(n)$, is affected by physical distances and transmission mediums. Our primary focus is on the processing delay $t_{proc}(n)$, particularly significant in SDN architectures where controller-induced processing might lead to bottlenecks.

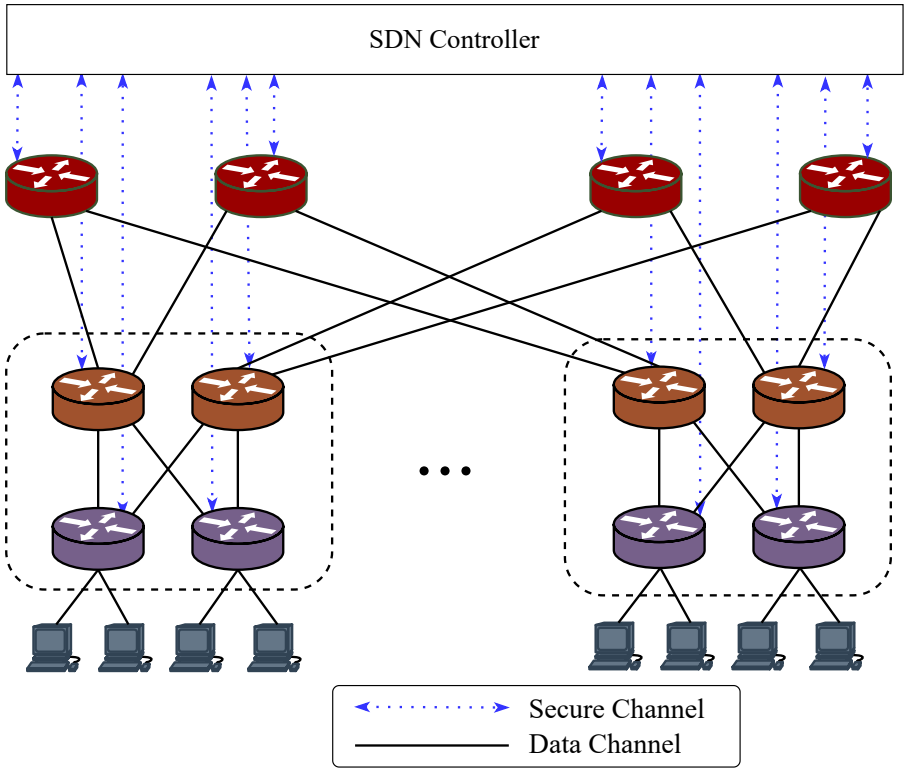


Figure 4.11: The architecture of software-defiend data center networks.

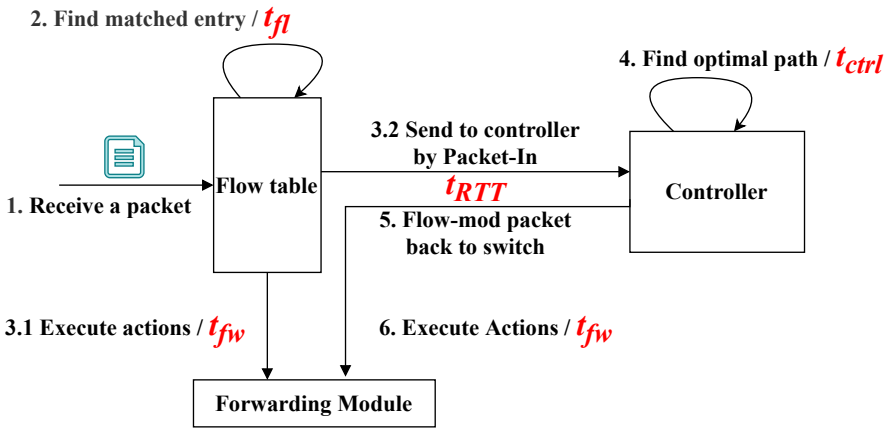


Figure 4.12: The processing delay in cRetor (traditional SDN) switches.

In traditional SDN schemes, as detailed in Jia et al.[88] and Darabseh et al.[122], packet processing delay involves a series of steps, visually represented in Figure 4.12. This process is broken down as follows:

- Step 1: Packet reception at the switch's input port.
- Step 2: Flow table lookup in the switch for a matching entry, with delay t_{fl}. Factors affecting this delay include the size and complexity of the flow table.
- Step 3:

  - Step 3.1: On finding a match, the switch executes the specified action (typically forwarding), incurring a delay t_{fw}. This delay can be influenced by switch processing speed and action complexity.
  - Step 3.2: If no match is found (Table-Miss), the switch sends a Packet-In message to the controller, taking half of the round trip time $t_{RTT}/2$.
- Step 4: The controller calculates the forwarding path, with computation delay t_{ctrl}, which depends on controller processing power and current workload.
- Step 5: Flow-Mod message transmission from the controller to the switch, also taking $t_{RTT}/2$.
- Step 6: Execution of the action command in the new flow entry by the switch, again with delay t_{fw}.

Cumulatively, these steps contribute to the total packet forwarding time in an SDN network, impacting key performance metrics like latency and throughput. Understanding these delays is crucial for assessing network efficiency and identifying potential bottlenecks.

Let $T = t_{RTT} + t_{ctrl}$ represent the total communication delay between the switch and the controller, essentially the waiting time for a data packet at the switch. We define the total processing delay as in Equation (4.3), based on Lin et al.[123]:

$$
\begin{aligned}
t_{\text{proc}}(n) &= t_{fl} + t_{fw} + I_\alpha(n) \cdot T \\
I_\alpha(n) &= \begin{cases} 0, & \text{if packet } n \text{ matches a flow table entry,} \\ 1, & \text{otherwise.} \end{cases}
\end{aligned} \tag{4.3}
$$

In this equation, t_{fl} represents the delay due to flow table lookup, and t_{fw} is the delay incurred for executing the forwarding action. The indicator function $I_\alpha(n)$ plays a critical role in distinguishing whether packet n finds a matching entry in the flow table ($I_\alpha(n) = 0$) or not ($I_\alpha(n) = 1$). When $I_\alpha(n) = 1$, indicating no match, the packet must be sent to the controller, adding the waiting time T to the total processing delay.

The scenarios triggering $I_\alpha(n) = 1$, thus necessitating controller intervention, typically include:

- The arrival of the first packet of a new data flow, which lacks a corresponding entry in the flow table.

- Failure of the next-hop node as specified in a flow entry, causing the entry to be invalid due to network faults.
- Deletion of flow entries for reasons such as table overflow or entry expiration.

Each of these scenarios can significantly impact network latency and controller workload. The first packet of a new flow or changes in network topology (due to faults or other factors) require controller involvement, potentially increasing delay and workload.

We define $t = 0$ as the moment when the controller receives the first Packet-In message containing a data packet. Considering subsequent data packets that arrive before the switch receives a response from the controller, i.e., their arrival times $t_n \in (0, T]$, the processing delay for packet n can be represented as:

$$\begin{aligned} t_{\text{proc}}(n) &= t_{fl} + t_{fw} + I_\alpha(n) \cdot (T - t_n) \\ I_\alpha(n) &= \begin{cases} 0, & \text{if packet } n \text{ matches a flow table entry} \\ 1, & \text{otherwise} \end{cases} \end{aligned} \tag{4.4}$$

We define $t = 0$ as the moment when the controller receives the first Packet-In message containing a data packet. When subsequent data packets arrive before the switch receives a response from the controller, with their arrival times denoted as $t_n \in (0, T]$, the processing delay for packet n can be represented as:

$$F_{t_n}(t) = 1 - \sum_{i=0}^{N(T)-1} \frac{(\lambda t)^i}{i!} e^{-\lambda t} \tag{4.5}$$

Where $N(T)$ is the total number of these subsequent arriving packets between 0 and T. Therefore, we have the CDF of t_{tw} as shown in equation (4.6).

$$F_{t_{wt}}(t) = 1 - F_{t_n}(T - t) = \sum_{i=0}^{N(T)-1} \frac{(\lambda(T-t))^i}{i!} e^{-\lambda(T-t)} \tag{4.6}$$

The expectation of processing delay $E(t_{proc})$, as shown in Equation (4.7), incorporates the probability of a flow table hit p_{hit} and the effect of packet arrivals within the waiting time window.

$$E(t_{\text{proc}}) = t_{fl} + t_{fw} + (1 - p_{hit}) \cdot T \cdot \left(1 - \frac{N(T)}{\lambda}\right) \tag{4.7}$$

Our objective is to minimize the expected waiting time, as defined by the following optimization problem:

$$\begin{aligned} \min \quad & \sum_{i=0}^{N(T)-1} \frac{(\lambda(T-t))^i}{i!} e^{-\lambda(T-t)} \\ \text{s.t.} \quad & \forall 0 < t \leq T \end{aligned} \tag{4.8}$$

By reducing $F_{t_{wt}}(t)$, we aim to decrease the waiting time for packets, thus enhancing the overall efficiency of the network. This theoretical framework provides valuable insights into optimizing packet processing in SDN networks, particularly in scenarios where reducing controller workload and improving response times are critical, as in the case of sRetor.

In data center networks with fixed topologies, reducing the total transmission time, denoted as T, poses significant challenges. To mitigate this, we propose to minimize the total processing time, t_{proc}. We categorize the forwarding paths computed by the controller into two types: 1) paths including the current node and subsequent nodes, and 2) entirely new paths bypassing the current node. The probability of the former is typically greater than the latter, as adopting a new path necessitates retransmitting all data packets directed to the current node. To decrease t_{proc}, our strategy involves making the first type of forwarding decisions at the current node itself, thereby identifying subsequent paths for data packets starting from the current node. This approach aims to circumvent the round-trip time delay, t_{RTT}, incurred through controller interactions via the secure channel of SDN.

Local routing decisions in SDN switches require that these switches possess information about candidate neighboring and destination nodes. Traditional SDN switches, often termed dummy switches, lack a control plane, making them incapable of gathering and analyzing topological data or making autonomous decisions. To overcome this limitation, we propose to integrate TPDL into SDN switches. This integration enables the switches to perform local distance calculations, thereby facilitating local routing decision-making.

The processing delay in the proposed sRetor method, as depicted in Figure 4.13, differs from that of cRetor primarily due to the addition of a TPDL forwarding step. This step is inserted between Steps 3.2 and 5.2 of the process. For a packet n with the indicator $I_\alpha(n) = 1$, sRetor will not immediately send it to the controller. Instead, this packet will first be sent to the TPDL calculator to attempt to compute an available next-hop node via TPDL locally. The packet is

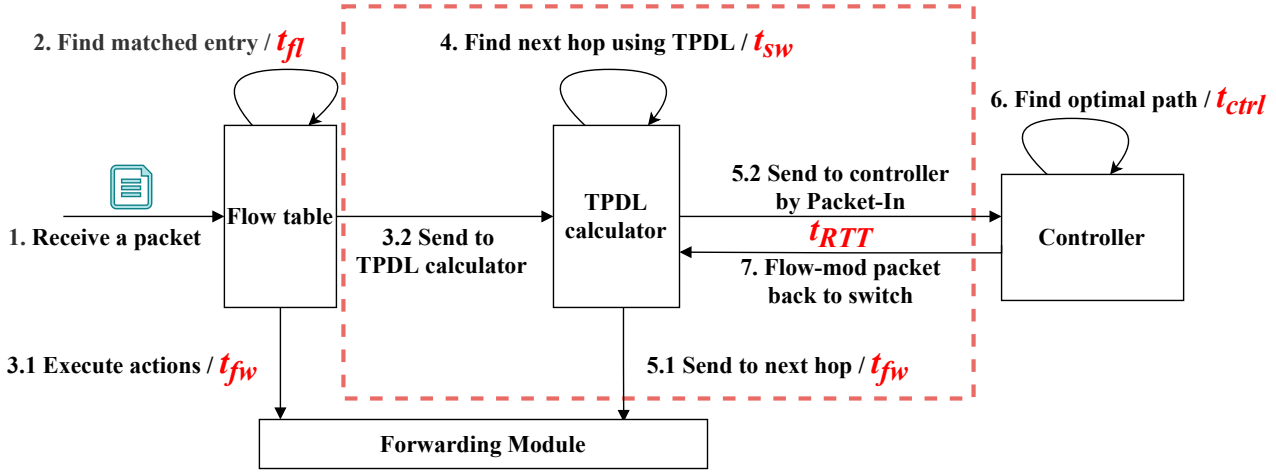


Figure 4.13: The processing delay in sRetor switches.

forwarded to the controller for the final forwarding decision only if the TPDL calculation fails.

We now perform a theoretical analysis of the latency involved in the forwarding process employed by sRetor. Within this context, let t'_{proc} represent the processing delay encountered by a given packet n in the sRetor framework. Since the integration of the TPDL forwarding step within the processing sequence, the expression for t'_{proc}, along with its expected value, are formulated in Equations (4.9) and (4.10) respectively.

$$\begin{aligned} t'_{proc}(n) &= t_{fl} + t_{fw} + I_\alpha(n) \cdot (t_{sw} + I_\beta(n) \cdot (T - t_n)) \\ I_\beta &= \begin{cases} 0, & \text{if the next hop node can be found locally,} \\ 1, & \text{otherwise.} \end{cases} \end{aligned} \tag{4.9}$$

$$E(t'_{proc}) = t_{fl} + t_{fw} + (1 - p_{hit}) \cdot \left(t_{sw} + (1 - p_{sw}) \cdot T \cdot \left(1 - \frac{N(T)}{\lambda} \right) \right) \tag{4.10}$$

Here, $p_{sw} = P(I_\beta(n) = 0)$, which is the probability of achieving fault-aware local forwarding. Correspondingly, the CDF of the waiting time t'_{sw} is as follows:

$$F_{t'_{wt}}(t) = p_{sw} + (1 - p_{sw}) \cdot \sum_{i=1}^{N(T)-1} \frac{(\lambda(T-t))^i}{i!} e^{-\lambda(T-t)} \tag{4.11}$$

To demonstrate that sRetor achieves lower latency compared to conventional SDN methodologies, it is crucial to ascertain that the difference in latency distributions, denoted as $\Delta P(t)$, is

positive. Mathematically, this requirement is represented as follows:

$$\Delta P(t) = F_{t'_{\text{wt}}}(t) - F_{t_{\text{wt}}}(t) > 0 \tag{4.12}$$

Where $F_{t'_{\text{wt}}}(t)$ and $F_{t_{\text{wt}}}(t)$ denote the latency distribution functions for sRetor and traditional SDN, respectively.

Building upon the foundational concepts outlined in Equation (4.6) for traditional SDN and Equation (4.11) for sRetor, we proceed to derive the specific formula for $\Delta P(t)$ as follows:

$$\Delta P(t) = p_{sw} \cdot \left(1 - \sum_{i=0}^{N(T)-1} \frac{(\lambda(T-t))^i}{i!} e^{-\lambda(T-t)}\right) \tag{4.13}$$

The condition $p_{sw} > 0$ clearly leads to the scenario where $\Delta P(t) \geq 0$. This inequality indicates that, under circumstances where the probability of successful switch-based processing is non-zero, the latency difference between sRetor and traditional SDN solutions is either positive or null. Consequently, it can be inferred that our proposed scheme is capable of achieving shorter delays in comparison to conventional SDN methodologies, particularly when switch-based processing successfully occurs with a probability greater than zero.

Our primary goal is to increase p_{sw}, the probability of successful switch-based processing, for better delay performance. In the TPDL-based local path calculation algorithm, assuming a fault-free environment, p_{sw} can potentially reach 1. This is because the shortest path to the target node can always be found in the original connected network topology. However, the selected next-hop node may become unreachable due to potential network failures. Our algorithm incorporates a mechanism to exclude failed nodes when selecting the next hop. In sRetor, the detection of a failed neighbor node is achieved by monitoring the expiration time (dead interval) of Hello messages. This expiration time is typically set to ϵ times the Hello interval. If a switch fails to receive a Hello packet from a neighbor within this expiration period, that neighbor will be marked as failed. Adjusting the expiration interval has implications for network performance. Specifically, a longer interval increases the pool of candidate nodes, thereby raising p_{sw}. However, this comes at the cost of potentially lower path-forwarding success rates, as it may include recently failed nodes. Balancing p_{sw} with forwarding success rate is crucial; thus, ϵ is often set to values like 3 or 4 [124].

Additionally, t_{sw}, the processing delay of the TPDL calculator, is also important. A lower

t_{sw} directly contributes to reduced overall forwarding delays. The TPDL algorithm, which computes paths using a distance formula, has a time complexity of $O(m)$, where m is the number of neighbor nodes. In fixed-topology networks like Fat-tree, where the number of neighbor nodes for each switch is defined by its k value, m is bounded, thereby limiting t_{sw}.

2. Controller Workload Model

In the realm of SDN, the centralized controller's role is significant, encompassing the management of all OpenFlow messages from the switches. Among these, the Packet-In message stands out as a particularly significant type of OpenFlow message. A switch sends a Packet-In message to the controller when it encounters a packet that it cannot forward by itself. This scenario is commonplace in traditional SDN networks.

The processing of Packet-In messages imposes a considerable demand on computational resources and control plane bandwidth. Kotani et al. [125] highlight the intensity of these requirements. Acknowledging this, the current section is dedicated to modeling and analyzing the controller's workload, particularly focusing on the generation probability of Packet-In messages. Through this model, we aim to quantify the impact of these messages on controllers in both cRetor and sRetor.

As mentioned in the previous section, Packet-In messages in traditional SDN schemes are typically generated under the condition $I_\alpha(n) = 1$. We primarily focus on two scenarios where this condition is met: 1) when the data packet n is the initial packet in a flow, and 2) when a link failure occurs along the packet's forwarding path. Consequently, the probability of packet n being forwarded to the controller as a Packet-In message by the switch is given by Eq. (4.14):

$$p_{pkt\_in}(n) = p_{1st}(n) + (1 - p_{1st}(n)) \cdot (1 - (1 - q)^m) \tag{4.14}$$

Here, $p_{1st}(n)$ denotes the probability of n being the first data packet in its respective flow. The variable q represents the link failure rate, while m signifies the length of the forwarding path.

In the sRetor scheme, Packet-In messages are generated exclusively when all next-hop nodes of a switch encounter failures. This scenario leads to a distinct probability P'_{pkt_in} for the generation of Packet-In messages in sRetor, as delineated in Eq. (4.15):

$$p'_{pkt\_in}(n) = p_{1st}(n) + (1 - p_{1st}(n)) \cdot \left(1 - \prod_{i=0}^{m} (1 - q^{c_i})\right) \tag{4.15}$$

In this equation, c_i represents the count of candidate next-hop neighbor nodes at the same distance from the destination node.

The difference in Packet-In message probabilities between sRetor and traditional SDN schemes is given by:

$$\begin{aligned} \Delta p_{pkt\_in} &= p_{pkt\_in}(n) - p'_{pkt\_in}(n) \\ &= \prod_{i=0}^{m}(1 - q^{c_i}) - (1 - q)^m \geq 0, \\ &\text{where, } 0 < q < 1 \text{ and } c_i \geq 1 \end{aligned} \tag{4.16}$$

Thus, it follows that $p'_{pkt\_in}(n) \leq p_{pkt\_in}(n)$. This implies that the controller in the sRetor scheme is burdened with fewer Packet-In messages compared to traditional SDN solutions such as cRetor. Consequently, sRetor is more adept at supporting larger-scale software-defined data center networks by reducing the controller's workload.

4.2.2 sRetor Architecture

This section introduces the comprehensive system architecture and the various components of sRetor. The primary design objective of sRetor is to empower SDN switches within software-defined data center networks with basic autonomous forwarding capabilities. A significant advantage of this design is the elimination of reliance on a centralized controller for routine forwarding tasks, leading to a notable reduction in flow setup times across the network.

The proposal of sRetor brings about a two-fold benefit. Firstly, it substantially alleviates the workload on the sRetor controller. By reducing its involvement in fundamental forwarding decisions, the overall efficiency and responsiveness of the controller are enhanced. Secondly, this reduction in workload enables the controller to allocate more resources towards advanced and critical network functions. These include, but are not limited to, load balancing and QoS management. Such a shift in resource allocation allows for a more dynamic and effective network management strategy, directly contributing to improved network performance and reliability.

The system architecture of sRetor, as depicted in Figure 4.14, builds upon the foundational principles of SDN. This architecture contains both the controller and the network switches, which maintain communication through an extended version of the OpenFlow protocol. The sRetor controller shares a structural resemblance with that of the cRetor. It is responsible for monitoring the real-time status of the entire network and handling fault information reported

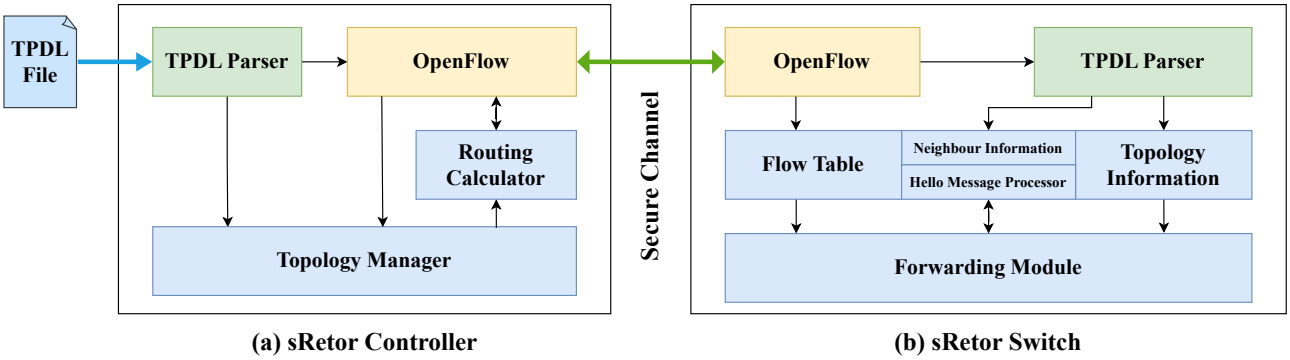


Figure 4.14: The system architecture of the sRetor controller and switches.

by the switches. In scenarios where the switches are incapable of independently forwarding packets due to complex network failures, the controller will identify and establish alternative forwarding paths for the affected flows, thereby ensuring uninterrupted network service. During the initialization phase, the controller will transmit a TPDL file to the switches via the OpenFlow channel. This file equips the switches with essential information, enabling them to make more informed, autonomous decisions about packet forwarding in the following running stage.

Once a switch in the sRetor network receives the TPDL file from the controller, its initial action is to parse it using a dedicated TPDL parser. This parsing is crucial for equipping the switch with the necessary data for subsequent distance calculations. As shown in the figure, the switch's forwarding decision-making process integrates inputs from three key components: the flow table, the neighbor information table, and the topology information.

The flow table, populated with entries from the controller, retains the highest priority in the forwarding decision process. This prioritization ensures a level of flexible control similar to that in traditional SDN networks. Regarding neighbor information, the switch relies on both static and dynamic sources. The static data originates from the TPDL file, while the dynamic data is collected from handling Hello messages. The Hello message processor plays a vital role in monitoring the real-time connectivity status with neighboring nodes and updating the neighbor information table accordingly to reflect any changes. Topology information, directly extracted from the TPDL file, empowers the forwarding module with efficient distance calculation capabilities. This information is essential to enable the switch to make reasonable decisions about the most efficient paths for data packet forwarding.

The details of the forwarding process, which leverages these components, will be elaborated on in

the following section. This detailed discussion will provide a more explicit algorithm description of how the sRetor improves packet forwarding in software-defined data center networks.

This section will introduce the intricate design and practical implementation of the routing algorithm tailored for the switches within sRetor. This algorithm is a cornerstone of the sRetor system, as it facilitates not only efficient routing but also integrates advanced functionalities like load balancing and fault recovery at the switch level.

Our focus is on crafting an algorithm that is robust in handling the dynamic nature of software-defined data center networks and agile in adapting to changing network conditions. The algorithm is designed to manage network traffic seamlessly, ensuring optimal path selection for data packets under normal circumstances. Additionally, it incorporates a sophisticated load-balancing mechanism that distributes network traffic evenly across available paths, thereby preventing any single link or node from becoming a bottleneck.

Furthermore, sRetor includes a comprehensive fault recovery scheme. This scheme is designed to detect and respond to network failures rapidly, choosing an alternative forwarding path to maintain network performance and minimize disruption. The combination of these features in the routing algorithm enhances the autonomy and efficiency of SDN switches in sRetor, moving towards a more resilient and self-sufficient network infrastructure.

The following subsections detail the specific components and logic of the routing algorithm, illustrating how it achieves these objectives and contributes to the overall efficacy of the sRetor system.

1. Packet Routing Process

Figure 4.13 illustrates the fundamental forwarding process implemented by switches in the sRetor scheme. The sequence for processing packets within a switch is methodically structured: initially, there is a flow table lookup, followed by a TPDL calculation, and finally, if necessary, sending Packet-In messages to the controller. Importantly, this process is designed so that the switch immediately executes the corresponding forwarding action as soon as a match is found at any stage. This processing pipeline upholds a critical operational principle: the flow table retains the highest priority in packet processing. It ensures that, despite the autonomous capabilities imparted to the switches by sRetor, the controller maintains comprehensive oversight and control over the switch's operations as required.

To significantly enhance forwarding efficiency in the sRetor scheme, the TPDL route calculator is designed to compute forwarding paths and write these computed paths into the flow table. This innovative use of the flow table's unused space effectively transforms it into a high-speed cache for route calculations. By doing so, it eliminates the need for redundant computations, thereby streamlining the routing process.

When a switch needs to forward a packet, its first action is to search the flow table for any matched entries. These entries might be configurations installed by the controller or cached entries from previous route calculations performed by the TPDL calculator. If the switch finds a matched entry, it can immediately use this information for packet forwarding, bypassing the need for further computation. This process ensures that the routing computation is only invoked when necessary, significantly reducing the frequency of these calculations and thus improving the overall forwarding speed.

The primary objective of conducting path calculations directly on the switch is determining the next-hop node for each data packet, as outlined in Algorithm 2. In this context, distance is utilized as the principal metric for routing decisions. It is important to note that sRetor intentionally avoids imposing high-load tasks on switches. Gathering comprehensive network performance metrics, such as bandwidth and end-to-end latency, generally demands substantial resources. Consequently, these metrics are not incorporated into the current version of the algorithm. However, the algorithm is inherently flexible and compatible with other low-cost metrics. This compatibility opens avenues for future enhancements and optimizations, allowing for the potential integration of additional metrics that could further refine the routing process without significantly increasing the computational load on the switches.

When a data packet needs to be forwarded from a source node n_{src} to a destination node n_{dst} through the current node n_{cur}, the TPDL routing calculator of n_{cur} initiates the process by examining the set of its available neighbor nodes. This set is initially derived from the TPDL file. However, it undergoes a filtering process where any unreachable neighbor nodes are excluded, as indicated by the non-reception of Hello messages within the expected interval. The result is an updated set of accessible neighbor nodes.

For each eligible neighbor node n_κ in the set $K(n_{cur}) \setminus n_{prev}$, where n_{prev} is the previous node in the path, the TPDL routing calculator applies the TPDL distance formula (detailed in Algorithm 1) to compute the distance D_κ from n_κ to the destination n_{dst}. Among these neighbor

Algorithm 2: Switch Path Calculation Algorithm

Input:

n_{cur}: Current node;

n_{dst}: Destination node;

n_{prev}: Previous node for the current packet;

$K(\text{Cur})$: List of neighbor nodes of the current node;

hello_interval: Hello message interval;

t_{now}: Current timestamp;

$L(n)$: Timestamp of the last Hello message from node n

Output: Next hop node for the current packet

```
for each neighbor node n_κ ∈ (K(n_cur) \ n_prev) do
    if t_now − L(n_κ) < ε · hello_interval then
        D_κ ← tpdl_distance(n_κ, n_dst)
    end
end
Find any n* such that D_n* = min(D_n);
D_cur ← tpdl_distance(Cur, Dst);
if D_n* < D_cur then
    return n*
else
    return φ
end
```

nodes, the algorithm identifies the node n^* with the shortest distance to n_{dst}, such that $D_{n^*} = \min(D_n)$.

The algorithm then proceeds to compare the distance from the current node n_{cur} to the destination D_{cur} with the calculated distance D_{n^*}. If $D_{n^*} < D_{cur}$, this implies that the next hop node n^* is closer to the destination n_{dst}, and thus, forwarding the packet to n^* is a viable option. Conversely, suppose D_{n^*} is not less than D_{cur}. In that case, it indicates that the neighbor node is either equidistant or farther from the destination compared to the current node, making forwarding to this node unproductive. This latter scenario typically arises when all originally feasible paths are disrupted due to failures. In such cases, the switch is programmed to notify

the controller, which then takes on the responsibility of reselecting a feasible path for effective packet forwarding.

The algorithm will automatically avoid simple faults to increase its reliability. This capability is shown in Line 2 of the algorithm, where a critical check is performed regarding the status of each neighbor node. Specifically, the algorithm assesses the time elapsed since receiving the last Hello message from a neighbor node. If this duration exceeds a predetermined threshold, calculated as $\epsilon \times hello\_interval$, it signifies a potential failure of that neighbor node. This node is excluded from the set of candidate nodes for routing computations. The detailed process of fault recovery will be discussed thoroughly later.

2. Load Balancing in Switches

The inherent regularity and redundancy in the topological structures of data center networks present an opportunity for implementing robust load-balancing algorithms, which are crucial for achieving high throughput. In sRetor, switches are equipped to perform load balancing at two distinct levels: packet level and flow level.

The implementation is characterized by its simplicity and adaptability for packet-level load balancing. When a switch is required to determine the next hop for a packet, it begins by identifying all the available neighbor nodes that are closest to the destination node. This identification forms the basis for the load-balancing decision. Once this set of potential next-hop nodes is determined, the switch applies a load-balancing strategy, such as round-robin or least load, to select the most appropriate node from this set. By incorporating these strategies, the packet-level load balancing scheme in sRetor ensures that network traffic is not concentrated on a single interface or path. Instead, packets are distributed more evenly across various interfaces, contributing to a balanced network load.

Flow-level load balancing in SDN represents a more sophisticated approach compared to packet-level balancing, primarily due to the necessity for the switch to track the next-hop node for each distinct flow. Similar to packet-level load balancing, when the first packet of a flow arrives at the switch, the switch selects a suitable next-hop node. Upon this selection, the switch leverages the available space within its flow table to create a record for this specific flow. This record includes the chosen next-hop node for the flow, effectively enabling the switch to remember the designated forwarding interface for each unique flow. This mechanism is illustrated in Step 2

of Figure 4.15. Here, the switch generates a new flow table entry, uniquely identified by the flow's characteristics and the destination port. This entry is then inserted into the flow table. For subsequent packets belonging to the same flow, the switch refers to this flow table entry for direct forwarding.

Such a methodology ensures that all packets from a single flow consistently follow the same path towards their destination. This consistency is vital for preventing issues like packet reordering or disorder, which can be inefficient in scenarios requiring strict sequence preservation, such as video streaming or real-time data transmission. Thus, flow-level load balancing in sRetor contributes to efficient network utilization and upholds the integrity of data flows.

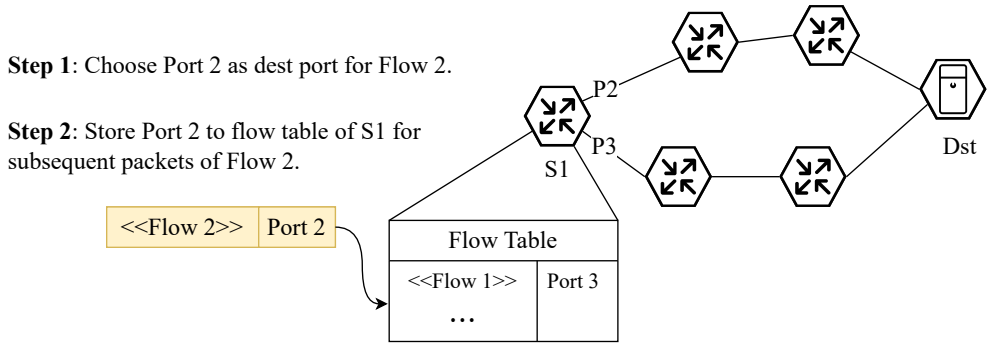


Figure 4.15: Flow-level load-balancing in sRetor.

In sRetor, the load balancing executed by switches is inherently based on local information, which, by nature, has a more limited scope compared to the comprehensive, global network view accessible to the controller. Consequently, while switch-based load balancing is effective for immediate and localized decision-making, it may not match the efficacy of a global load-balancing strategy directed by the controller, which benefits from a holistic understanding of the network's state.

However, the workload in the sRetor controller has been further reduced by delegating routine forwarding tasks to the switches; the controller is unburdened from constant, low-level decision-making. This freed-up capacity enables the controller to concentrate on more critical, overarching tasks such as monitoring for hotspot traffic congestion.

The controller can intervene with global traffic optimization strategies when such hotspots or potential congestion points are detected. This might include rerouting traffic, adjusting schedules, or implementing other measures that require a network-wide perspective. The synergy between the switches' local load balancing and the controller's global traffic optimization ef-

forts allows the network to operate with enhanced efficiency and reliability. In essence, sRetor's design capitalizes on the strengths of both local and global perspectives, harnessing them in a complementary manner to ensure smooth and effective network operation.

3. Failure Recovery Mechanism

In the semi-centralized control architecture of sRetor, both the switches and the controllers are endowed with the capabilities to make forwarding decisions and manage failures. This dual-capability setup ensures a more resilient and adaptable network. Within this framework, switches are primarily responsible for addressing simple, local failures, utilizing their immediate operational context. On the other hand, controllers equipped with a global view of the network are designated to handle more complex, multi-point failures, where a broader perspective is essential for effective resolution.

When confronting a link failure between a switch and its neighboring nodes, the nature of the failure can generally be categorized into two distinct types:

- **One link in the shortest path is broken, but alternative shortest paths exist:** This scenario is quite prevalent in data center networks, especially in structures like the Fat-Tree topology, where multiple equivalent paths are the norm. When one of these paths fails, it leads to the situation described. In such cases, the switch is typically able to identify an alternate nearest neighbor node n^* to the destination node, where $D_{n^*} < D_{cur}$. This capability allows the switch to switch links rapidly without the controller's intervention. Nevertheless, it remains crucial for the controller to be informed of this failure via the switch's report. This enables the controller to maintain an accurate view of the network topology. If the controller determines that this failure impacts traffic balance, it can undertake traffic engineering strategies for optimization.

- **All shortest paths are broken:** In this more severe scenario, no viable next-hop nodes remain along any of the shortest paths, leaving the switch unable to find an available neighbor node n^*, where $D_{n^*} < D_{\text{cur}}$. Though rare under normal circumstances, this situation implies that the source node has become unreachable to the destination node or that other viable paths in the network do not include the current switch. These cases require an assessment based on the global network topology, beyond the scope of the current switch's local information. Therefore, the switch should stop local forwarding

and send the packet to the controller. Depending on the network's overall status, the controller will then either reroute the packet along other optimal paths or, if necessary, discard the packet.

The fault recovery mechanism in sRetor is illustrated in lines 6–12 of Algorithm 2. This mechanism plays an important role in the forwarding process of each switch. Specifically, it doesn't immediately forward the packet after the switch calculates the distance to the destination node from the closest neighbor node D_{n^*}. Instead, the switch first compares D_{n^*} with its own distance to the destination D_{cur}. Such a comparison is crucial for avoiding forwarding packets on unnecessarily long or indirect paths in the event of a fault. This mechanism ensures that each forwarding decision the switch takes progressively moves the packet closer to its destination. It effectively prevents inefficient and potentially problematic scenarios, such as cyclic forwarding, where a packet might loop between the same set of nodes without moving toward its destination. As this fault handling mechanism is built into the sRetor switches, the response to a fault is rapid and fast. This rapid response has minimal perceptible impact on higher-level applications, ensuring a seamless experience for end-users.

For more complex failure scenarios that are beyond the scope of the switches' autonomous resolution capabilities, the sRetor controller steps in. Utilizing the global network view provided by SDN, the sRetor controller is equipped to manage concurrent multi-point failures and identify globally optimal paths for rerouting traffic. In scenarios where the controller intervenes in failure handling, the fault recovery time in sRetor will be equivalent to that in cRetor. However, as indicated by Eq. (4.15), the likelihood of encountering such complex failures is relatively low, thus minimizing their overall impact on the network's performance.

4.2.3 Numerical Simulation Results

We have undertaken comprehensive numerical simulations to validate the accuracy and reliability of the theoretical model proposed in the previous sections. These simulations are specifically designed to focus on two aspects of the sRetor system: the waiting time of data packets and the workload imposed on the controller.

1. Packet Waiting Time

In this experimental setup, we conducted numerical simulations in the theoretical framework established by Eq. (4.6) for traditional SDN and Eq. (4.11) for the sRetor system. The parameters used in these simulations are listed in Table 4.2 in detail. First, we modeled the arrival of data packets using a Poisson point process, which is a widely recognized method for representing packet arrival events. Then our simulation program strictly replicated the processing of these packets within the network. This involves simulating the processing delays incurred by packets at both the switch and the controller levels, as well as the mechanisms managing the waiting lists at these nodes.

Table 4.2: Simulation parameters on packet waiting time.

Parameter	**Value**
λ	2000
P_{sw}	85%
t_{prop}	300 μs
t_{sw}	100 μs
t_{ctrl}	100 μs
Bandwidth	1Gbps

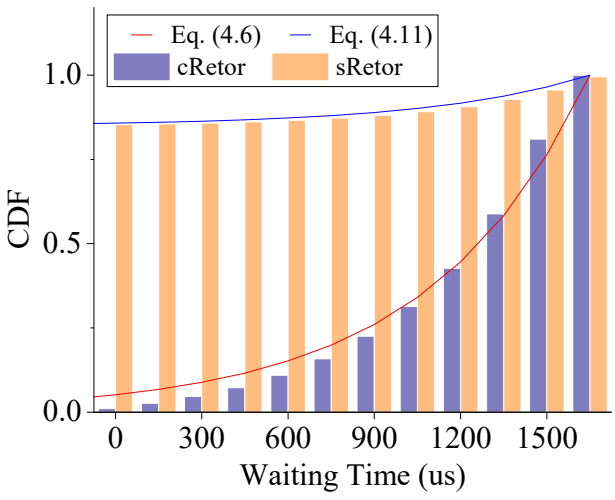


Figure 4.16: CDF of packet waiting time in numerical simulation.

Figure 4.16 shows the cumulative distribution function (CDF) of packet waiting times as derived from our numerical simulations. The results are particularly revealing when listed with the analytical models expressed in Eq. (4.6) and Eq. (4.11). Notably, the histogram generated from the simulation aligns closely with these theoretical predictions, lending empirical support

to our models.

The data delineates an obvious difference in packet waiting times between sRetor and cRetor. Specifically, it was observed that in sRetor, over 85% of packets experienced waiting times within the 0-150 μs range. This contrasts the cRetor system, where only about 3% of packets fell within the same waiting time window. The results effectively represent the probability distribution of packet waiting times, with higher CDF values at a given time point indicating a greater frequency of packets experiencing waiting times up to that mark. Consequently, the markedly higher CDF values for sRetor in the 0-150μs range compared to cRetor clearly illustrate sRetor's superior performance in this metric. In contrast, a substantial portion of the packet waiting times in cRetor are spread across a broader range of 150-1750μs, suggesting longer delays.

These findings from the simulation empirically validate our previous analysis that posited the benefits of local forwarding on switches in reducing packet waiting times. This reduction is a marginal improvement and a significant enhancement in forwarding performance. By effectively lowering packet waiting times, sRetor demonstrates its potential to significantly boost the efficiency and responsiveness of software-defined data center networks compared to conventional methods like cRetor.

2. Probability of Packet-In Messages

We also extend the scope of our numerical simulations to the probability of generating Packet-In messages under various link failure rates. These simulations aim to quantify the frequency with which packets are sent from the switches to the controller, a factor that directly impacts the controller's workload and, by extension, the overall network efficiency. The simulation parameters used are shown in Table 4.3.

Table 4.3: Simulation parameters on Packet-In message probability.

Parameter	**Value**
Topology	16-ary Fat-tree
P_{1st}	3%
q	$1\% \sim 10\%$
m	6
c_i	[1, 8, 8, 1, 1, 1]

These parameters have been carefully selected to create a realistic and comprehensive simulation environment. Each parameter plays a crucial role in accurately modeling the behavior of both sRetor and cRetor systems under various network conditions:

- Topology (16-*ary* Fat-tree): The choice of a 16-*ary* fat-tree topology is significant as fat-tree is widely used in modern data center networks.
- P_{1st} (3%): This parameter represents the probability of the first-hop switch failing to find a matching flow entry. The relatively low value of 3% was chosen to reflect the ratio of the number of first packet of a new flow to the number of general packets.
- q ($1\% \sim 10\%$): The link failure rate is a critical parameter in our simulation. By varying q from 1% to 10%, we can examine system performance under a wide range of network conditions, from near-ideal to highly stressed. This range allows us to comprehensively evaluate how sRetor and cRetor systems respond to different levels of network instability, which is crucial for assessing their robustness and adaptability.
- m (6): This parameter denotes the number of hops in the longest path within our simulated network. The value of 6 is typical for a fat-tree topology and enables us to examine multi-hop scenarios effectively.
- c_i [1, 8, 8, 1, 1, 1]: These values represent the number of candidate paths at each hop. This sequence reflects the path diversity typical in fat-tree topologies, with more options in the middle layers. By incorporating this realistic path distribution, we can accurately simulate the decision-making processes in both sRetor and cRetor systems, particularly in terms of alternate path selection and load balancing.

The experimental results are presented in Figure 4.17, which clearly illustrates the differences in Packet-In message generation between sRetor and cRetor. These results show a significantly lower probability of Packet-In message generation in sRetor compared to cRetor. This finding aligns closely with the predictions derived from our previous theoretical analysis, validating the accuracy of our model.

The reduced likelihood of Packet-In messages in sRetor can be attributed to its switches' enhanced autonomous routing capabilities. These switches are designed to make forwarding decisions locally, eliminating the need for frequent queries to the controller. Consequently, the sRetor controller receives fewer controller messages, leading to a lower workload compared to

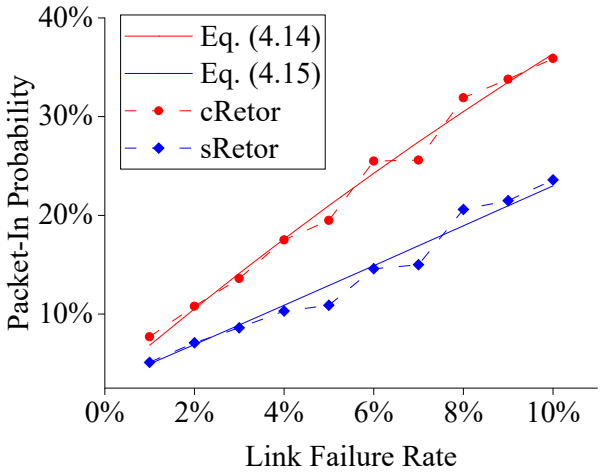


Figure 4.17: Packet-In probability with different link error rates in numerical simulation.

the cRetor controller.

4.2.4 Experiments and Evaluation

1. Experimental Setup

To evaluate the performance of the sRetor routing scheme, we implemented sRetor switches and a sRetor controller based on Ryu[112] in the Estinet network simulator[126]. Estinet is known for its capability to simulate and emulate networks. It supports both traditional routing methods like OSPF and BGP as well as OpenFlow-based SDN networks. This makes it an ideal platform to compare different routing methods, including our sRetor scheme. Ryu is a controller framework built using Python that is specifically designed for SDN and supports the OpenFlow protocol. Various SDN controllers are developed using Ryu, making it a reliable choice for our implementation. Regarding the sRetor controller, it incorporates the same TPDL parser that we used in cRetor. This parser was developed with the ANTLR[114] tool. Using ANTLR allowed us to effectively create a parser that can accurately interpret the TPDL files necessary for the sRetor scheme.

In our experiment, we compared sRetor with several other routing methods: OSPF, Fat-tree two-layer routing[29]/BCube BSR[30], PEMA algorithm[111] and our earlier work, cRetor[88].

For OSPF routing, we used the built-in functions of the Estinet network simulator, where the OSPF protocol implementation of the Quagga software routing suite[116] is integrated. We also implemented the Fat-tree routing method in Estinet, following the method described in

[29]. In this setup, routing tables for each node are pre-generated according to the Fat-tree topology, and the switches in the Fat-tree use these tables to forward packets based on specific prefixes and suffixes.

To verify the applicability of sRetor on different network topologies, this experiment was also conducted on the BCube topology, another common data center network topology. As a server-centric DCN topology, forwarding decisions in BCube are made on servers rather than switches. We chose a 2-layer BCube topology for our experiment because the number of forwarding nodes (servers) in this setup is similar to a Fat-tree topology with $k = 4$. This similarity helps in making a more balanced comparison. We kept other settings, like connection parameters, the same as those used for the Fat-Tree topology. For BCube, we also implemented the BSR algorithm to see how it performs compared to sRetor.

All the detailed settings and parameters we used in this experiment are listed in Table 4.4.

Table 4.4: Simulation Parameters.

Parameter	**Value**	
Experimental network topology	Fat-tree	BCube
Scale of experimental topology	$k = 4$, $k = 8$	2 layers
Number of switch nodes	20, 80	8
Number of server nodes	16, 128	16
Link bandwidth	1Gbps	1Gbps
Link propagation delay	1μs	1μs

2. Flow Start Time

In our study, we define the flow start time as the total delay experienced by the first data packet as it travels from the source node to the destination node, passing through multiple switches. Mathematically, the start time of a flow t_{flow} can be expressed as follows:

$$t_{\text{flow}} = \sum_{i=1}^{m} \tau_i(n) \tag{4.17}$$

Where $\tau_i(n)$ represents the point-to-point delay of packet n at the i^{th} switch, and m is the total number of switches the packet traverses.

During this experiment, we ran simulations on different routing protocols, focusing on measuring their flow start times. The experimental results have been summarized in Table 4.5. The

data clearly shows that cRetor and PEMA, which are based on the traditional SDN architecture, have a longer flow start time compared to the other routing methods. This increased time is due to the need for communication between the switches and the controller, especially for Packet-In and Flow-Mod messages, which are necessary during the flow initiation in cRetor. Moreover, we observed that the flow start time for cRetor increases as the network becomes larger. This trend is due to the growing workload on the controller in larger networks, which in turn affects its responsiveness to Packet-In messages.

Conversely, sRetor's semi-centralized architecture results in a significant improvement in flow start-up times. This is achieved through its reliance on local routing decisions. The table illustrates that sRetor's flow start-up time is comparable to traditional table lookup routing schemes such as OSPF and Fat-tree. Notably, unlike cRetor, the flow start-up time for sRetor does not grow with the increase in network scale, as the size of the network does not influence its local decision-making process.

Table 4.5: Flow start time and convergence time on different routing schemes with different network scales.

Topology Size	**Routing Method**	**Flow Start Time (ms)**	**Convergence Time (ms)**
$k = 4$ Fat-tree	sRetor	1.77	1.79
	cRetor	5.66	1007.08
	OSPF	2.12	50221.33
	Fat-Tree	2.24	2.41
	PEMA	4.90	7041.87
$k = 8$ Fat-tree	sRetor	1.77	1.78
	cRetor	7.64	1843.48
	OSPF	2.65	52001.71
	Fat-Tree	2.26	2.39
	PEMA	9.87	18002.36
2-layer BCube	sRetor	1.77	1.78
	cRetor	5.70	2001.06
	OSPF	1.23	50213.41
	BSR	2.24	2.24

3. Network Convergence Time

Another essential metric we focus on in our study is network convergence time. In SDN architecture, where the control and data planes are separate, the method to measure network convergence time differs. We have already explained these measurement methods in our previous experiments with cRetor.

The simulation results for network convergence time have been summarized in Table 4.5. From these results, we observe that the three topology-aware routing methods used in this experiment, including sRetor, show better performance in terms of convergence time when compared to traditional link-state routing protocols like OSPF. sRetor, much like other dedicated topology-aware routing methods such as Fat-tree/BSR, requires almost no extra time for network convergence. Right from the start of the simulation, these methods can directly use local topology information for forwarding, which significantly speeds up the network's ability to converge.

Additionally, it is worth noting that sRetor's convergence time stays mostly the same with different network sizes. This is because the convergence process we mentioned is not affected by the scale of the network topology. This feature makes sRetor particularly well-suited for large-scale data center networks, where maintaining fast convergence times is crucial.

4. Fault Recovery Time

The fault recovery time is also related to the cumulative probability distribution function of data packet waiting time in the theoretical analysis. This connection is especially relevant in traditional SDN networks, where link faults often result in the generation of Packet-In messages. These messages typically cause delays in the network's ability to recover from faults.

For our experiment, we introduced simple faults into the simulation. These faults disrupted the original paths that switches were used for packet forwarding. Despite these interruptions, the switches were still able to find alternative shortest paths.

Figure 4.18 presents a snapshot of the statistical data we collected from different routing methods when these faults occurred. This snapshot gives us an insight into how each routing method responds to faults and how quickly they can recover.

The results shown in the figure clearly demonstrate how the sRetor switch effectively recovers

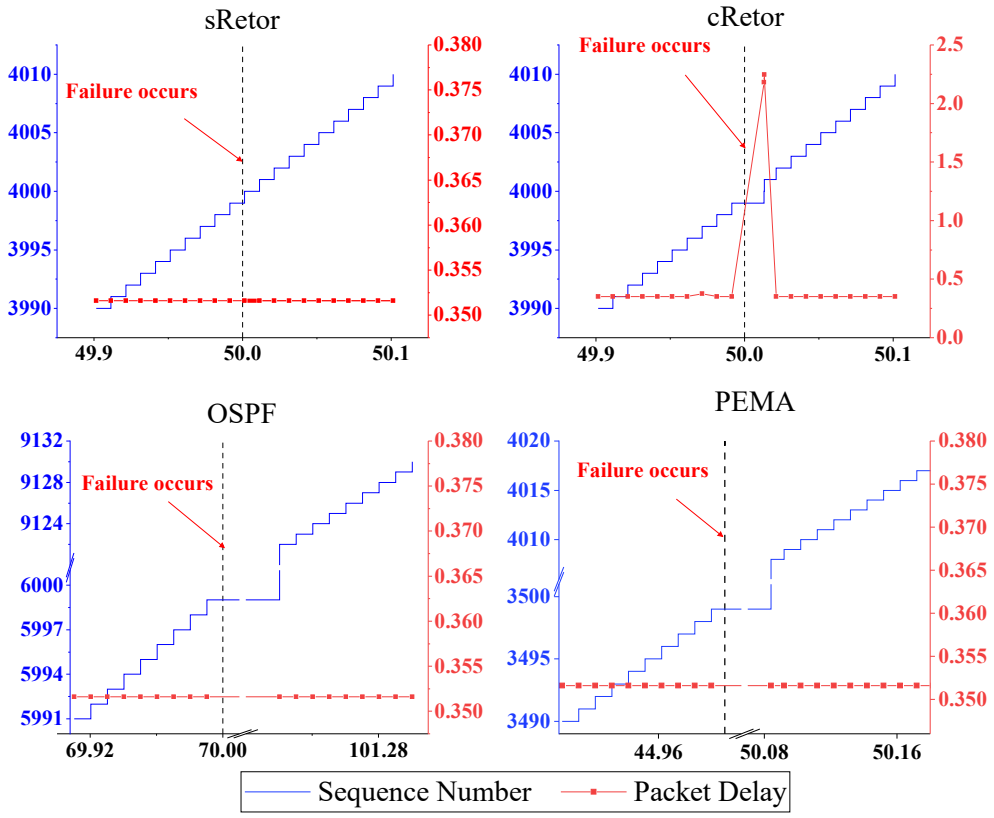


Figure 4.18: The sequence number and delay of packets received in the server side when simple failure occurs.

from faults using its local decision-making capabilities. When a fault happens, the sRetor switch continues to forward packets smoothly. The packets reach their destination within the expected time frame, and importantly, no packet loss is observed.

On the other hand, the situation with cRetor is quite different. After a fault, we noticed a significant increase in packet delay. It went up from 0.35 milliseconds to more than two milliseconds. An interesting observation here is that, in cRetor, while there was no packet loss, two packets almost reached the destination node simultaneously due to the increased delay. This outcome aligns with our model's prediction that packets arriving between 0 and T must wait for the first packet to be processed.

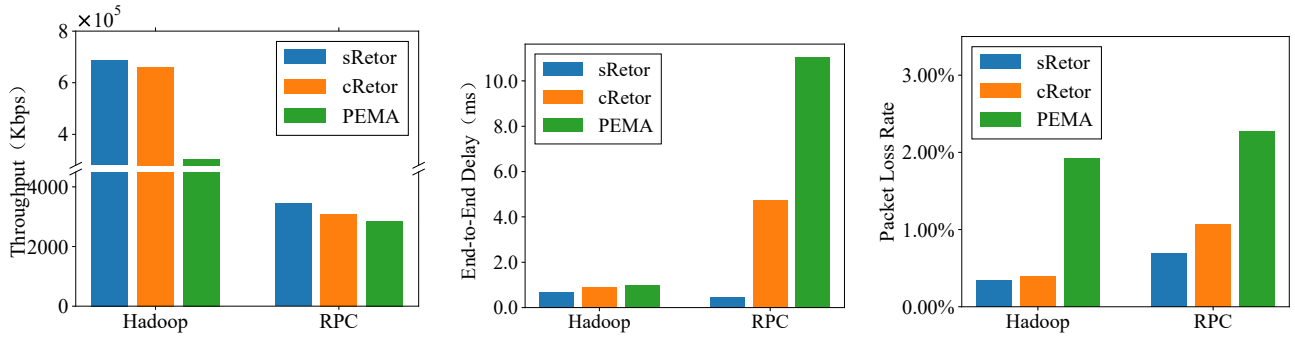
Regarding OSPF, the occurrence of a fault led to a prolonged network disruption, lasting over 30 seconds. This disruption caused massive packet loss, highlighting a significant difference in OSPF's fault recovery capability compared to sRetor and cRetor. In PEMA, the failure also caused a network disruption of about 5 seconds, leading to the loss of corresponding packets.

These findings from the experiment show how each routing method handles faults and their impact on network performance. Particularly, they emphasize sRetor's robustness in maintaining network stability and packet delivery in the face of faults, which is crucial for efficient network operations.

5. Evaluation in Real Traffic Scenarios

In order to comprehensively evaluate the performance of sRetor, we have also carried out experiments using traffic scenarios from real-world data centers. Specifically, we used traffic characteristics from Hadoop at the Facebook data center and RPC request traffic from the Google data center, as described in [127]. The goal was to investigate the performance of sRetor under traffic conditions that are typically found in actual data center networks.

To simulate this real-world traffic in our experiment, we developed a traffic generation tool for the Estinet simulator. This tool was based on the DCTG tool from Mellanox [128]. With this tool, we could create traffic following the characteristics mentioned in [127] and inject it into our simulated network. The experiment focused on comparing the performance of sRetor, cRetor, and PEMA in a Fat-tree network topology with $k = 4$.



(a) Throughput (b) End-to-end Delay (c) Packet Loss

Figure 4.19: The comparison of throughput, end-to-end delay and packet loss of sRetor and cRetor in real scenario traffic patterns.

The results from our experiment with real-world traffic scenarios are shown in Figure 4.19. These results show that sRetor performs better than cRetor and PEMA in several key metrics, including throughput, end-to-end delay, and the overall packet loss rate. It's worth noting that even though sRetor was not originally designed specifically to optimize these particular metrics, they still show improvements. This is due to the shorter time it takes to establish flows and the reduced workload that sRetor offers on the controller. In the context of data center networks,

there is typically a very high number of flows. As mentioned in [129], more than a million flows per second will often arrive at each switch. Therefore, even small flow-handle enhancements can lead to noticeable improvements in overall network performance. So, the advantages that sRetor offers per flow can add significant benefits when we look at the total performance of the network.

4.3 Comparison and Discussion

Both cRetor and sRetor represent novel approaches to topology-aware routing in software-defined data center networks, leveraging TPDL to address the challenges of large-scale deployments. However, they differ in their architectural philosophies and implementation strategies, each offering distinct advantages and trade-offs.

cRetor maintains a purely centralized architecture, making it easier to implement and manage, especially for networks transitioning from traditional SDN setups. Its simplicity in switch design also reduces hardware costs and complexity. However, this centralization can lead to higher initial packet delays and increased controller workload.

In contrast, sRetor's semi-centralized approach distributes intelligence between controllers and switches, achieving lower packet waiting times (85% of packets within 0-150μs compared to only 3% in cRetor) and reduced controller overhead through local forwarding decisions. While sRetor requires more sophisticated switch implementations and initial TPDL configuration, it offers superior performance in terms of flow setup time and failure recovery.

The comparison between cRetor and sRetor reveals a clear trade-off between simplicity and performance. While cRetor offers a more straightforward implementation path with acceptable performance for many scenarios, sRetor provides superior performance metrics at the cost of increased complexity. The choice between these approaches should be guided by specific deployment requirements, existing infrastructure capabilities, and performance needs.

This analysis suggests that both solutions have their place in modern data center networks, with cRetor serving as an excellent entry point for SDN adoption and sRetor offering a more advanced solution for environments demanding optimal performance. Future developments in this area may lead to hybrid approaches that better balance these competing concerns.

4.4 Summary

In this chapter, we first discuss the regular patterns found in data center network topologies and explore routing algorithms that are aware of these topologies. The primary objective of this chapter is to design general topology-aware routing algorithms for data center networks.

To achieve this goal, we propose two novel topology-aware routing schemes for software-defined data center networks based on the Topology Description Language: cRetor and sRetor. These schemes use the regular topological structures identified by TPDL to lessen the workload on controllers in software-defined networks. They address critical challenges like topology discovery and basic forwarding, which help solve the problem of controller bottlenecks that have been a barrier to large-scale SDN deployment. cRetor follows a centralized approach, applying TPDL to the SDN controller while keeping the switches simple. This method offers benefits to those migrating from traditional SDN solutions. On the other hand, sRetor builds upon cRetor but adopts a semi-centralized architecture. It offloads some functions from the controller to the switches, reducing the controller's load even further and enhancing overall performance.

The experimental results demonstrate that both cRetor and sRetor, which are proposed in this chapter, have certain benefits over traditional routing methods. The basic cRetor algorithm can substantially reduce the load on the SDN controller, enabling the large-scale deployment of SDN in data center networks. The enhanced sRetor algorithm can further reduce the controller workload, address the inherent limitations of traditional SDN, achieve flow setup times close to those of traditional dedicated topology-aware routing algorithms, and improve performance in failure recovery. These findings reveal the potential of cRetor and sRetor to facilitate the wider adoption and scalability of SDNs in data center environments.

The routing methods proposed in this chapter address the challenges of efficient routing within data center networks. However, various types of failures constantly occur in data center networks, significantly impacting their efficiency. In the following chapter, we will explore and investigate the issue of failure detection within data center networks, aiming to achieve efficient failure detection with low overhead.

Chapter 5

Probing Matrix Construction Method for Data Center Networks

In modern society, the significance and indispensability of data centers have exponentially increased. They host a wide range of distributed services, including microservices[130], data analytics[131], and artificial intelligence models[1], [132]. These services support many applications that people use every day. The high-bandwidth networks within data centers interconnect numerous servers, ensuring that these services maintain stability and the capacity to scale up in response to increasing demands.

As data center networks keep growing and adopting new networking technologies, network failures have become more of a common occurrence than accidental. Failures in data center networks often lead to more severe effects compared to traditional networks, including service interruptions or even the loss of crucial data [133]. Furthermore, the complexity of managing large-scale data center networks makes it challenging for network administrators to identify and recover from faults rapidly. Consequently, rapid fault detection and localization are crucial in data center network operations.

The fault detection module is an essential component for ensuring the stability of data transmission and the security of the system. Traditionally, networks typically relied on passive fault detection methods (such as SNMP[62] and NetFlow[63]). These methods periodically collect statistical data on incoming and outgoing traffic at network device ports. The methodology is to detect and locate faults through this traffic analysis. However, due to the unpredictability of

traffic in data centers, these passive methods sometimes do not provide accurate results[134], [135]. In response to this issue, active probing has become a more popular approach for fault detection in data center networks. This method involves regularly sending probe messages from some nodes to other nodes. The purpose is to check whether the links between these nodes work correctly. Researchers have proposed and adopted many different active network probing schemes. A well-known example is Microsoft's PingMesh[27], which is a typical active probing scheme. PingMesh installs probe agents on edge servers. These agents measure the latency between servers and analyze these measurements to figure out the current status of the network.

Although beneficial, active detection schemes like PingMesh face two main challenges. Firstly, it is difficult to use deterministic probing paths in traditional data center networks because these networks often have many equivalent paths. Secondly, network administrators need to select the probing paths carefully. This selection is crucial for ensuring high detection accuracy without causing too much extra network traffic. Picking the correct set of paths affects how well the faults are detected, but having too many paths can lead to a lot of additional network load [136].

The recent emergence of technologies such as software-defined networking and programmable switches has led to new fault detection methods that use in-band network telemetry (INT) [137]–[139]. These INT-based methods are promising because they can send probes along specific paths, which helps overcome the issue of having many equivalent paths. However, generating a set of reasonable detection paths with high accuracy and low overhead remains an unresolved challenge. This is especially hard in large data center networks, which require high reliability and capability to support intricate applications. So far, many of the solutions proposed in research fail to offer satisfactory detection accuracy and reasonable overhead.

The reliability of active fault detection schemes relies significantly on the strategic selection of probe paths. PingMesh [140] employs a three-tiered approach for probe list generation, including intra-rack, inter-rack, and inter-data center paths. This strategy is designed based on the experience of experts in network management. While this method aims to encompass a broad range of potential faults, its accuracy leaves room for improvement. To refine this process, deTector [28] introduced the Probing Matrix Construction (PMC) and Packet Loss Localization (PLL) algorithms. These algorithms represent a leap forward in detecting and diagnosing

network faults. However, there remains potential for further improvement in terms of detection efficiency and accuracy. Additionally, the current theoretical framework for understanding and analyzing fault detection and localization processes still exhibits certain limitations, suggesting an area ripe for continued research and development.

This chapter introduces a formal model to compute fault detection accuracy within a given network topology and fault probing matrix. In this model, the construction of the fault probing matrix is redefined as an optimization problem. This problem aims to minimize detection costs while adhering to a specified accuracy threshold. Additionally, we propose a novel metric for evaluating probing matrices. This metric offers a rapid assessment of their detection efficacy. Leveraging this metric, we develop an iterative algorithm for generating fault-probing matrices. This algorithm incrementally selects paths that maximize accuracy gains, continuing this process until the desired level of accuracy is achieved. Our experiments indicate that this approach achieves superior accuracy in fault detection compared to conventional methods, particularly when the number of probing paths is insufficient.

Moreover, this chapter proposes an algorithm for constructing a probing matrix that surpasses the accuracy of traditional heuristic algorithms. The research on deep reinforcement learning has attracted researchers' attention in various domains, including fault detection. In recent years, deep reinforcement learning models have been widely applied in the field of fault detection, as their automated feature learning capabilities offer new potential for solving fault detection problems [141]. The application of reinforcement learning techniques can significantly enhance the efficiency of fault detection and localization for network administrators.

Consequently, this chapter proposes another innovative probing matrix construction algorithm based on deep reinforcement learning. This method uses a deep reinforcement learning model to build a fault-probing matrix with higher accuracy. The experimental results demonstrate that the deep reinforcement learning-based approach proposed in this chapter is practical. In particular, when constrained by a fixed number of probing paths, this algorithm successfully generates a fault probing matrix that achieves greater fault localization accuracy compared to conventional methods.

Specifically, this chapter makes several significant contributions to the field of end-to-end active fault detection and localization in data center networks:

- This chapter establishes a robust theoretical model for active fault detection and localiza-

tion. Through an in-depth analysis of the fault localization process, the chapter derives formulas that quantitatively assess detection accuracy across various probing matrices and fault rates.

- Based on insights from the theoretical model, the chapter introduces a novel evaluation metric. This metric is instrumental in assessing the efficiency of probing matrices. Additionally, a heuristic algorithm is presented that utilizes this metric to construct probing matrices to minimize fault detection overhead.
- This chapter offers a fresh perspective by proposing a probing matrix construction algorithm that harnesses deep reinforcement learning. This innovative approach employs a deep reinforcement learning model, specifically a PPO-based solution, for generating probing matrices. The matrices produced via this method demonstrate superior detection accuracy compared to those generated by traditional algorithms.

The contributions of this chapter significantly advance the state-of-the-art in network fault detection, offering a blend of theoretical insights and practical algorithms that collectively enhance the reliability and efficiency of data center networks. The algorithm proposed in this chapter is particularly important for fault detection within data center networks. By improving the accuracy and efficiency of fault detection processes, this research can aid data center operators in enhancing network reliability while reducing operational costs. Furthermore, the insights and methodologies introduced here lay a foundational framework for future research in data center network optimization and fault tolerance. This paves the way for the creation of more robust and efficient data center infrastructures. This study not only provides new perspectives on fault detection in data center networks but also has the potential to significantly boost the reliability and availability of data center services, reshaping how these critical infrastructures are managed and maintained.

5.1 System Model and Theoretical Analysis

This section delineates the system model for end-to-end active fault detection in data center networks. Initially, it outlines the fundamental architecture of the proposed fault detection system. Following this, the section delves into a detailed analysis of the fault detection accuracy formula. This analysis considers specific conditions of the network topology and the fault detection matrix. Building upon these theoretical foundations, the section concludes by formulating and discussing corresponding optimization problems. These problems aim to enhance the efficacy of fault detection within the given system model, addressing both accuracy and overhead.

5.1.1 System Model

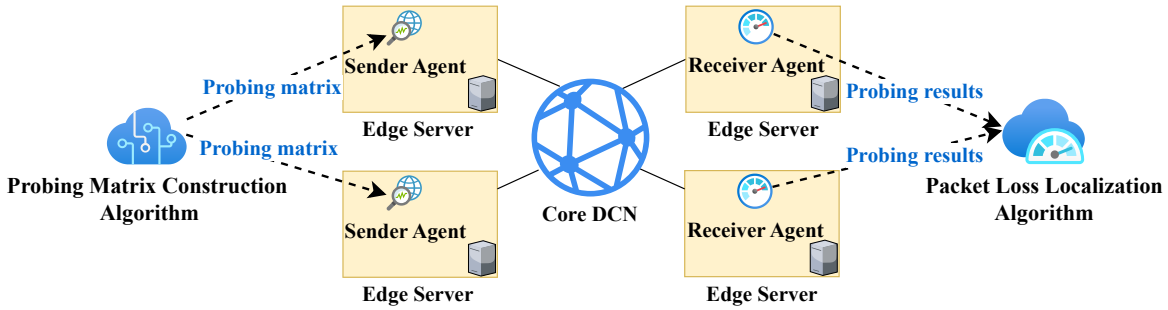


Figure 5.1: The system architecture of end-to-end active probing solutions.

Figure 5.1 illustrates the system model for an end-to-end active fault detection mechanism. The core of this method is the deployment of probing agents on edge servers, which are responsible for dispatching and receiving probe messages. These agents gather aggregated message data and analyze it to identify and locate network faults. The system model includes several key components:

- **Probing Matrix (PM):** The PM is a matrix that outlines the network's fault detection pathways. Each row in this matrix denotes a distinct detection path, while each column represents a link in the network. A value of '1' in the matrix signifies the inclusion of that link in a particular path.
- **Probing Matrix Construction (PMC) Algorithm:** The PMC algorithm generates the probing matrix. Tailoring the matrix to fit the network's structure and specific detection objectives, the algorithm is generally executed before the commencement of

Table 5.1: Notation Table.

Notation	Definition
n	Number of links in the network
$\boldsymbol{L} \in \mathbb{R}^{n\times 1}$	Current status of network links
k	Total number of available detection paths
$R \in \mathbb{R}^{k\times n}$	Candidate detection path matrix
m	Number of detection paths in the detection matrix
$\boldsymbol{P}$	Detection path
$D \in \mathbb{R}^{m\times n}$	Selected detection matrix
$\mathcal{F}_p$	PMC algorithm
$\boldsymbol{s}$	Detection results of matrix D
$\boldsymbol{L'}$	Inferred link state
$\mathcal{F}_m$	PLL algorithm
$\mathcal{A}(D)$	Fault detection accuracy
$Q \in \mathbb{R}^{2^n\times n}$	Fault matrix
$\mathcal{S} \in \mathbb{R}^{2^n\times m}$	Detection results of Q
$\hat{S}_{N\times m}$	Compressed $\mathcal{S}$ after removing duplicate rows
N	Number of unique rows in $\mathcal{S}$
$U_{N\times 1}$	Mapping of $\hat{S}$ and $\mathcal{S}$
$\hat{Q}_i$	Candidate failure scenarios for ith detection result
$\mathcal{G}_t$	Set of indistinguishable faults at step t
C	Index to measure detection results
σ	Standard deviation
β	Distinguishability
π	Reinforcement learning strategy
S	State space
A	Action space
W	End-of-round reward

detection activities. However, it may also be rerun periodically to adapt to changes within the network.

- **Packet Loss Localization (PLL) Algorithm:** This algorithm is the analytical core of the system, processing probe data collected by the agents to detect and localize faults within the network.
- **Probing Agent:** Residing on the edge servers, each probing agent is a lightweight software module equipped with both sending and receiving capabilities. The sending module assembles and dispatches probes according to the PM, guiding them along predetermined paths. Conversely, the receiving module logs probe results and periodically reports this information to the PLL for analysis.

This comprehensive system model integrates these components to facilitate efficient and accurate fault detection and localization within data center networks. The architecture of this system model is highly compatible with contemporary software-defined data center networks. It capitalizes on the inherent flexibility and programmability of these networks. Both the PMC algorithm and the PLL algorithm can be seamlessly integrated into the network controller. This integration empowers the controller to centrally manage the generation of probe matrices and the analysis of probe statistics, thereby achieving rapid response and prompt recovery in the event of faults.

Moreover, the sending and receiving agents are designed as lightweight software modules. These are deployed on edge servers within the data center network. These agents operate with negligible overhead due to their simplicity and minimal resource requirements. This design ensures that their presence and functioning have little impact on the primary services of the data center, guaranteeing the network's operational integrity.

Building on the system model described earlier, we now focus on analyzing the fault detection problem that this model aims to address. Let's consider a network topology comprising n links. To represent the status of these links within the network, we employ a binary vector $\boldsymbol{L} \in \mathbb{I}^n$:

$$\boldsymbol{L} = \begin{pmatrix} L_1 \\ L_2 \\ \vdots \\ L_n \end{pmatrix} \tag{5.1}$$

In this representation, L_i corresponds to the status of the i-th link in the network. A value of $L_i = 1$ signifies that the link functions normally, whereas $L_i = 0$ indicates a failure in that specific link.

In the context of our network model, forwarding paths are composed of multiple interconnected segments. Our system specifically focuses on probing paths where both the source and destination are located at edge server nodes. These probing paths are crucial for the fault detection process. To define the possible set of probing paths, we introduce a candidate probing path matrix, denoted as $\boldsymbol{R}$:

$$\boldsymbol{R} \in \mathbb{I}^{k \times n} = \begin{pmatrix} R_{11} & R_{12} & \dots & R_{1n} \\ R_{21} & R_{22} & \dots & R_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ R_{k1} & R_{k2} & \dots & R_{kn} \end{pmatrix} \tag{5.2}$$

Here, k represents the total number of available probing paths within the network. Each row in the matrix $\boldsymbol{R}$ corresponds to a distinct probing path, while each column represents a link in the network. A value of $R_{ij} = 1$ indicates that link j is part of the probing path i, while a value of 0 signifies that link j is not involved in that particular path.

The objective of the PMC algorithm is to select a specific subset of m probing paths, denoted as $\{p_1, p_2, \dots, p_m\}$, from the k available candidate probing paths. This selection process leads to the construction of the probe matrix $\boldsymbol{D}$, which is derived from the candidate probing path matrix $\boldsymbol{R}$. The construction of $\boldsymbol{D}$ involves extracting the rows corresponding to the selected paths $R_{p_1}, R_{p_2}, \dots, R_{p_m}$ from $\boldsymbol{R}$. This yields the following matrix representation:

$$\boldsymbol{D} \in \mathbb{I}^{m \times n} = \begin{pmatrix} D_{11} & D_{12} & \dots & D_{1n} \\ D_{21} & D_{22} & \dots & D_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ D_{m1} & D_{m2} & \dots & D_{mn} \end{pmatrix} \tag{5.3}$$

The values and meanings of the elements in $\boldsymbol{D}$ are identical to those in $\boldsymbol{R}$. In essence, $\boldsymbol{D}$, the detection matrix, is a refined subset of the broader candidate detection matrix $\boldsymbol{R}$, defined as:

$$\boldsymbol{D} = \mathcal{F}_p(\boldsymbol{R}) \tag{5.4}$$

In this equation, $\mathcal{F}_p$ symbolizes the function of the PMC algorithm, illustrating how the algorithm selects and extracts the optimal probing paths from the candidate matrix to construct $\boldsymbol{D}$.

The detection result for each probing path within the network offers insights into the operational status of the links along that path. Specifically, if any link in a probing path is faulty, the entire

path is marked as faulty. This means that a *faulty* status for a probing path implies that at least one of its constituent links has encountered a failure.

To represent these detection results concisely, we use a binary vector $\boldsymbol{s}$ with a length of m, corresponding to the number of probing paths defined in the probing matrix $\boldsymbol{D}$. This vector is structured as follows:

$$\boldsymbol{s} \in \mathbb{I}^m = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_m \end{pmatrix} \tag{5.5}$$

In this representation, each element of the vector $\boldsymbol{s}$ correlates to the state of a respective probing path. An element value of 0 in this vector indicates that the corresponding probing path is faulty, while a value of 1 signifies that the path is functioning normally.

Finally, the PLL algorithm plays a crucial role in deducing the network-wide link state $\boldsymbol{L}$ based on the detection results $\boldsymbol{s}$. This process leads to the derivation of the inferred link state $\boldsymbol{L}'$, as encapsulated in Eq. (5.6):

$$\begin{aligned} \boldsymbol{L}' &= \mathcal{F}_m(\boldsymbol{R}, \boldsymbol{D}, \boldsymbol{s}) \\ &= \mathcal{F}_m(\boldsymbol{R}, \mathcal{F}_p(\boldsymbol{R}), \boldsymbol{s}) \end{aligned} \tag{5.6}$$

Here, the inferred state of the network links, $\boldsymbol{L}'$, is obtained through a combination of the candidate probing path matrix $\boldsymbol{R}$, the PMC algorithm $\mathcal{F}_p$, the PLL algorithm $\mathcal{F}_m$, and the detection results $\boldsymbol{s}$.

Therefore, the key challenge in effective fault detection revolves around the design of optimal PMC algorithms ($\mathcal{F}_p$) and PLL algorithms ($\mathcal{F}_m$) to minimize the difference between the inferred link state $\boldsymbol{L}'$ and the actual link state $\boldsymbol{L}$. We define this difference as Diff($\boldsymbol{L}', \boldsymbol{L}$). In this chapter, we evaluate Diff($\boldsymbol{L}', \boldsymbol{L}$) in terms of accuracy, which is quantified as the ratio of the number of correctly detected link faults to the total number of actual faults in the network.

5.1.2 Problem Formulation

This section models the detection accuracy within data center networks. Our goal is to establish an analytical model that allows for calculating detection accuracy across any network topology, considering a specific fault localization strategy. Thus, the problem we aim to address is determining the fault detection accuracy, denoted as $\mathcal{A}(\boldsymbol{D})$, concerning a given topology. This

involves analyzing the relationship between the topology matrix $\boldsymbol{R}$ and the detection matrix $\boldsymbol{D}$. By formulating this problem, we can better understand and quantify how different network configurations and detection strategies impact the overall accuracy of fault detection in data center networks.

In a network topology where the link set is $\boldsymbol{L}$, we can define the entire set of possible failure scenarios as the power set of $\boldsymbol{L}$, denoted as $\mathcal{P}(\boldsymbol{L})$. The cardinality of this power set, representing the total number of failure scenarios, is $|\mathcal{P}(\boldsymbol{L})| = 2^n$. Based on this, we construct a failure matrix $\boldsymbol{Q}$ to encapsulate all these scenarios, as expressed in Eq. (5.7):

$$\boldsymbol{Q} \in \mathbb{I}^{2^n \times n} = \begin{pmatrix} Q_{11} & Q_{12} & \dots & Q_{1n} \\ Q_{21} & Q_{22} & \dots & Q_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ Q_{2^n 1} & Q_{2^n 2} & \dots & Q_{2^n n} \end{pmatrix} \tag{5.7}$$

In $\boldsymbol{Q}$, each row represents a distinct failure scenario, corresponding to a specific state of $\boldsymbol{L}$. The element Q_{ij} within this matrix indicates the status of link j in the i^{th} scenario: 1 for a fault and 0 for normal operation. This matrix thus systematically represents the diverse range of possible failure states in the network.

As previously discussed, the probing result for any given path in the network depends on the status of all the links that constitute the path. Accordingly, we can represent the probing results for all potential fault scenarios in a matrix $\mathcal{S}$, where $\mathcal{S} \in \mathbb{I}^{2^n \times m}$. $\mathcal{S}$ is a binary matrix, and the formula for its calculation is presented in Eq. (5.8):

$$\mathcal{S} = \neg\mathcal{B}(\boldsymbol{Q} \cdot \boldsymbol{D}^T) \tag{5.8}$$

Where $\mathcal{B}$ denotes a binarization function, transforming the matrix product into a binary format. Each element $\mathcal{S}_{ij}$ within $\mathcal{S}$ corresponds to the result of probing the j^{th} path under the i^{th} fault scenario. Specifically, $\mathcal{S}_{ij} = 1$ implies that all links in the probing path p_j are functioning normally. Conversely, $\mathcal{S}_{ij} = 0$ indicates the presence of one or more faulty links in that path.

Now, consider the theoretical accuracy formula associated with the fault localization function, $\mathcal{F}_m$. As delineated in Eq. (5.6), $\mathcal{F}_m$ effectively serves as an inverse mapping that interprets the probing results to deduce the current state of network links. However, it's crucial to acknowledge that this mapping is often not one-to-one. This limitation stems primarily from factors such as the design of the probing matrix and constraints related to probing costs. Consequently,

for a specific set of probing results, it's possible that multiple link failure scenarios could produce those same results. This lack of a unique correspondence means that there could be several candidate failure scenarios, represented by different Q_is, that align with the probing results. In such instances, the PLL algorithm faces a challenge: it cannot definitively determine which particular Q_i represents the actual network state. This inherent ambiguity in the fault localization process underscores the complexities of achieving high accuracy in network fault detection. It also highlights the need for advanced algorithmic solutions to effectively construct a better probing matrix that will reduce these constraints.

Let's consider a scenario where all link failures in the network are assumed to follow a binomial distribution with a failure probability p, denoted as $X \sim B(n, p)$. We then focus on the matrix $\mathcal{S}$ and define $\hat{S}_{N\times m}$ as the matrix obtained by removing duplicate rows from $\mathcal{S}$, where N represents the number of unique row vectors in $\mathcal{S}$.

To systematically analyze the relationship between probing results and failure scenarios, we introduce a mapping matrix $U_{N\times 1}$. This mapping matrix serves as an important bridge between the unique probing results and their frequency of occurrence in the complete set of failure scenarios. Formally, U is defined as:

$$U = \{U_i\}, \quad \forall U_i \in U, U_i = |\{S_j\}|, \quad \text{where } \forall j, S_j = \hat{S}_i \tag{5.9}$$

Here, each element U_i represents the count of identical probing result patterns in the original matrix $\mathcal{S}$ that match the unique pattern $\hat{S}_i$. This mapping is essential for two key reasons:

- It quantifies the frequency of each unique probing result pattern, providing insight into the likelihood of specific failure scenarios.
- It enables the weighted consideration of different failure patterns in subsequent probability calculations.

To illustrate this mapping, consider a simple example where multiple failure scenarios result in the same probing pattern. If three different failure scenarios produce identical probing results represented by $\hat{S}_1$, then $U_1 = 3$. This information is crucial for accurately calculating the probability of correctly identifying the actual failure scenario when observing this specific probing pattern.

Next, we define $\hat{Q}_i$ as the set of all possible link state vectors that could yield the probing result

$\hat{S}_i$. The elements of $\hat{Q}_i$ are described by:

$$\hat{Q}_i = \{Q_j\}, \text{ where } j \in [1, U_i], \text{ and } \forall j, S_j = \hat{S}_i \tag{5.10}$$

Here, each $\hat{Q}_{ij}$ is a vector representing a distinct failure scenario in the network. This formulation allows us to systematically represent the array of potential failure states corresponding to specific probing outcomes.

Now consider the accuracy of correctly inferring the failure probing matrix $\hat{\boldsymbol{Q}}_{\boldsymbol{ij}}$ from a given probing result, $\mathcal{S}_i$. If the PLL algorithm selects a failure probing matrix $\hat{\boldsymbol{Q}}_{\boldsymbol{ij}}$ at random from the set of possible matrices $\hat{\boldsymbol{Q}}_{\boldsymbol{i}}$, then the probability of this selection being correct can be expressed by Eq. (5.11):

$$P = \sum_{i=1}^{N} \left(\frac{1}{U_i} \cdot \sum_{j=1}^{U_i} \left(p^{\Sigma \hat{\boldsymbol{Q}}_{\boldsymbol{ij}}} \cdot (1-p)^{n - \Sigma \hat{\boldsymbol{Q}}_{\boldsymbol{ij}}} \right) \right) \tag{5.11}$$

In this equation, n denotes the total number of links within the probing matrix. The term $p^{\Sigma \hat{\boldsymbol{Q}}_{\boldsymbol{ij}}}$ represents the probability that the links indicated as faulty in $\hat{\boldsymbol{Q}}_{\boldsymbol{ij}}$ are indeed faulty, while $(1-p)^{n - \Sigma \hat{\boldsymbol{Q}}_{\boldsymbol{ij}}}$ denotes the probability that the other links are operational. This formulation provides a probabilistic measure of the PLL algorithm's accuracy in identifying the actual state of the network based on the results of the probing paths.

Although the current methodology provides a baseline for fault localization, this chapter seeks to enhance this accuracy by optimizing the selection of $\hat{\boldsymbol{Q}}_{\boldsymbol{ij}}$ from $\hat{\boldsymbol{Q}}_{\boldsymbol{i}}$. To this end, we propose choosing the failure probing matrix $\hat{\boldsymbol{Q}}_{\boldsymbol{ij}}$ that presents the fewest link failures among all the options in $\hat{\boldsymbol{Q}}_{\boldsymbol{i}}$. Formally, this is expressed as:

$$j^* = \underset{j}{\operatorname{argmin}} \, \Sigma \hat{\boldsymbol{Q}}_{\boldsymbol{ij}} \tag{5.12}$$

This approach prioritizes the scenario with the fewest number of faults, under the assumption that fewer faults are more likely in a typical network environment.

Consequently, the revised probability that the PLL algorithm correctly identifies the complete link status is depicted in Equation (5.13):

$$P' = \sum_{i=1}^{N} \left(p^{\Sigma \hat{\boldsymbol{Q}}_{\boldsymbol{ij^*}}} \cdot (1-p)^{n - \Sigma \hat{\boldsymbol{Q}}_{\boldsymbol{ij^*}}} \right), \quad j^* = \underset{j}{\operatorname{argmin}} \ \Sigma \hat{Q}_{ij} \tag{5.13}$$

In this equation, p represents the probability of a link failure. In real-world commercial data centers, this probability typically does not exceed 20% [119]. Thus, we can reasonably assume

that $p < 0.5$. Under this assumption, it can be demonstrated that the value obtained from Eq. (5.13) is equal to or lower than that from Eq. (5.11), indicating an improvement in fault localization accuracy with this revised method.

The proof for demonstrating that Eq. (5.13) yields a higher or equal value compared to Eq. (5.11) involves the following steps:

1. Let's start by defining two probabilities for any j in U_i:

$$P_{1j} = p^{\Sigma\hat{\boldsymbol{Q}}_{\boldsymbol{ij}}} \cdot (1-p)^{n-\Sigma\hat{\boldsymbol{Q}}_{\boldsymbol{ij}}} \tag{5.14}$$

$$P_2 = p^{\Sigma\hat{\boldsymbol{Q}}_{\boldsymbol{ij}}} \cdot (1-p)^{n-\Sigma\hat{\boldsymbol{Q}}_{\boldsymbol{ij}}} \tag{5.15}$$

   To establish the desired inequality, we need to show that for any given U_i, P_2 is greater than the average of P_{1j} values.

2. We then introduce a ratio K defined as:

$$K = \frac{P_{1j}}{P_2} = \left(\frac{p}{1-p}\right)^{\Sigma\hat{\boldsymbol{Q}}_{\boldsymbol{ij}} - \Sigma\hat{\boldsymbol{Q}}_{\boldsymbol{ij^*}}} \tag{5.16}$$

   Given that in Eq. (5.13), j^* is chosen such that it minimizes $\Sigma\hat{Q}_{ij}$, it follows that $\Sigma\hat{\boldsymbol{Q}}_{\boldsymbol{ij}} - \Sigma\hat{\boldsymbol{Q}}_{\boldsymbol{ij^*}} \geq 0$ for all j within U_i.

3. Considering the assumption that $p < 0.5$, which implies $\frac{p}{1-p} < 1$, we deduce that $K < 1$. Consequently, $P_{1j} < P_2$ for all j in U_i. This establishes that the average of P_{1j} values is less than P_2, thus completing the proof.

While Eq. (5.13) provides a probability corresponding to an exact match between the predicted and actual failure scenarios, in practical applications, researchers often assess fault detection and localization performance using accuracy. Accuracy is typically defined as the ratio of the number of correctly detected faulty links to the total number of actual faulty links. To align our model with this typical evaluation metric, we can modify the original equation to encapsulate probabilities under various statistical measures, as demonstrated in Eq. (5.17):

$$\begin{aligned} P(f) &= \sum_{i=1}^{N} \sum_{j=1}^{U_i} \cdot \left(p^{\Sigma\hat{\boldsymbol{Q}}_{\boldsymbol{ij}}} \cdot (1-p)^{n-\Sigma\hat{\boldsymbol{Q}}_{\boldsymbol{ij}}}\right) \cdot \frac{f(\hat{\boldsymbol{Q}}_{\boldsymbol{ij^*}}, \hat{\boldsymbol{Q}}_{\boldsymbol{ij}})}{\Sigma\hat{\boldsymbol{Q}}_{\boldsymbol{ij}}} \\ j^* &= \arg\min_j \Sigma\hat{\boldsymbol{Q}}_{\boldsymbol{ij}} \end{aligned} \tag{5.17}$$

In this equation, f represents statistical functions applicable to different evaluation metrics. The calculation formulas for specific metrics such as the accuracy rate, miss rate, and false alarm

rate are detailed in Eq. (5.18) to (5.20). This adaptation allows for a more comprehensive and practical evaluation of fault detection and localization methods.

$$\mathcal{A}(D) = P(\frac{\#\text{ of detected faults}}{\#\text{ of all faults}}) = f(\hat{\boldsymbol{Q}}_{ij^*}, \hat{\boldsymbol{Q}}_{ij}) = \Sigma(\hat{\boldsymbol{Q}}_{ij^*} \cap \hat{\boldsymbol{Q}}_{ij}) \tag{5.18}$$

$$P(\frac{\#\text{ of undetected faults}}{\#\text{ of all faults}}) = f(\hat{\boldsymbol{Q}}_{ij^*}, \hat{\boldsymbol{Q}}_{ij}) = \Sigma\hat{\boldsymbol{Q}}_{ij} - \Sigma(\hat{\boldsymbol{Q}}_{ij^*} \cap \hat{\boldsymbol{Q}}_{ij}) \tag{5.19}$$

$$P(\frac{\#\text{ of false alarm}}{\#\text{ of all faults}}) = f(\hat{\boldsymbol{Q}}_{ij^*}, \hat{\boldsymbol{Q}}_{ij}) = \Sigma\hat{\boldsymbol{Q}}_{ij^*} - \Sigma(\hat{\boldsymbol{Q}}_{ij^*} \cap \hat{\boldsymbol{Q}}_{ij}) \tag{5.20}$$

The size of the probing matrix $\boldsymbol{D}$, measured in terms of the number of rows, significantly influences the overall probing costs. Specifically, a larger probing matrix necessitates the deployment of more probing agents within the data center network. This increase in the number of agents correlates with a higher consumption of bandwidth resources during the probing process. Consequently, it is crucial to balance probing costs and the required probing accuracy.

In light of this, the primary goal of this chapter is to develop a probing matrix construction algorithm that effectively minimizes probing costs while satisfying a predefined probing accuracy threshold, denoted as $\mathcal{A}$. The optimization challenge can be formally expressed as follows: The objective is to design a probing matrix $\boldsymbol{D}$ that requires the least amount of probing resources, hence minimizing costs, without compromising the stipulated accuracy criteria $\mathcal{A}$. This problem can be encapsulated in the following optimization formula:

$$\begin{aligned} \min \quad & \text{rows}(D) \\ \text{s.t.} \quad & \sum_{i=1}^{N}\sum_{j=1}^{U_i} \cdot \left(p^{\Sigma\hat{\boldsymbol{Q}}_{ij}} \cdot (1-p)^{n-\Sigma\hat{\boldsymbol{Q}}_{ij}}\right) \cdot \frac{f(\hat{\boldsymbol{Q}}_{ij^*}, \hat{\boldsymbol{Q}}_{ij})}{\Sigma\hat{\boldsymbol{Q}}_{ij}} \geq \mathcal{A} \\ & j^* = \text{argmin } \Sigma\hat{\boldsymbol{Q}}_{ij} \\ & f(\hat{\boldsymbol{Q}}_{ik}, \hat{\boldsymbol{Q}}_{ij}) = \Sigma(\hat{\boldsymbol{Q}}_{ik} \cap \hat{\boldsymbol{Q}}_{ij}) \end{aligned} \tag{5.21}$$

In this formula, $|\boldsymbol{D}|$ represents the size of the probing matrix. This optimization approach aims to deliver a probing matrix that achieves the desired accuracy with the most efficient use of network resources.

5.2 Algorithm Design

It is important to recognize that this complex optimization problem of generating an end-to-end failure probing matrix has been classified as NP-hard [142]. This implies that finding an optimal solution within a polynomial time frame is not feasible with current computational methods.

Given this constraint, this chapter introduces two distinct approaches to constructing a practical and effective failure probing matrix. The first approach is a heuristic algorithm, which is designed to navigate the complexities of the problem efficiently, generating a viable failure probing matrix within a reasonable time frame. The heuristic method focuses on balancing computational efficiency with the effectiveness of the probing matrix. The second approach involves the application of deep reinforcement learning. The aim is to leverage the advanced capabilities of deep learning models to further refine and enhance the accuracy of the failure probing matrix. This deep reinforcement learning-based algorithm represents an innovative step towards improving fault detection and localization in complex network environments. By exploring these two methodologies, the chapter contributes to the ongoing efforts to develop robust, efficient, and accurate tools for failure detection in data center networks.

5.2.1 Heuristic Failure Probing Matrix Construction Algorithm

In addressing the challenges associated with efficiently generating a failure probing matrix, this section introduces a heuristic algorithm that strategically decomposes the overarching optimization problem into manageable subproblems. This decomposition enables a more targeted and effective approach to constructing the probing matrix.

The algorithm adopts an iterative methodology, progressively building the final probing matrix by selectively adding probing paths. At each iteration, it focuses on identifying and incorporating a probing path that yields the maximum increase in detection accuracy, denoted as $\Delta\mathcal{A}$. This selection and addition process continues until the probing matrix's cumulative accuracy aligns with the predefined accuracy threshold. By employing this step-by-step approach, the algorithm effectively balances the dual objectives of achieving the desired detection accuracy while maintaining low probing overhead. Each iteration serves as a step towards optimizing the probing matrix, ensuring that each added path contributes meaningfully to the overall detection capabilities of the system.

Although the iterative approach offers a strategic way to enhance probing accuracy, there remains a critical computational challenge. Specifically, the task of computing the probing accuracy of the newly updated failure probing matrix $\boldsymbol{D'}$ at each iteration is computationally intensive. As indicated by Eq. (5.17), the time complexity for calculating probing accuracy for any given failure probing matrix scales exponentially, with a complexity of $O(2^n)$, where n represents the number of physical links in the network. This level of computational demand poses a significant challenge, especially when considering the vast scale of modern data center networks. For instance, Microsoft operates more than 100 data centers on a global scale with millions of servers [19]. Given such large-scale environments, the algorithm's efficiency becomes crucial. An alternative evaluation metric is urgently needed to enable the algorithm to run efficiently. This metric should possess a significantly lower computational overhead to reflect changes in probing accuracy feasibly.

As mentioned in the previous section, a single measurement result $\mathcal{S}$ can map to multiple potential fault conditions, represented by $\boldsymbol{Q}_i$. Consequently, the algorithm faces a decision-making process where it must select a final detection result from among these multiple candidate states $\hat{\boldsymbol{Q}}_{\boldsymbol{i}}$. Various selection strategies can be employed, such as randomly choosing a state or opting for the state with the fewest faults. However, irrespective of the method employed, there is a fundamental trade-off between the number of candidate states, denoted as $|\hat{\boldsymbol{Q}}_{\boldsymbol{i}}|$, and the accuracy of detection. Specifically, a smaller set of candidate states generally leads to a higher probability of accurately selecting the correct state, thereby enhancing overall detection accuracy.

Based on this observation, it is important to establish a new evaluation metric that leverages the partitioning of fault sets. This metric is designed to assess the detection accuracy quantitatively, reflecting the algorithm's capability to distinguish between fault conditions based on the available measurement results. The primary principle of this metric is that the more distinctly the fault sets are partitioned, the more effectively the algorithm can locate the actual fault condition using the measurement data.

Define $\mathcal{G}_t$ as the set of all indistinguishable fault sets at a given step t. That is,

$$\mathcal{G}_t = \{G_1, G_2, \ldots, G_k\}, \tag{5.22}$$

where each G_i within $\mathcal{G}_t$ represents a distinct indistinguishable fault set at step t.

In the initial phase of constructing the probing matrix, the algorithm groups all possible failures into the set $\mathcal{G}_0$. This initial grouping signifies that, in the absence of any probing paths, all failure scenarios are indistinguishable from one another.

As the algorithm progresses and a new probing path p is added to the failure probing matrix $\boldsymbol{D}$, it systematically partitions each fault set within $\mathcal{G}_t$. This partitioning creates two subsets for every element in $\mathcal{G}_t$, categorized based on the presence or absence of the associated failure links on the newly added probing path p. The partitioning process from step t to step $t+1$ can be described as:

$$\mathcal{G}_{t+1} = \{G'_1, G''_1, G'_2, G''_2, ..., G'_k, G''_k\} \tag{5.23}$$

Where,

$$G'_i = \{g | g \in G_i \text{ and } g \times p^T > 0\} \tag{5.24}$$

$$G''_i = \{g | g \in G_i \text{ and } g \times p^T = 0\} \tag{5.25}$$

Here, G'_i represents the subset of G_i where the failure links are part of the probing path p, and G''_i represents the subset where the failure links are not part of p.

By incrementally adding new probing paths to $\boldsymbol{D}$, the iteration counter t correspondingly increases until the specified accuracy constraint is met. Within this framework, each set G_i within the evolving partition $\mathcal{G}_t$ is associated with a corresponding set of potential fault conditions, $\hat{\boldsymbol{Q}}_{\boldsymbol{i}}$. This algorithm aims to enhance probing accuracy by minimizing the cardinality of each $\hat{\boldsymbol{Q}}_{\boldsymbol{i}}$.

Therefore, we propose a new metric to measure the quality of set partitioning, denoted as C. This metric is designed to meet two critical criteria for effective partitioning:

1. The first criterion focuses on minimizing the size of each subset G_i within the partition $\mathcal{G}$. The rationale here is straightforward: smaller subsets facilitate more precise identification of failures.

2. The second criterion emphasizes the need for an even distribution of elements across all subsets $|G_i|$ within $\mathcal{G}$. This uniformity is crucial as it prevents accuracy degradation that could arise from specific subsets being disproportionately large.

With these requirements in mind, we define the metric C as follows:

$$C(\mathcal{G}) = \frac{1 + \sigma(\mathcal{L}(\mathcal{G}))}{|\mathcal{G}|} \tag{5.26}$$

Where, $\mathcal{L}(\mathcal{G})$ represents the set of sizes of all subsets in $\mathcal{G}$, i.e., $\mathcal{L}(\mathcal{G}) = \{|G_i| : G_i \in \mathcal{G}\}$. The function σ calculates the standard deviation of these subset sizes.

A smaller value of C indicates more optimal partitioning. It signifies that each subset in $\mathcal{G}$ contains fewer elements and is more uniformly distributed among the subsets. This balanced partitioning is key to improving the accuracy of the failure probing matrix, as it aligns with both the minimization of subset sizes and the uniformity of element distribution across subsets.

However, the initial number of faults to be partitioned, denoted as $\Sigma\mathcal{L}$, is 2^n, which still presents a substantial computational complexity. To effectively address this, we introduce the distinguishability parameter β, as suggested by Peng et al. [28]. This parameter serves as a practical constraint, limiting the number of concurrent faults considered in the network. Specifically, β denotes the maximum number of faults that can occur simultaneously in the network. By incorporating this constraint, the total number of fault scenarios to be partitioned is significantly reduced to $\sum_{i=1}^{\beta} \binom{n}{i}$. This reduction is based on the combinatorial principle, considering only the fault combinations up to the maximum limit set by β.

It is important to note that in the context of our probing matrix analysis, the exact enumeration of all possible fault scenarios is less critical. What's crucial is ensuring that β is sufficiently large to allow for continued partitioning of the fault set. This approach strikes a balance between manageability and comprehensiveness: it retains enough scenarios to effectively reflect the probing matrix's detection capabilities while limiting the computational complexity to a practical level.

The failure probing matrix construction algorithm, as shown in Algorithm 3, implements a sophisticated approach to iteratively build an optimal probing matrix. The algorithm begins with an initialization phase, where it generates the initial set of fault conditions Q using the LINKOR function. This function considers all possible fault combinations up to β simultaneous faults, grouping these conditions into a single initial partition $\mathcal{G}$. The probing matrix D is initialized as empty and will be constructed incrementally through the subsequent iterations.

The core of the algorithm operates through a continuous iteration loop that persists until the probing accuracy meets the required threshold $\mathcal{A}$. During each iteration, the algorithm systematically evaluates every candidate probing path p from the set R. This evaluation process involves simulating how each potential path would partition the current fault sets and calculating the resulting partition quality metric C. This simulation has important physical

Algorithm 3: Heuristic Failure Probing Matrix Construction Algorithm

Input: R: Set of candidate probing paths

β: Upper limit on the number of simultaneous faults that can be identified

$\mathcal{A}$: Required accuracy

Output: Failure probing matrix $D_{m\times n}$

$Q \leftarrow \text{LINKOR}(R, \beta)$;

$\mathcal{G} \leftarrow \{Q\}$;

$D = \emptyset$;

while $Accuracy(D) < \mathcal{A}$ **do**

  for $p \in R$ **do**

    $\mathcal{G}_p \leftarrow$ Divide $\mathcal{G}$ using p;

    $C_p \leftarrow \frac{1+\sigma(\mathcal{L}(\mathcal{G}))}{|\mathcal{G}|}$;

  end

  $p \leftarrow \text{argmin}_{p' \in R} C_{p'}$;

  $D \leftarrow D \cup \{p\}$;

  $R \leftarrow R \setminus \{p\}$;

  $\mathcal{G} \leftarrow \mathcal{G}_p$;

end

return D;

implications, as each probing path essentially acts as a discriminator in the network, separating fault conditions based on whether they affect the path's performance or connectivity.

The path selection process represents a critical component of the algorithm, where it identifies and selects the candidate path that yields the minimum C value. This selection criterion has significant physical meaning in the context of network diagnostics. The chosen path is one that distinguishes between different fault conditions while creating the most balanced partition of fault sets. In practical terms, this selection maximizes the diagnostic information gained from each probe, ensuring efficient use of network resources. Once selected, this path is incorporated into the probing matrix D and removed from the candidate set R to prevent redundant selection.

The algorithm's computational complexity of $O(m \times k \times n)$ encompasses multiple operational aspects. The term m represents the number of iterations required to build the complete probing matrix, k reflects the evaluations needed for each candidate path per iteration, and n corre-

sponds to the operations necessary to process each path's length. This complexity structure ensures the algorithm remains computationally feasible while maintaining its effectiveness in fault detection.

In the context of Algorithm 3, the index C plays a crucial role in guiding the selection of new probing paths until the set partitioning process is complete. Completeness in this case means that each subset in $\mathcal{G}$ contains only a single element. As long as the partitioning is ongoing (i.e., not every subset in $\mathcal{G}$ is reduced to one element), C remains a valid and effective metric for evaluating the potential improvement brought by each new probing path. However, it is important to ensure that the algorithm does not terminate prematurely, which can occur if the set becomes fully partitioned before satisfying the accuracy requirements. In other words, the condition $|\mathcal{G}| \leq \mathcal{L}(\mathcal{G})$ should be met to prevent early termination of the iteration process. Our experimental findings indicate that with $\beta = 1$, the cardinality of $\mathcal{G}$ tends to rapidly equate to $\mathcal{L}(\mathcal{G})$, thereby hindering further iterations. Setting β to 2 or 3 is typically sufficient to address this issue, depending on the specific accuracy targets required.

5.2.2 Deep Reinforcement Learning-Based Failure Probing Matrix Generation Algorithm

The heuristic algorithm discussed in the previous section is adept at iterative improvements, selecting enhanced probing paths at each step. However, this method tends to converge on local optima without assurance of reaching the global optimum due to inherent computational complexity constraints.

While the algorithm is capable of producing an optimized failure probing matrix, there remains a demand for solutions that edge closer to the optimum, especially in contexts where probing overhead is a critical concern. Reinforcement learning techniques have shown significant promise in solving complex combinatorial optimization problems [143]. Therefore, this section introduces deep reinforcement learning into the generation of the failure probing matrix, aiming to achieve accuracy surpassing traditional algorithms.

Notably, the proximal policy optimization algorithm [97] has been identified as a more effective and stable option than other reinforcement learning strategies, such as DQN, for these kinds of optimization tasks [144]. Hence, we apply the PPO algorithm to constructing probing paths. The application of PPO to probing path construction benefits from its policy stability

feature, which is maintained by restricting the extent of policy modifications, facilitating quicker optimization even in scenarios limited by sample availability. By integrating feedback based on the C metric into the PPO algorithm, this strategy ensures the identification of probing paths that contribute to a more refined and accurate failure probing matrix.

Although the above algorithm can generate relatively optimal failure probing matrices, researchers still hope to find probing matrices closer to the optimum in certain cost-sensitive scenarios. Reinforcement learning methods have achieved remarkable success in solving complex combinatorial optimization problems [143]. Therefore, this section introduces deep reinforcement learning into generating failure probing matrices, aiming to surpass traditional algorithms' accuracy. Related research shows the PPO algorithm is more effective and stable than others like DQN for these problems [144]. Hence, we apply PPO to constructing probing paths. PPO ensures policy stability by limiting policy change magnitudes and optimizing policies quickly without much data. Our approach provides C value-based feedback so PPO can find more reasonable probing paths.

This chapter abstracts the issue as a corresponding environment to address the challenge of constructing a probing matrix using deep reinforcement learning techniques. Within this setting, the actions undertaken by the algorithm involve making selections from a pool of candidate fault detection paths with the goal of assembling a comprehensive probing matrix. However, the solution space for this task is extremely large, characterized by an exponential number of potential choices, leading to a significantly large action space for the deep reinforcement learning algorithm. This expansion of the action space becomes even more massive as the network size increases.

An excessively large action space can adversely affect the deep reinforcement learning model's efficacy, complicating the algorithm's ability to identify effective strategies [145]. This complexity underscores the need for innovative approaches in algorithm design to navigate the challenges presented by the vast action spaces typical of deep reinforcement learning applications in network fault detection.

To address the challenge of the expansive action space, this chapter introduces a refined reinforcement learning environment. This environment treats the construction of the probing matrix as a sequential, multi-step problem. Such an approach allows the problem to be broken down into smaller, more tractable sub-problems. Within each step, the DRL strategy is

tasked with selecting a single probing path to augment the current collection of paths. This methodology substantially narrows the action space for each decision point, as the DRL model evaluates individual probing paths rather than the entire spectrum of possible solutions. The incremental nature of this selection process not only streamlines the decision-making landscape but also enhances the DRL model's learning efficiency by focusing on the stepwise improvement of the probing path set.

Moreover, it's important to note that while this approach to reducing complexity might potentially negatively impact the final outcomes, subsequent experimental results have shown that this method still achieves optimization results superior to traditional approaches. One contributing factor to the success of this approach lies in our method of assigning rewards for actions. Rewards are given at each step of the process and upon the completion of the final probing matrix construction. This dual-phase rewarding system incentivizes the DRL model to optimize both immediate and long-term outcomes, ensuring that each action taken contributes positively toward constructing an effective probing matrix. This observation indicates a well-calibrated balance between managing action space complexity and maintaining high-quality solutions. By effectively handling the complexities of the action space, the DRL model demonstrates enhanced learning capabilities, thereby surpassing traditional approaches in addressing the probing matrix construction problem.

Building upon the aforementioned framework, this chapter presents a problem environment characterized by state spaces, action spaces, and reward mechanisms, as detailed below:

State Space: The state space provides critical information required for every iteration of the DRL strategy's path selection decisions. Within this setup, the set of currently selected probing paths is used as the state information, which is then fed into the policy model π. The policy model π identifies and incorporates a new probing path into the existing collection based on the present state. Therefore, the state vector $S \in \mathbb{I}^{i \times k}$, is defined as follows:

$$S = (S_1, S_2, \ldots, S_k),\ S_i \in \{0, 1\} \tag{5.27}$$

In this equation, S_i denotes the presence of the candidate probing path p_i within the selected path set, with $S_i = 1$ signifying its selection and 0 its absence. The variable k counts all candidate probing paths.

Action Space: After simplification, the action space in this environment at each step is the

probing path selected by the strategy π during that step. The selection is represented directly by the index number of the candidate probing paths. Thus, the action space A can be formalized as a discrete set of integer actions:

$$A = i, \quad i \in \{1, 2, \ldots, k\} \tag{5.28}$$

Reward: In deep reinforcement learning methods, rewards and penalties are crucial components for providing feedback on strategy performance. However, designing an effective reward function for real-world problems is inherently complex. Due to the diversity in problem complexity and characteristics, there can be significant variation in reward function design, presenting certain challenges. This environment's rewards can be divided into two parts: 1) single-step rewards obtained after each action step and 2) final rewards received upon completing a stage and achieving a set of probing paths that meet the requirements.

- *Single-step reward:* After completing each output action step, the reward algorithm first penalizes the selection of repeated probing paths because such paths are meaningless for probing. In this case, the reward algorithm directly issues a large negative reward value (i.e., penalty) and terminates the task. For each unique probing path, the reward algorithm also gives a small penalty to encourage the learning strategy π to continue exploring until the probing matrix reaches the desired size.

- *Final reward:* We use the proposed index C for the reward at the end of each stage. However, since C is inversely related to detection effectiveness (i.e., smaller C values correspond to better detection outcomes), and deep reinforcement learning strategies π converge towards maximizing reward values, this section utilizes the reciprocal of C as the final reward value.

$$W = \frac{1}{C} = \frac{|\mathcal{G}|}{1 + \sigma(\mathcal{L}(\mathcal{G}))} \tag{5.29}$$

This structured approach to defining the state space and action space and designing rewards and penalties is tailored to guide the DRL model efficiently through the probing matrix construction process. By systematically evaluating and expanding the probing path set, the model leverages the current configuration to inform subsequent selections, ensuring the strategy evolves with an understanding of which paths most effectively contribute to the matrix's refinement. Simultaneously, by incentivizing the exploration of new paths and penalizing redundancies, the

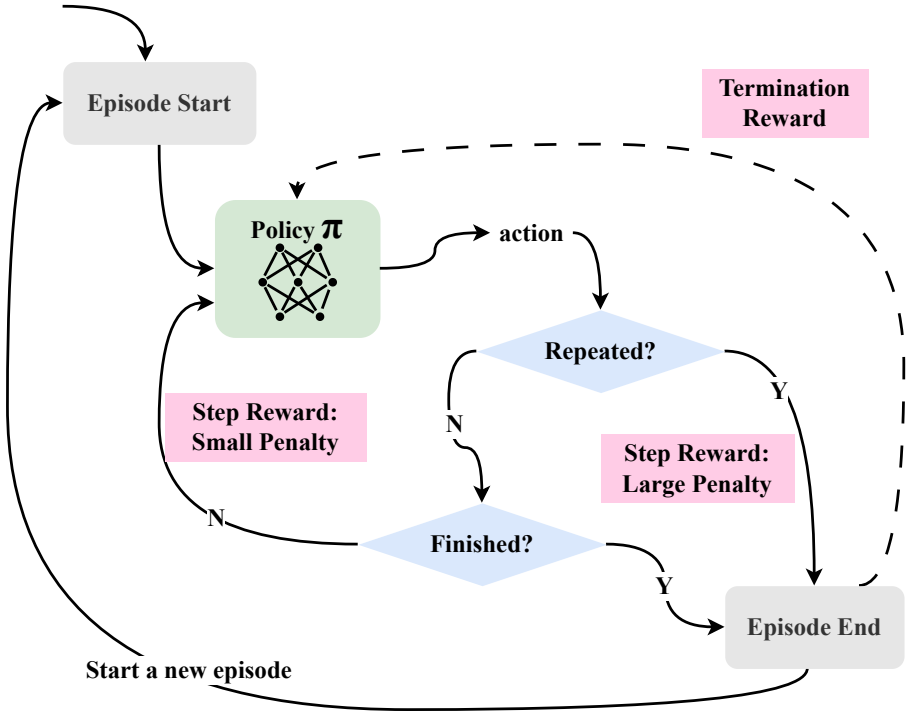


Figure 5.2: The training process of the PPO-based PMC algorithm.

DRL strategy is encouraged to continuously refine its approach, aiming for an optimal balance between thorough exploration and the overall efficiency of the probing matrix. This comprehensive methodology ensures that each step the DRL model takes is deliberate and informed, maximizing the potential for achieving a highly effective and accurate probing matrix.

As depicted in Figure 5.2, the training process showcases how the designed reward mechanism propels the DRL strategy towards selecting probing paths that lead to a higher final reward R. Uniquely, this scheme abstains from using the reward R at intermediate steps and refrains from implementing a discount rate during these phases. The rationale behind this approach is the model's prioritization of the final outcome over the specifics of the intermediate process. Such a strategic focus significantly reduces the risk of converging on local optima, a challenge often encountered with the heuristic algorithm discussed in the preceding section. This method ensures that the DRL strategy maintains a long-term perspective, optimizing for the most effective and efficient probing matrix configuration by the end of the training process.

5.3 Experimental Evaluation and Discussion

This section begins by validating the accuracy of the theoretical model proposed in this chapter. Through simulation experiments, we assessed and verified the predictive outcomes of the model. The experimental results demonstrate a high degree of fit between the theoretical model and the actual simulation outcomes. This indicates that the theoretical model for fault detection accuracy proposed in this chapter can effectively reflect the fault detection results and provide a theoretical foundation for subsequent research. Furthermore, this section conducts a performance evaluation of the two fault detection matrix generation algorithms proposed in this chapter. By analyzing the experimental results under various parameters, this section explores the applicability of these two algorithms. Depending on the specific requirements of different real-world problems, data center network administrators can choose the appropriate algorithm based on its characteristics.

The experimental findings demonstrate the feasibility and effectiveness of the theoretical model and the probing matrix construction algorithms presented in this chapter. This lays a solid theoretical and technical groundwork for future research and applications in the field.

5.3.1 Experimental Setup

To evaluate the performance of the two failure probing matrix construction algorithms we proposed, a simulation platform was constructed for experimental assessment.

A Python-based simulation platform was developed to evaluate the detection accuracy of the probing matrices generated by the PMC algorithm. The evaluation process on this platform proceeds as follows:

1. Generate a Fat-Tree network topology with a specified link failure rate q.
2. Execute the provided PMC algorithm to generate the corresponding probing matrix, and deploy related agents on appropriate edge servers.
3. Run detection agents on edge server nodes to transmit probe packets.
4. Apply the PLL algorithm to locate failures based on the collected probe data.
5. Compare the inferred link failures with the actual link failures to calculate metrics such as detection accuracy.

Additionally, to simulate a more realistic probing topology, this chapter also employs the NS3 network simulator to construct a similar experimental environment. Within NS3's network topology, the transmission and reception processes of probes on edge servers are simulated. After receiving packets, the receiving agents built on destination nodes summarize the results and send them to the PLL algorithm for fault localization. NS3's forwarding model closely mirrors actual data center network nodes, thus better evaluating the chapter's algorithms in real-world scenarios.

This experiment uses deTector[28] and PSD[56] as benchmarks, as they are leading algorithms in end-to-end fault localization. The PSD algorithm does not involve the β parameter, and its default parameters are used in subsequent evaluations. We have improved upon deTector in several vital aspects, including theoretical analysis, superior metrics, and a solution based on DRL. By comparing them with deTector and PSD, the algorithms proposed in this chapter are able to demonstrate their respective advantages.

The simulation platform in this chapter and other related experiments are all executed on a workstation equipped with an Intel Core i7-8700 processor and 32GB of memory. The detailed simulation parameters are listed in the following table.

Table 5.2: Simulation parameters.

Parameter	**Value**
Network Topology	Fat-Tree (k=4)
Number of Pods	2
Total Nodes	36
Number of Links	48
Number of Edge Servers	16
Candidate Probing Paths	128
Selected Probing Paths	1-20
Link Failure Rate q	0.05, 0.1
Identificability β	2, 3

5.3.2 Theoretical Model

This experiment leveraged a $k = 4$ Fat-tree network topology featuring two pods and 128 candidate probing paths to validate the proposed theoretical model under varying conditions of link failure rates. The corresponding probing matrices were generated using Algorithm 3 and evaluated across different sizes (from 1 to 20 probing paths) using the PLL algorithm. The comparative analysis of theoretical predictions and simulation outcomes, as depicted in Figures 5.3 through 5.5, illustrates the fidelity of the proposed model in accurately reflecting real-world fault detection and localization scenarios.

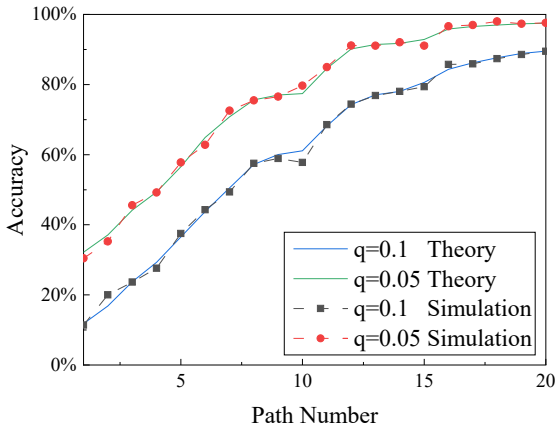


Figure 5.3: The comparison of theoretical and simulation values of the detected failure rate (accuracy) in Eq. (5.17) with different link failure rates.

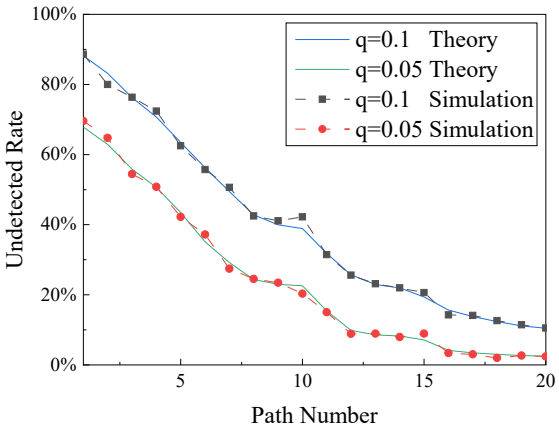


Figure 5.4: The comparison of theoretical and simulation values of the undetected failure rate in Eq. (5.17) with different link failure rates.

The results shown in the figure demonstrate that the simulation values (detection rate, undetected rate, and false alarm rate) under different link failure rates ($q = 0.05$ and $q = 0.1$)

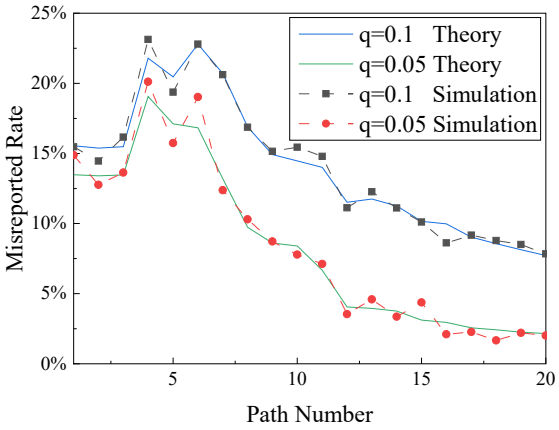


Figure 5.5: The comparison of theoretical and simulation values of the misreported failure rate in Eq. (5.17) with different link failure rates.

closely match the theoretical values provided by equation (5.17). This indicates that the model proposed in Section 5.1 can accurately reflect the outcomes of fault detection and localization under a given network topology and failure probing matrix. This correlation demonstrates the model's capability to serve as a reliable predictor for fault detection and localization effectiveness within specified network configurations and probing matrix parameters.

5.3.3 Heuristic PMC Algorithm

To evaluate the efficiency of the PMC algorithm based on the metric C proposed in this chapter, this section first compares the accuracy of the failure probing matrices obtained when selecting probing paths using two different metrics ($\Delta\mathcal{A}$ and C) in Algorithm 3.

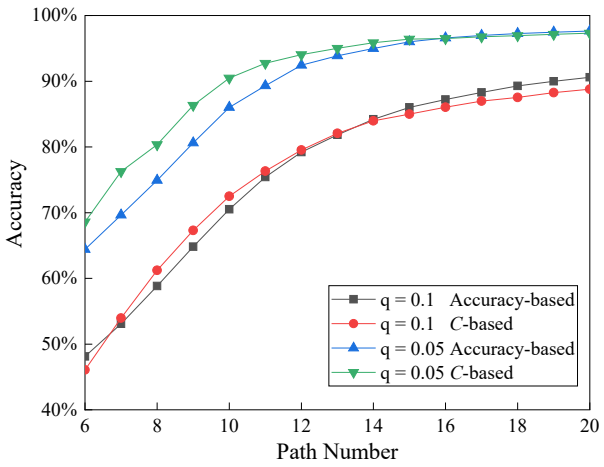


Figure 5.6: The accuracy results when using different indicators in Algorithm 3.

As illustrated in Figure 5.6, the accuracy levels produced by applying $\Delta\mathcal{A}$ and C as evaluation indicators and running the fault detection algorithm are remarkably close under the same link failure rate. Furthermore, the accuracy trends of the two failure probing matrices are also consistent. This demonstrates that our proposed metric C effectively reflects changes in detection accuracy and can accurately represent the contribution of each probing path to the final accuracy.

Next, this section evaluates the performance of the heuristic PMC algorithm proposed in Section 5.2.1 and compares it with previous work deTector[28] and PSD[56].

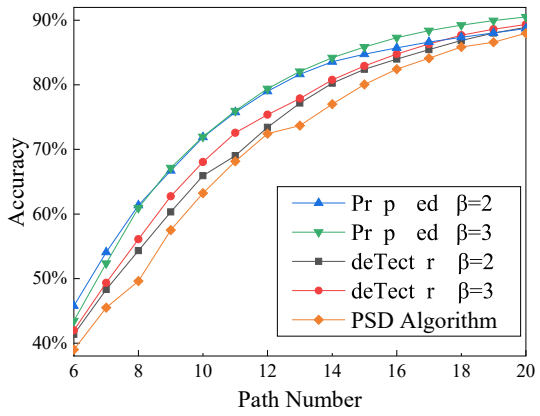


Figure 5.7: The comparison of theoretical accuracy of the proposed algorithm and deTector in different identifiability β.

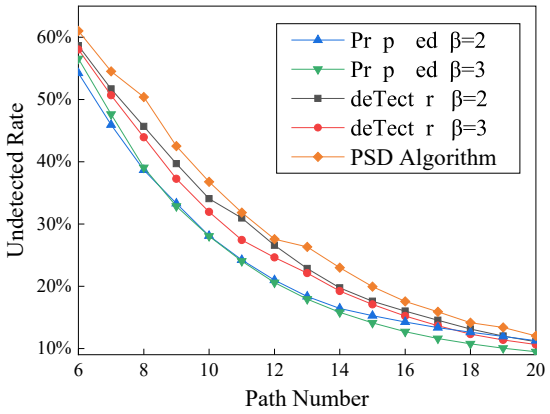


Figure 5.8: The comparison of theoretical undetected failure rate of the proposed algorithm and deTector in different identifiability β.

The performance and differences in theoretical values between Algorithm 3, deTector and PSD, as outlined in Equations (5.18), (5.19), and (5.20), are depicted from Figures 5.7 to 5.9. The

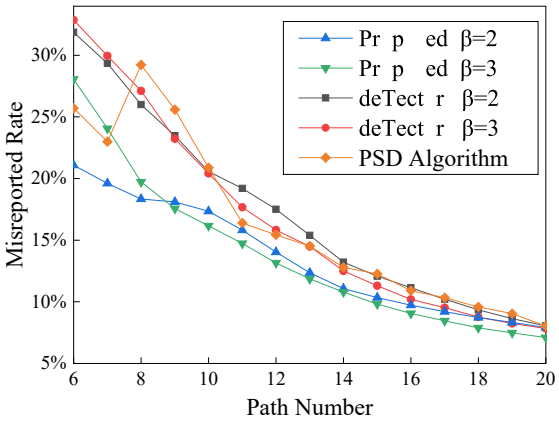


Figure 5.9: The comparison of theoretical misreported failure rate of the proposed algorithm and deTector in different identificability β.

results indicate that the heuristic PMC algorithm proposed in this chapter overall outperforms the PMC algorithm in deTector and PSD. Specifically, compared to the deTector and PSD, Algorithm 3 achieves higher accuracy and lower false-negative and false-positive rates under the same number of probing paths. It is noteworthy that the advantages of the algorithm presented in this chapter become more pronounced when the number of probing paths is not saturated. This implies that, in scenarios with limited probing resources, utilizing this algorithm can yield superior detection results with a limited number of probing paths.

In addition to the aforementioned theoretical comparisons, this section also simulates failure probing scenarios in data center networks to evaluate the detection performance of Algorithm 3, deTector and PSD. These two algorithms' simulation experiments were conducted in numerical simulation and an NS3-based discrete event network simulator. To ensure fairness of comparison, the same failure localization algorithm, PLL, was adopted for both algorithms. This algorithm can effectively locate failed links based on the statistical results of failure probing, making it suitable for deployment in data center networks.

In numerical simulations, this section uses two algorithms to generate failure probing matrices with different numbers of probing paths. Based on the generated failure probing matrices, simulated probing agents are generated. The simulation generates corresponding faulty paths according to the link failure rate parameters. The probes sent by the simulated probing agents produce probing results for each path based on the link failure situations along that path. Finally, the PLL algorithm synthesizes and analyzes the aforementioned probing results to infer the faulty links. The accuracy results obtained from the numerical simulation are presented in

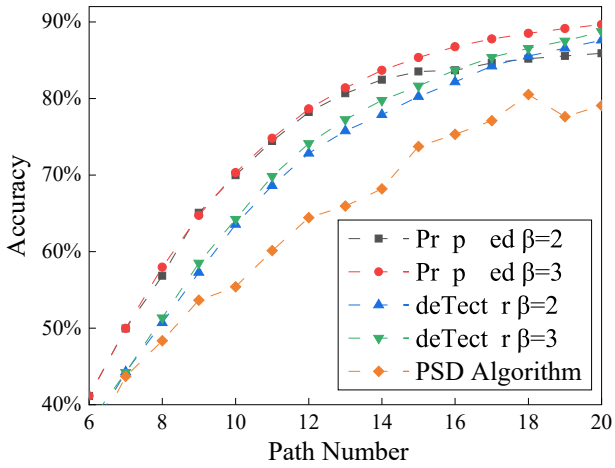


Figure 5.10: The comparison of the numerical simulated failure localization accuracies of the two algorithms in different identifiability β.

Figure 5.10, providing a comparative analysis of the proposed method with deTector and PSD.

To verify the effectiveness of the proposed algorithm in real data center networks, this section also implements the entire process of probing matrix construction, probe agent deployment, and fault localization algorithm in the NS3 network simulator. Specifically, a data center network topology was first constructed in NS3, and two types of probing matrices were generated using different matrix generation algorithms, corresponding to respective probing paths. Subsequently, each probe agent deployed a probing program that sent probe packets according to the pre-generated probing paths and recorded the results of each probe packet on the receiving agent. Finally, all probe results are collected in the controller, where the PLL algorithm is used for fault localization. The experimental results shown in Figure 5.11 demonstrate that, compared with the deTector and PSD algorithm, the proposed algorithm achieved better fault detection accuracy in actual data center networks.

These experiments presented above demonstrate the proposed algorithm's significant improvements in fault detection accuracy. As the number of probing paths increases, the proposed algorithm's detection accuracy improves faster. Additionally, compared to deTector and PSD, the proposed algorithm can achieve similar detection accuracy with fewer probing paths, which implies that the algorithm can reduce overall probing overhead.

Notably, the choice of the distinguishability parameter β also impacts the accuracy of the selected fault detection matrix. The accuracy curves for $\beta = 2$ and $\beta = 3$ are nearly identical

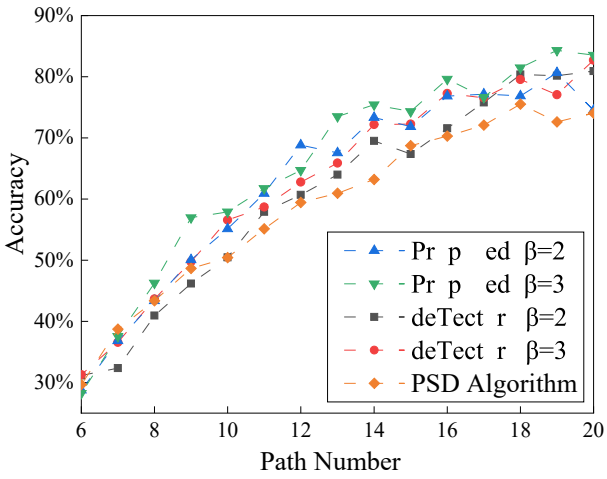


Figure 5.11: The comparison of the NS3 simulated failure localization accuracies of the two algorithms in different identifiability β.

initially. However, as the number of paths exceeds 15, the curve for $\beta = 2$ levels off, whereas the curve for $\beta = 3$ continues to grow. Investigation of the fault detection process revealed that this occurs because when the number of paths exceeds 15, all fault sets in the algorithm using $\beta = 2$ have been completely partitioned, leaving only one unique element in each set. Thus, the number of sets cannot enhance subsequent path detection choices. In contrast, with $\beta = 3$, there are more elements in the partitioned sets, which allows further division even as the number of paths increases beyond 15, thereby guiding the algorithm's probing path selection.

This provides insights for selecting an appropriate value of β. Specifically, the value of β can be determined based on the accuracy requirements for the given scenario. Under a certain accuracy requirement, a smaller β value that satisfies the accuracy goal is preferable, as it reduces the overhead associated with deploying probing agents and probing bandwidth.

5.3.4 DRL-based PMC Algorithm

This section evaluates the fault localization accuracy of the PMC algorithm powered by a deep reinforcement learning model, as proposed in this chapter. The primary objective of utilizing deep reinforcement learning is to enhance the final localization accuracy, aiming to approximate the global optimum closely. Therefore, the DRL-based PMC method proposed in this chapter focuses on improving the final reward value in Eq. (5.29) without considering intermediate values as optimization targets.

Table 5.3: Comparison of the accuracy of the three algorithms under different β values and number of probing paths.

PMC Algorithm	**Accuracy** ($\beta = 2$, paths = 10)	**Accuracy** ($\beta = 3$, paths = 15)
deTector PMC algorithm [28]	63.68%	83.19%
PSD Algorithm [56]	63.23%	80.01%
Heuristic PMC algorithm	72.13%	85.87%
DRL-based PMC algorithm	73.01%	86.68%

The final accuracy comparison of the four algorithms is shown in Table 5.3. The data from the table illustrates that under the same number of probing paths and β values, the DRL-based PMC algorithm can find a failure probing matrix closer to the optimal solution than other algorithms. This indicates that using the DRL-based PMC method can achieve superior detection outcomes under extremely constrained probing conditions.

It's important to note that although the PMC algorithm based on deep reinforcement learning can find a better probing matrix, it also requires longer training times compared to traditional heuristic algorithms. However, considering the vast solution space of this problem, deep reinforcement learning is still an efficient method. For instance, in a Fat-tree network topology with only 128 candidate probing paths, the complexity of finding the optimal failure probing matrix containing ten probing paths from these candidates is as high as $\binom{128}{10} \approx 2 \times 10^{14}$. The complexity further increases exponentially when trying to find 15 or even 20 probing paths. This implies that, even for such a complex combinatorial problem, the DRL-based PMC algorithm can find a near-optimal probing matrix that is closer to the global optimum than traditional methods after several hours of training.

Additionally, it is imperative to emphasize that the inference time is significantly shorter compared to the pre-training time. Therefore, a fully trained DRL model is highly efficient in generating fault detection matrices. This efficiency during the inference phase makes the DRL-based approach particularly attractive for real-time or near-real-time applications where the speed of generating or updating fault detection matrices is crucial despite the upfront investment in training time.

In this experiment, a comparative study was conducted between two widely used deep rein-

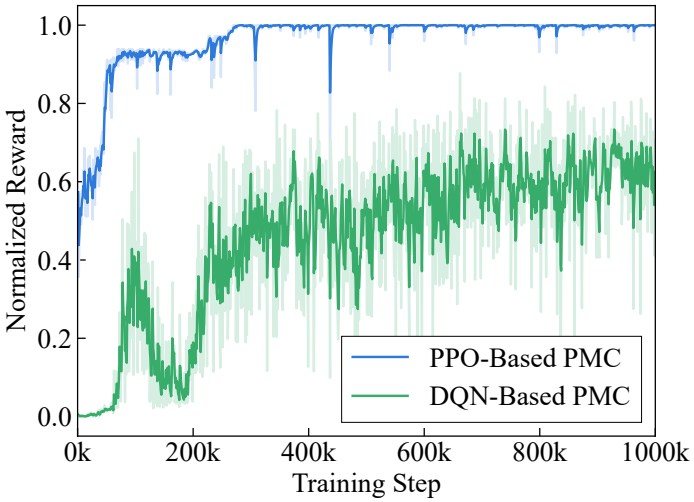


Figure 5.12: The training performance comparison of PPO and DQN-based PMC algorithms.

forcement learning algorithms: PPO and DQN. Despite both being popular in the realm of deep reinforcement learning, they exhibit notable differences in performance across various types of problems. As depicted in Figure 5.12, the PPO algorithm outperforms the DQN algorithm regarding learning speed, fluctuation levels, and the final normalized reward obtained.

Specifically, compared to DQN, the PPO algorithm demonstrates a faster rate of learning growth, meaning it achieves higher performance levels within the same training duration. This is crucial because a shorter learning process translates to reduced training costs and an increased operational frequency of the algorithm. Additionally, PPO exhibits less fluctuation, indicating higher stability throughout the training, which directly impacts the model's reliability. Lastly, the final normalized reward achieved by the PPO algorithm significantly surpasses that of DQN. In this experiment, the final reward of the DQN algorithm is only about 60% of that PPO achieves, further showcasing PPO's distinct advantage in solving complex combinatorial optimization problems.

The superior training performance of PPO compared to DQN can be attributed to several key factors. First, PPO demonstrates higher sample efficiency than DQN. This is primarily because PPO can reuse previous experiences at each update step rather than discarding them outright, like DQN. Second, PPO maintains stability by limiting the magnitude of policy updates and avoiding large updates that could potentially harm performance. Finally, PPO's parallel processing capabilities provide better adaptability when handling multi-task and large-scale problems. Therefore, choosing PPO as the algorithm to address the complex

combinatorial optimization problem in this research not only effectively solves the problems but also gains theoretical backing. This further verifies that the decision to select PPO in this chapter is based on its clear advantages over DQN across various metrics.

Throughout the training process, completing 1000K steps of PPO training took approximately 2 hours on an Intel Core i7-8700 workstation with 32GB of RAM, using only the CPU. Since reinforcement learning typically employs smaller neural networks, the performance gains from using a GPU would be negated by the communication overhead between the CPU and GPU.

Our experimental results demonstrate the effectiveness of the proposed solutions. The results show that our theoretical model is consistent with actual scenarios, and both our heuristic algorithm and DRL-based PMC algorithm outperform other methods. But our algorithms still exhibit certain limitations and drawbacks.

The DRL-based algorithm delivers superior detection performance, achieving higher accuracy with equivalent probing overhead or reduced probing overhead with the same target accuracy. In contrast, the heuristic algorithm provides expedited construction speed but slightly lower detection performance. As a result, these two algorithms cater to distinct application scenarios. When probing overhead is highly sensitive and a smaller probing matrix is desired, the DRL-based algorithm should be employed given the same target accuracy. On the other hand, when the construction speed of the probing matrix is prioritized and its size is less critical, the heuristic algorithm should be selected.

Additionally, scalability remains a significant limitation for both algorithms. As the size of data center networks increases, the matrix construction speed of these methods still has considerable constraints. Further research is warranted in this area.

In summary, the heuristic PMC algorithm and the DRL-based PMC algorithm proposed in this chapter are suited for different usage scenarios. For given target accuracy, in scenarios extremely sensitive to fault detection costs, the DRL-based PMC algorithm should be chosen because it can provide a detection matrix with higher accuracy for the same detection cost. On the other hand, if the target scenario has higher requirements for the speed of generating detection matrices and is relatively insensitive to detection costs, the heuristic PMC algorithm proposed in this chapter is more appropriate because it can deliver high-accuracy detection matrices in a shorter time frame.

5.4 Summary

This chapter first models the end-to-end proactive failure detection method and proposes a theoretical model for detection accuracy given the network topology and probing matrix, filling a gap in theoretical analysis in this field. Subsequently, an alternative metric C for evaluating the effectiveness of detection matrices is designed, and based on this metric, a novel heuristic fault detection matrix generation algorithm is introduced. The evaluation results indicate that the new algorithm, focusing on improving ΔC at each selection step, can achieve the specified accuracy faster than previous algorithms. This chapter also explores integrating deep reinforcement learning into the fault detection matrix construction problem. The DRL-based algorithm enhances the final detection matrix's accuracy by providing feedback on the balance of set partitioning to the policy model.

Both algorithms are experimentally evaluated through numerical simulations and simulations based on the NS3 simulator. The results demonstrate that the DRL-based algorithm can identify more optimal fault detection matrices than previous methods. Lastly, the chapter discusses how the proposed fault probing matrix construction algorithms are suited for different application scenarios, aiming for higher detection accuracy or faster construction speed.

In conclusion, this chapter contributes significantly to the field by providing a solid theoretical foundation for active fault detection methods and introducing innovative algorithms that improve upon traditional approaches. By leveraging heuristic and deep reinforcement learning strategies, it offers versatile solutions tailored to specific operational needs, paving the way for more effective and efficient fault detection in complex network environments.

This chapter presents a fault detection matrix construction algorithm that reduces the detection overhead and enhances the accuracy of fault detection and localization in data center networks. To achieve a high fault-tolerance capability in data center networks, it is necessary to detect and localize faults and respond to them appropriately. In the next chapter, we will explore and introduce algorithms capable of efficient traffic scheduling in the scenarios of faults.

Chapter 6

DRL and GNN-based Fault-aware Flow Scheduling Method for Data Center Networks

With the exponential generation of massive amounts of data and increasing demands for large-scale information processing, the capabilities of a solitary server no longer suffice to meet today's demand. Data centers leverage specific network topologies to bring together an enormous number of server resources to provide powerful computing capability, ample storage capacity, and bandwidth for high-speed transmission [146]. This consolidation enables the processing of big data. Over the past decade, data centers have become pivotal infrastructures underpinning the modern, Internet-driven information era. A wide range of online services and applications, such as video streaming[147], AI platforms [148] and more, are supported by data centers. The growth of these services and applications has led to a surge in data storage and access demand. Researchers must devise efficient traffic scheduling methods to meet these access demands to minimize network transmission latency and response time. Thus, the challenge of orchestrating network traffic distribution has gained increased research attention [149].

In addition, as data center networks continue to expand in size, issues associated with network failure have become increasingly unavoidable [150]. While traditional survival strategies and failure recovery mechanisms exist, they often operate on longer timescales and focus primarily on connectivity restoration rather than performance optimization. Modern data center applications, particularly those requiring real-time processing or low-latency responses [151]–[153],

demand more sophisticated approaches that can maintain service quality during the failure recovery process.

For instance, while fast failover mechanisms can reroute traffic around failed components within milliseconds, they typically use pre-computed backup paths that may not account for current network conditions or ongoing traffic patterns. This can lead to suboptimal resource utilization and potential congestion, especially in scenarios with multiple concurrent failures or during high-traffic periods. Furthermore, traditional recovery mechanisms often prioritize connection restoration over performance optimization, potentially leading to degraded service quality even after basic connectivity is restored.

Therefore, while existing failure recovery mechanisms provide essential basic resilience, there remains a crucial need for intelligent traffic scheduling that can:

- Dynamically optimize flow distribution during the recovery process.
- Balance network load across available paths while accounting for current traffic patterns.
- Maintain application-specific performance requirements even under degraded network conditions.
- Proactively adjust traffic distribution to prevent congestion in surviving network segments.

However, traffic scheduling in data center networks poses a complex online decision-making challenge. The integration of fault scenarios increases these complexities, making many conventional methods unable to generate adequate results. Traditional traffic scheduling methods rely heavily on heuristic algorithms, requiring the algorithm designers to have a comprehensive understanding of the traffic load and network circumstances within the data center in order to achieve an appropriate scheduling strategy [154]. The emergence of software-defined networking technology[26] offers a potential solution to address this problem more efficiently and comprehensively. The centralized decision-making approach of SDN technology facilitates the implementation of traffic scheduling algorithms based on SDN, which enables more efficient methods for network resource allocation and management. As a result, the adoption of SDN technology provides better flexibility and diversity in the scheduling of traffic routing, thereby enabling the quality of network services. Moreover, SDN employs a design paradigm that separates the control plane from the data plane, meaning that the decision-making layer of the network is no

longer scattered across underperforming switches but centralized on general-purpose controller servers. This transition empowers network designers to assign robust general-purpose computing resources and dedicated acceleration devices to controllers [155], as well as to reduce excessive computational overheads. Even with the utilization of multi-controller technology [156], the hardware cost is significantly reduced compared to the cost of replacing and upgrading switches in conventional networks.

On this basis, optimization methods with high computational demands, such as deep reinforcement learning, have been used in software-defined data center networks. The breakthroughs achieved by DRL in diverse fields have paved the way for innovative approaches to traffic scheduling problems within data center networks. In general, DRL aims to develop algorithms and models that are capable of learning directly from data to make decisions without following predetermined heuristic rules. Currently, DRL is recognized as one of the most advanced frameworks for dealing with sequential decision-making problems [157]. Its effectiveness has been emphatically demonstrated across different applications and applied to various real-world scenarios, including those related to network optimization. However, many contemporary traffic scheduling techniques based on DRL [79] overlook the influence of faults on the scheduling algorithm. Most of the sophisticated DRL-based network optimization models suffer from lacking generalization [158]. Therefore, changes in topology can lead to performance drops under faulty conditions. The main reasons behind this are that DRL's neural networks—which are based on conventional fully connected or convolutional neural networks—are not well-suited to learning the nuances of graph structures, especially those associated with dynamic network topologies.

Recently, graph neural networks have attracted the attention of researchers and have been used in DRL. GNNs are now a vital tool for manipulating graph data structures since they are an effective deep learning model for non-Euclidean graph structures [159]. The fundamental idea of GNN is to use deep learning models for processing complicated graphical data through iterative processing [160]. GNNs are suitable for data center networks since the network topology is naturally a graph structure. GNNs in DCNs are able to exploit the spatial information in network topologies and exhibit cross-topology generalization capabilities [161]. The capability also enables GNNs to efficiently detect and react to network topology changes caused by faults.

In this chapter, we propose an approach that combines deep reinforcement learning with graph

neural networks to optimize traffic scheduling in the presence of faults. First, a theoretical analysis was conducted of the challenges associated with traffic scheduling in data center networks under fault scenarios. This analysis led to the development of a suitable mathematical model and the formulation of an optimization problem. Given the intricate complexity of this problem, this chapter presents an approach that combines message-passing neural networks with DRL. The proposed approach leverages the generalization ability of graph neural networks across different topologies, enabling this approach to effectively handle changes in the network caused by faults during the scheduling process.

Specifically, the main contributions of this chapter are as follows:

1) This chapter presents a theoretical analysis of the traffic scheduling challenges in data center networks under fault scenarios. This chapter derives mathematical models and formulates the optimization problem of fault-aware traffic scheduling.

2) This chapter presents a novel approach that combines MPNN with DRL to address the issue of fault-aware traffic scheduling in data center networks. We provide a comprehensive overview of the system architecture, environment setting, model structure, and training process.

3) We trained and evaluated our fault-aware traffic scheduling algorithm. Compared to other DRL-based algorithms, the proposed method increased the flow completion rate in faulty scenarios by over 12%, reducing the average packet loss rate by approximately 76%.

The rest of this chapter is organized as follows. Section 6.1 provides a theoretical analysis of the fault-aware traffic scheduling problem, proposing its system model and problem formulation. Section 6.2 describes in detail the architecture and specifics of our proposed method. Section 6.3 evaluates our method, demonstrating its performance improvements compared to other DRL-based methods. Finally, Section 6.4 concludes this chapter.

6.1 System Model & Problem Formulation

In this section, we construct a mathematical model and provide an analysis of the problem of fault-aware traffic scheduling in data center networks. Based on the proposed model, we formulate the corresponding optimization problems.

6.1.1 System Model

The traffic scheduling algorithm proposed in this section relies on the software-defined networking architecture, where the forwarding decision in the network is made by a central controller, and the switches are standard OpenFlow switches. The centralized controller can be independently deployed on separate servers, providing powerful computational resources to run reinforcement learning-based traffic scheduling algorithms. The architecture of the system is shown in Fig. 6.1.

In this system, the SDN controller connects to the OpenFlow switches in the data plane through its southbound interface (OpenFlow) and dynamically collects network topological structure and switch statistical information. Various controller applications interact with the SDN controller through the northbound interface. The controller applications serve various purposes, including data collection, path selection, and load balancing. Controllers can be endowed with varying capabilities by implementing distinct applications. In the proposed system, we deploy traffic scheduling applications to enable controllers to collect network data and make optimized scheduling decisions.

To analyze fault conditions and traffic scheduling in data center networks, we first define the topology of a data center network. The topology of a data center network can be modeled as a typical graph consisting of a set of nodes and a set of edges. We can define it using the following formula:

$$G = (V, E) \tag{6.1}$$

where V is the set of all nodes in the topology, representing the various network devices (including switches, servers, etc.); and E is the set of all edges in the topology, representing the links between the different network devices in the data center network.

We then model the faults in data center networks. Faults can be categorized into two types: node faults and link faults. From the perspective of network data transmission, a node fault

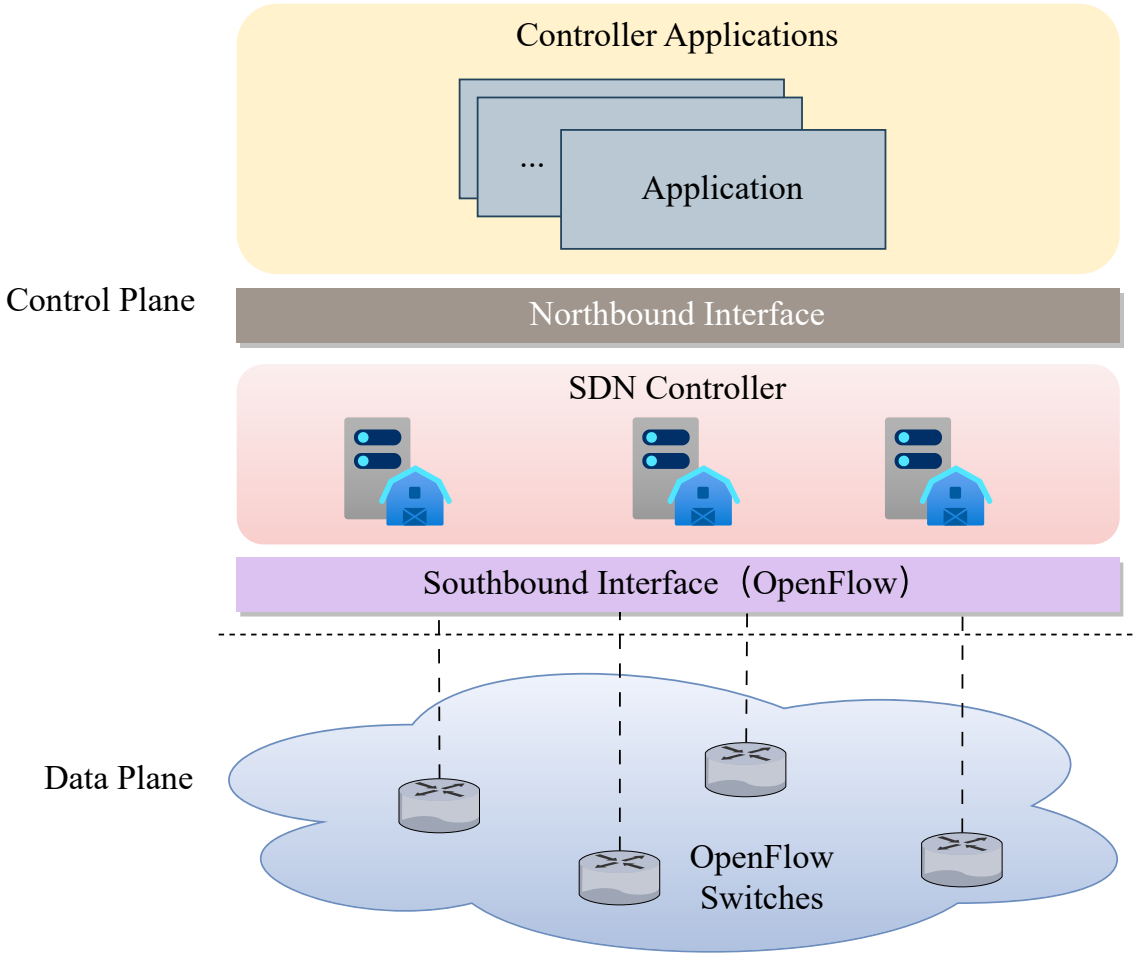


Figure 6.1: The system architecture of the proposed flow scheduling method based on GNN and DRL.

can be equivalently considered as all links connected to the faulty node failing simultaneously [162]. Therefore, node faults can also be modeled as link faults to simplify the system model. We define a function $Z(e,t)$ to describe the fault status of link e in the network topology at time t. The function is defined as:

$$Z(e,t) = \begin{cases} 1, & \text{if link } e \text{ is faulty at the time } t, \\ 0, & \text{otherwise.} \end{cases} \tag{6.2}$$

We propose to use a Markov chain to describe the dynamic behavior of $Z(e,t)$. In this Markov-chain-based model, each edge e can be in one of two states: normal (0) or faulty (1). The state transitions are controlled by the following parameters:

- Fault rate $\lambda(e)$: The probability of transitioning from a normal state to a faulty state
- Recovery rate $\mu(e)$: The probability of transitioning from a faulty state back to a normal

Table 6.1: Notation Definition.

Symbol	**Definition**
G	Graph formed by the data center network topology
V	Set of nodes in the topology
E	Set of edges in the topology
$Z(e,t)$	Fault status of edge e at time t
$\tau(e)$	Duration of the fault on edge e
$\lambda(e)$	Fault rate of edge e
$\mu(e)$	Recovery rate of edge e
N	Number of flows
F	Set of all flows
d_{f_i}	Data rate demand of flow f_i
c_e	Link capacity of edge e
$P(f_i)$	Set of all optional paths for flow f_i
$\mathcal{S}(f_i)$	Selected path for flow f_i
X_i	Completion status of flow f_i
Y_i	Packet loss rate of flow f_i
$t_s(f_i)$	Start transmission time of flow f_i
$t_e(f_i)$	End transmission time of flow f_i

state

Thus, the state transition diagram of this Markov chain is as follows:

$$0 \underset{\mu(e)}{\overset{\lambda(e)}{\longleftrightarrow}} 1 \tag{6.3}$$

Correspondingly, $Z(e,t)$ can be characterized by the following probability-generating function (PGF):

$$P_{Z(e,t)}(z) = p_0(t) + p_1(t)z \tag{6.4}$$

where,

- $p_0(t)$ signifies the likelihood of edge e being normal at time t (i.e., $Z(e,t) = 0$),
- $p_1(t)$ denotes the likelihood of edge e being faulty at time t (i.e., $Z(e,t) = 1$).

As for the duration of the fault, this model defines $\tau(e)$ to represent the sustained duration of the failure on link e. The exponential distribution is a common model for the distribution of fault durations. Therefore, we assume that $\tau(e)$ follows an exponential distribution with parameter $\mu(e)$:

$$\tau(e) \sim \text{Exponential}(\mu(e)) \tag{6.5}$$

In this way, this fault-aware model uses two variables, $Z(e,t)$ and $\tau(e)$, to describe the fault conditions in the network. We can adjust the parameters $\lambda(e)$ and $\mu(e)$ to adapt to different network conditions and fault patterns.

Now we consider the flow scheduling model in the network. We define the set of all flows in the network as F, and the total number of flows as N. Thus, F is denoted as follows:

$$F = \{f_1, f_2, \ldots, f_N\} \tag{6.6}$$

Each flow $f_i \in F$ has a corresponding data rate demand, which we denote as d_{f_i}. Correspondingly, the link capacity of each link e can be defined as c_e. In this system, the transmission and forwarding procedures of the flow are affected by faults. Therefore, regarding the association characteristics between faults and links, as well as the temporal correlation features of faults, we introduce the following parameters for the flow:

- $P(f_i)$: The set of all candidate forwarding paths for flow f_i without considering faults. Each path p consists of multiple links e.
- $t_s(f_i)$ and $t_e(f_i)$: The start and end transmission times of flow f_i.

Then, we define $\mathcal{S}_{f_i,p}$ as the decision variable. That is, when $\mathcal{S}_{f_i,p} = 1$, the path $p \in P(f_i)$ is chosen as the final forwarding path for the flow f_i. Conversely, when $\mathcal{S}_{f_i,p} = 0$, it indicates that path p is not chosen. We define the selected path as $\mathcal{S}(f_i)$. Accordingly, the formula for the decision variable $\mathcal{S}$ is:

$$\mathcal{S} = \mathcal{R}(f_i, c_e, d_{f_i}, P(f_i)) \tag{6.7}$$

$\mathcal{R}$ represents the scheduling algorithm that needs to be designed. This algorithm $\mathcal{R}$ chooses one path from all available paths of the flow f_i as the final forwarding path according to link capacities c_e, flow demand d_{f_i} and candidate paths $P(f_i)$.

6.1.2 Problem Formulation

The quality of the forwarding path chosen by algorithm $\mathcal{R}$ dictates the efficiency and reliability of the network. To evaluate algorithm $\mathcal{R}$, this model focuses on two essential metrics: flow completion rate and average packet loss rate.

The flow completion rate X is defined as the ratio of flows where all packets are successfully transmitted from the source node to the destination node among the total number of flows. This can be expressed as:

$$X = \frac{1}{N} \sum_{i=1}^{N} X_i \tag{6.8}$$

Where,

$$X_i = \begin{cases} 1, & \text{if } \int_{t_s(f_i)}^{t_e(f_i)} Z(e,t)dt = 0, \forall e \in \mathcal{S}(f_i), \\ 0, & \text{otherwise.} \end{cases} \tag{6.9}$$

In other words, the flow f_i is deemed successful only when its selected path $\mathcal{S}(f_i)$ remains fault-free throughout the active duration of f_i.

The average packet loss rate Y is defined as the average packet loss rate for each individual flow. It is defined as:

$$Y = \frac{1}{N} \sum_{i=1}^{N} Y_i \tag{6.10}$$

Where,

$$Y_i = \frac{1}{t_e(f_i) - t_s(f_i)} \int_{t_s(f_i)}^{t_e(f_i)} \left(\alpha \sum_{e \in \mathcal{S}(f_i)} Z(e,t) + \beta \right) dt \tag{6.11}$$

In this context, we adopt a simplified model for packet loss. Under this assumption, every fault link is expected to increase the packet loss rate. Here, α denotes the coefficient that encapsulates the impact of faults on the packet loss rate, and β represents the baseline packet loss rate, which is the packet loss rate when no faults are present. To accurately reflect the real-world packet loss rate, the values of parameters α and β must be calibrated and validated based on actual network data and observations.

Among the two metrics in Eq. (6.8) and Eq. (6.10), the flow completion rate serves as a positive gauge: a higher value suggests better network efficiency and service level. Conversely, the flow's average packet loss rate is a negative gauge, with a lower value indicating fewer network disturbances (such as congestion and packet loss), hence reduced performance degradation and

instability. This model combines these two metrics and proposes the following optimization problem:

$$\max_{\mathcal{S}} \; X - Y \tag{6.12}$$

$$s.t. \sum_{p \in P(f_i)} \mathcal{S}_{f_i,p} = 1, \; \forall f_i \in F \tag{6.13}$$

$$\sum_{f_i \in F} \sum_{p \in P(f_i)} \mathcal{S}_{f_i,p} \times d_{f_i} < c_e, \; \forall e \in p, \forall p \in P(f_i) \tag{6.14}$$

$$\int_{t_s(f_i)}^{t_e(f_i)} \sum_{e \in \mathcal{S}(f_i)} Z(e,t) = 0, \; \forall f_i \in F \tag{6.15}$$

The constraint (6.13) ensures that each traffic demand is restricted to selecting a single path. Constraint (6.14) ensures that the total data rate assigned to the link does not exceed its capacity. Lastly, constraint (6.15) restricts that the chosen path should not encompass any fault-aware links during the transmission duration ($t_s(f_i) \sim t_e(f_i)$) of the flow.

Solving this complicated problem using conventional optimization methods is considerably difficult, especially given the challenge of accurately modeling processes such as packet loss during forwarding. This chapter advocates for the use of deep reinforcement learning to resolve this issue. Specifically, we leverage neural networks to approximate the relationship between optimization objectives and policies in the deep reinforcement learning framework.

6.2 System Framework

This section introduces a traffic scheduling approach for software-defined data center networks. This approach leverages both graph neural networks and deep reinforcement learning to address the fault-aware data center network traffic scheduling problem discussed in the previous section.

6.2.1 Controller Applications

As mentioned in the last section, we deploy corresponding applications on the controller to implement traffic scheduling functions. As shown in Fig. 6.2, the proposed system mainly involves the following applications:

- The **State-sensing application** and the **topology-sensing application** are used to collect relevant statistical information and network topology structure from the data

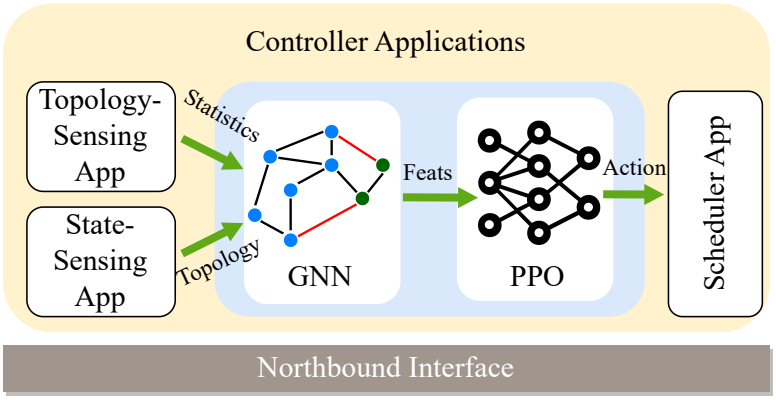


Figure 6.2: The controller applications in this system.

plane.

- The **Reinforcement learning decision application** uses the node statistics and network topology structure provided by the state-aware and topology-aware applications to run pre-trained graph neural networks and deep reinforcement learning models, producing the corresponding traffic scheduling decision results.
- The **Scheduler application** distributes and installs flow tables on the related OpenFlow switches according to the forwarding decision from the reinforcement learning decision application.

The critical part of the proposed method lies in the application of reinforcement learning for decision-making. This application uses a model structure combining MPNN and PPO, which makes flow scheduling decisions based on the current state of the network and the characteristics of the flow. Compared to previous DRL flow scheduling algorithms, the main feature of this method is the introduction of MPNN as a pre-feature extractor for the PPO algorithm. MPNN takes network topology and node statistics as input and broadcasts the features on nodes to their adjacent nodes through message passing. After several rounds of message passing, the relevant features on nodes will gradually propagate to other nodes across the network topology. A significant advantage of this graph neural network structure is that it can adapt to different network topologies.

Therefore, the workflow of this system is as follows:

1. The OpenFlow switches use the southbound interface to connect to the SDN controller during network operation. They then periodically report the state of the network, includ-

ing various switch statistics, to the state-sensing application within the SDN controller. The topology-sensing application detects the current network topology using probe packets, providing fault-aware capabilities to the model.

2. The OpenFlow switches will report to the SDN controller whenever a new flow needs to be scheduled in the network. The controller then runs the reinforcement learning decision application to make a schedule decision for this flow.

3. The reinforcement learning decision application makes scheduling decisions based on node statistics from the state-sensing application and network topology from the topology-sensing application. It first uses MPNN for node feature embedding, then passes the embedded features to a pre-trained PPO model for reinforcement learning inference, ultimately obtaining the decision.

4. The reinforcement learning decision application sends the scheduling decision to the scheduler application. Then the scheduler application schedules data plane traffic through the interface of the SDN controller.

6.2.2 DRL Environment

The deep reinforcement learning scheduling model serves as the core component of our approach. The rationality of the results inferred by this model determines the traffic scheduling performance. Therefore, we need to carefully design the training environment, reward, and training parameters of this model to achieve better model performance.

In reinforcement learning, an agent interacts with the environment through states, actions, and rewards. The agent takes action a corresponding to state s, and then obtains the corresponding reward r. Therefore, a training environment consisting of states, actions, and rewards can be abstracted as a triplet (S, A, R). The detailed design is as follows:

1) State: S represents the state space, describing all possible states of the environment. In this problem's environment, the state includes three parts: a) the current network topology, b) the statistics of the switches, and c) the source and destination nodes of the flow.

We have not used the usual graph data structures to describe the topology of the network. This is due to the fact that data center networks are often relatively large in scale, which makes the full topological structure of the data quite complicated and large in size. On the other

hand, this problem focuses on fault scenarios, where the underlying structure of the network topology remains mostly unchanged, with only some link failures occurring. Therefore, our method represents the current network topology using a fault link set L_{err}, while embedding the fundamental topology structure into the model.

We adopt four commonly used and easily gathered metrics for switch statistics on OpenFlow switches: packets received, bytes received, packets sent, and bytes sent. These metrics indicate switch workload, and the differences between sent and received data can reflect overload.

The environment state is as follows:

$$\begin{aligned} S &= (L_{err}, K, src, dst) \\ &= (L_{err}, \{k_1, k_2, \ldots, k_n\}, src, dst) \end{aligned} \tag{6.16}$$

Where L_{err} represents the set of all link failures, K is the set of all statistics, m is the number of failures, n denotes the number of switches, and src and dst are the source and destination nodes of the flow to be scheduled, respectively. $k_i \in K$ represents the relevant statistical quantities of the i-th switch, namely:

$$k_i =< p_i^r, b_i^r, p_i^t, b_i^t > \tag{6.17}$$

The above statistical quantities denote the number of packets received by switch i, the number of bytes received by switch i, the number of packets sent from switch i, and the number of bytes sent from switch i.

2) Action: A represents the action space, which includes all actions that intelligent agents can perform. $a_t \in A(s_t)$ represents the action chosen at time step t, where $A(s_t)$ is the action space under state s_t.

Traffic scheduling based on SDN typically uses forwarding paths as the action space. However, the data structure for describing forwarding paths is complex. Furthermore, outputting forwarding paths directly as actions increases model complexity and hinders model training convergence. To reduce the complexity of the model and improve the efficiency of training and inference, we have optimized the action in this environment. By combining the topological structure of the Fat-Tree topology, we use the indexes of the core switches to denote the forwarding path that includes the corresponding core switch. This simplifies the action space to $A = \{0, 1, \ldots, p-1\}$ instead of all possible forwarding paths. Similar optimizations can be made for other common data center network topologies.

3) Reward: R denotes the reward function, which describes the feedback received by the agent after performing an action. The reward function can be defined as:

$$R : S \times A \to \mathbb{R} \tag{6.18}$$

That is, based on the chosen action and current state, the environment returns a reward value R. We use the optimization objective function mentioned previously in Equation (6.12) as the final reward value. This makes the agent's goal to maximize the proportion of completed flows and minimize the packet loss rate, while satisfying the constraint conditions.

6.2.3 Policy Network Structure

The policy network π_θ maps the current environmental state to an action, i.e., $a_t = \pi_\theta(s_t)$. Multi-layer, fully connected networks are commonly used as policy networks in deep reinforcement learning networks. However, network faults can change the network topology dynamically, and the generalization ability of conventional fully connected networks is unable to adapt to such situations. In contrast, GNNs have excellent capabilities in responding to dynamic topologies. We propose to integrate GNNs into DRL to improve performance under various fault scenarios. The message-passing neural network is a common type of GNN with a simple structure, strong reasoning abilities, and the ability to adapt to high throughput, which makes it well-suited for network traffic scheduling.

MPNN is primarily applied to graph data structures. Nodes within a graph can communicate with and update their features through their connections. After multiple rounds of message passing, the node features can spread to multi-hop neighbors or even globally. As shown in the left half of Fig. 6.3, MPNN mainly includes three steps:

- Message broadcast stage: In this stage, each node in the graph sends a "message" to its surrounding neighbor nodes, which is derived from transforming the node's own feature x. Our scheme uses a single dense layer containing weights W and biases b to perform a linear transformation.

  $$y = f(x) = W \cdot x + b \tag{6.19}$$

  Then, by multiplying with the graph's adjacency matrix, it passes its message to all neighboring nodes.

  $$\text{msg} = G_{\text{adj}} \cdot y \tag{6.20}$$

Here, G_{adj} is the adjacency matrix of the topology.

- Node Update Stage: In this phase, the messages coming from other nodes are aggregated with the current node's own feature to obtain a new feature matrix, which is then used to update the current node's feature. We also use a single dense layer to aggregate features according to weights. To introduce non-linearity, we include a ReLU activation layer after the fully connected layer. The specific process is:

$$x = ReLU(f([x|\text{msg}])) \tag{6.21}$$

- Feature Readout Stage: After repeating the above two stages for several rounds, the node features have been fully propagated. At this point, the model can aggregate some or all node features to form global features of the graph. Due to the characteristics of the Fat-Tree topology, only the core switch features are read out in this method's readout phase, which will be used in the subsequent stage. This design utilizes the node feature diffusion mechanism so that the core switches can gather the entire network's features and reduce the deep neural network scale in later stages. Therefore, this method is scalable over large networks.

The MPNN architecture in our implementation consists of multiple specialized components designed specifically for network traffic scheduling. The message passing mechanism is structured as follows:

1. Node Feature Design: Each node (switch) in the network maintains a 5-dimensional feature vector:

- Binary indicator for source/destination status (1-dim)
- Traffic statistics including packets received, bytes received, packets sent, and bytes sent (4-dim)

These features are chosen to capture both topological importance and current traffic load conditions.

2. Message Passing Architecture: Our MPNN employs a three-layer architecture with the following components per layer:

- Message Function: A linear transformation that processes node features before passing:

$$M(h_i) = W_m \cdot h_i + b_m \tag{6.22}$$

where h_i represents the features of node i, and W_m, b_m are learnable parameters.

- Update Function: A combination of the aggregated messages and current node features:

$$U(h_i, m_i) = ReLU(W_u \cdot [h_i || m_i] + b_u) \tag{6.23}$$

where m_i represents the aggregated messages at node i, and $[\cdot||\cdot]$ denotes concatenation.

3. Layer Configuration:

- Input Layer: Takes the 5-dimensional node features.
- Hidden Layer: Processes features in a higher-dimensional space.
- Output Layer: Produces the final node embeddings.

This multi-layer design allows the network to capture both local and global topological information through iterative message passing, while maintaining computational efficiency through dimensionality control.

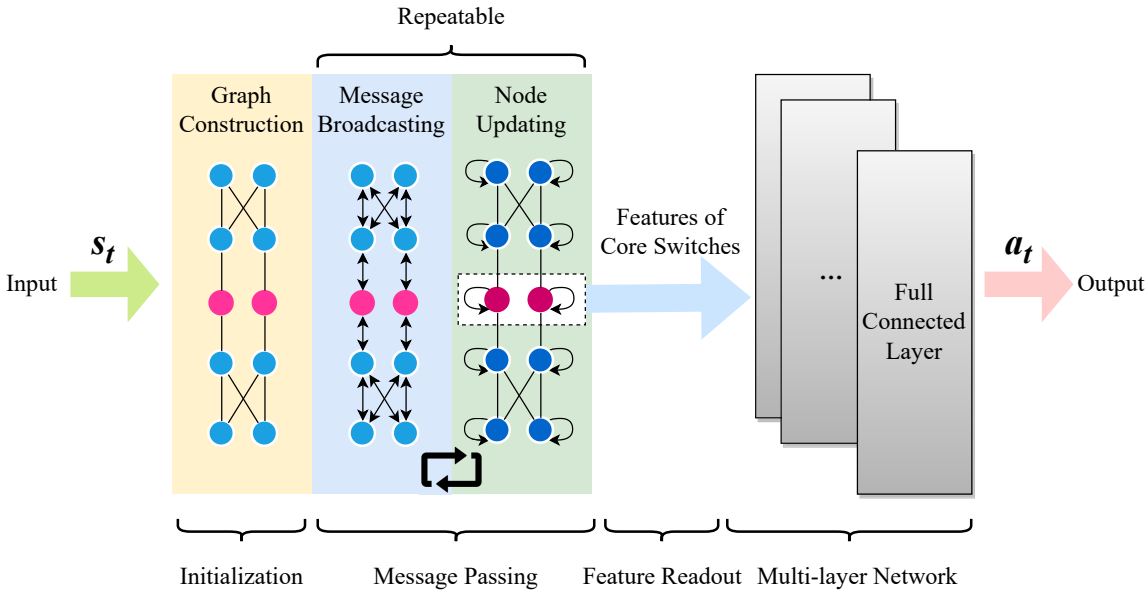


Figure 6.3: The structure of MPNN-based policy model.

After the feature extraction using the MPNN, the graph representation processed by the MPNN is sent as an input to a series of fully connected layers to generate action outputs.

Specifically, this approach divides the policy network model into two steps:

(1) Graph Feature Extraction: First, the MPNN extracts features from the network graph,

including all node state information, adjacency relationships, etc. Each node receives an enhanced new feature through multiple rounds of message passing and aggregation. Finally, the readout function aggregates these node features into a new global graph feature.

(2) PPO Policy Network: The global graph feature is input into the fully connected PPO network. The PPO policy network further processes the input graph features and outputs an action or a set of actions. These actions are used to schedule the traffic distribution state of the network.

In this process, the PPO algorithm optimizes the policy network. The system forms an end-to-end learning framework by collecting environmental feedback, computing policy gradients, and iteratively updating policy parameters based on gradient updates. The system can automatically adapt to changes in the network environment through iterative learning and make optimal traffic distribution decisions.

6.2.4 Training Process of DRL with PPO algorithms

We employ the PPO algorithm for the training process of the reinforcement learning algorithm. At each training step, the reinforcement learning algorithm will choose the action a_t according to the state s_t and observes the reward R_{s_t,a_t} obtained when transitioning to the next state s_{t+1}. The PPO algorithm calculates the policy gradient based on the rewards and accordingly updates the model parameters to improve the predicted reward values and the efficiency of action selection. It uses a policy optimization algorithm so that the model can select the optimal action.

The detailed training process is presented in Algorithm 4. The overall training procedure can be summarized in two steps:

(1) Gathering Data: Action a_t is adopted in the environment in accordance with the current policy $\pi_\theta(a|s)$. Then reward r_t is obtained and the environment will move to the next state s_{t+1}.

(2) Loss Computation and Model Parameters Update: The algorithm computes the policy loss $\mathcal{L}\text{PPO}(\theta)$ based on the collected data, where θ are the neural network parameters.

$$\mathcal{L}\text{PPO}(\theta) = \frac{1}{T}\sum_{t=1}^{T} \min\Big(r_t(\theta)A_t^{\text{clip}},$$
$$\text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right) A_t^{\text{clip}}\Big) - \lambda\mathcal{H}(\pi_\theta(a_t|s_t)) \tag{6.24}$$
$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \tag{6.25}$$
$$\theta \leftarrow \theta - \alpha\nabla_\theta\mathcal{L}_{\text{PPO}}(\theta) \tag{6.26}$$

In this framework, T denotes the number of steps in a round, and θ refers to the model parameters. $\pi_\theta(a_t|s_t)$ denotes the probability of choosing action a_t under state s_t, and $\pi_{\theta_{\text{old}}}(a_t|s_t)$ represents the probability of selecting action a_t under state s_t using the old model parameters before update. A_t signifies the advantage function, $\mathcal{H}$ is the entropy regularization term, λ is the entropy coefficient, α is the learning rate, and ϵ is the clipping parameter in the PPO loss function.

The model utilizes the Generalized Advantage Estimation (GAE) algorithm to calculate the corresponding GAE coefficients for each state-action pair and then derives the policy loss function $L(\theta)$. $L(\theta)$ consists of two components: one is the objective function $J(\theta)$, and the other is a regularization part. The regularization adopted in PPO is PPO-Clip, which restricts the maximum change ratio of the policy during an update to a fixed clip ratio δ.

6.3 Evaluation and Discussion

We implemented and trained the corresponding reinforcement learning model following the aforementioned design to evaluate the MPNN and DRL-based fault-aware traffic scheduling models proposed in this chapter. This section first presents the experimental setup and evaluation metrics used to evaluate the model. We then show the evaluation results of our model compared to DRSIR[79] and TRO [86], previous DRL-based traffic scheduling methods. We also include the fast reroute method FRR [67] into our experiments for comparison. Finally, we discuss the insights obtained from this comparative analysis.

6.3.1 Experimental Setup

In this study, we implemented a lightweight network simulation environment using Python based on discrete event simulation. This simulation environment accomplishes the core func-

Algorithm 4: Training Process

Input: Policy π_θ;

Actor-Critic value function $V(s)$;

Discount factor γ;

Clipping parameter ϵ;

Entropy regularizer $\mathcal{H}$;

Entropy coefficient λ;

Learning rate α;

Batch size b;

Maximum number of iterations M;

Number of training epochs N_E.

Output: Policy parameters θ

Initialize θ and $\theta_{old} \leftarrow \theta$;

for $i \leftarrow 0, \ldots, M$ **do**

  Collect b trajectories $D = (s_t, a_t, r_t, s_{t+1})$ by executing policy $\pi_{\theta_{old}}$ in the environment;

  Compute instant rewards $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$;

  Compute advantage estimate $A_t = R_t - V(s_t)$;

  for $j \leftarrow 0, \ldots, N_E$ **do**

    Compute loss $\mathcal{L}\text{PPO}(\theta)$ according to Eq. (6.24);

    Update policy parameters θ according to Eq. (6.26);

  end

  $\theta_{old} \leftarrow \theta$;

end

return θ

tions of discrete event simulation, including maintaining a simulation event queue sorted by time and supporting operations such as insertion and deletion.

This simulation environment emulates the complete packet forwarding process and the interaction between the control plane and the data plane in the SDN architecture. At the same time, it has a relatively simple structure and high operation speed, enabling ultra-fast real-time simulation. The actual runtime for a 300-second simulation scenario is one or a few seconds.

Model training often requires collecting large amounts of data. Traditional simulation software like Mininet [115] may take substantial time to gather such amounts of data. However, our simulation environment can rapidly generate abundant data for model training, greatly decreasing training time.

Table 6.2: Experimental Parameters

Simulation Parameter	**Values**
Network Topology	Fat-tree topology with $k = 4$
Link Failure Rate	[2%, 4%, 6%, 8%]
Average Failure Duration	[4, 8, 12, 16, 20]
Failure Duration Distribution	Exponential distribution
Number of Flows	256
Control Plane Latency	1ms
Link Datarate	1Gbps
Batch Size	128
Learning Rate	Linear decay from 1e-3 to 1e-5
Full Connected Hidden Layer	[64, 64]
Activation Function	ReLU
Discount Factor (Gamma)	0.999
GAE Lambda	0.95
Maximum Gradient Norm	0.7

During the training and validation process, the commonly used Fat-Tree network topology in data centers is adopted as the test topology. The simulation software generates faults corresponding to the link failure rate and fault duration parameters specified in Section 6.1. An all-to-all traffic pattern is employed, where traffic is randomly generated throughout the network simulation time, and each server can act as both a source node and a destination node. The detailed simulation parameters are presented in Table 6.2.

The reward value of the evaluation environment during the training process is shown in Fig. 6.4. The model achieved good convergence within approximately 400,000 steps, and the reward value was relatively high. On a workstation with an Intel Core i9-13900K CPU, the CPU-based training time was about 1 hour.

The primary evaluation metrics for the comparative experiments include the flow completion

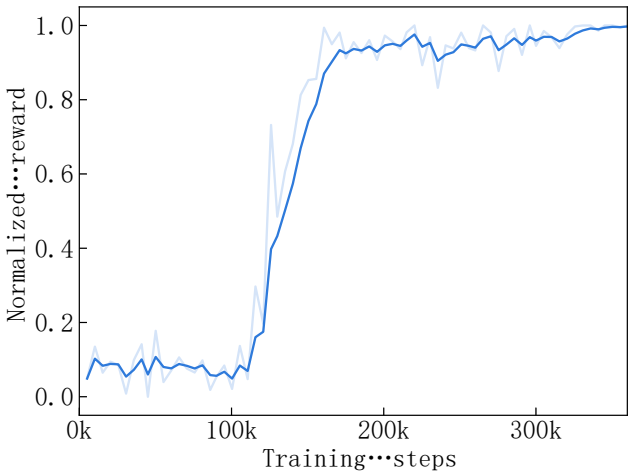


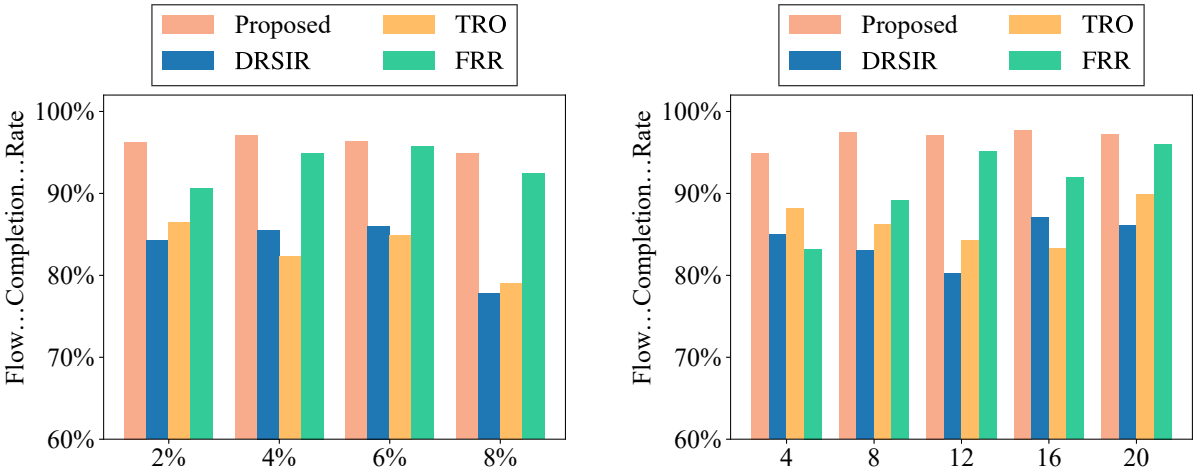
Figure 6.4: The normalized reward values during the training process.

rate and the average packet loss rate used in Section 6.1. Furthermore, this experiment also evaluated and compared the average flow completion time and the throughput, for these methods to validate the effectiveness of the fault-aware traffic scheduling method proposed in this chapter.

6.3.2 Flow Complete Rate and Average Packet Loss Rate

First, we compare the flow completion and average packet loss rates of the proposed methods, DRSIR, TRO and FRR. In the experiment, flows where all data packets are successfully forwarded from the source node to the destination node are counted as completed flows. Flows with missing packets are recorded as incomplete flows. The flow completion rate is calculated as the ratio of completed flows to the total number of flows in the simulation. The average packet loss rate of flows is computed according to the method in Eq. (6.10), by calculating the packet loss rate for each individual flow and then taking the average. The experiment was conducted under various link fault rates and average fault durations, with results shown in Fig. 6.5 and Fig. 6.6.

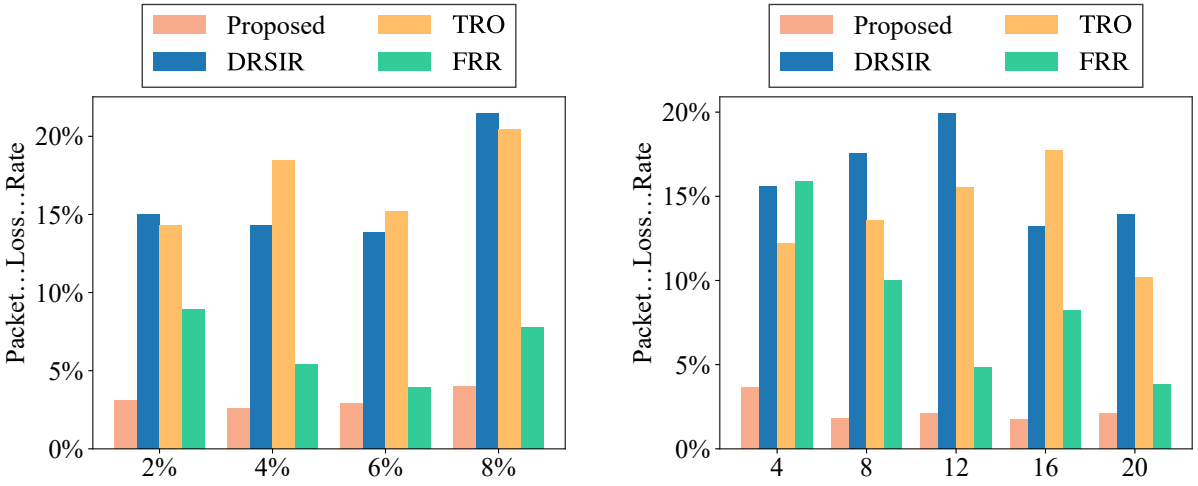
The results in Fig. 6.5 demonstrate that the fault-aware method proposed in this chapter achieves a higher flow completion rate compared to DRSIR, TRO and FRR under various link failure rates and average fault durations. In Fig. 6.5a, under different link failure rates, the average improvement of this work compared to DRSIR, TRO and FRR is 15.42%, 15.68% and 2.95%, respectively. In Fig. 6.5b, under different average fault durations, the average improvement of this work compared to DRSIR, TRO and FRR is 15.04%, 12.26% and 6.60%,



(a) Link fault rates

(b) Average durations of link faults (s)

Figure 6.5: The comparison of the flow completion rates of the proposed method with DRSIR, TRO and FRR under different link fault rates and average durations of link faults.



(a) Link fault rates

(b) Average durations of link faults (s)

Figure 6.6: The comparison of the average packet loss rate of the proposed method with DRSIR, TRO and FRR under different link fault rates and average durations of link faults.

respectively.

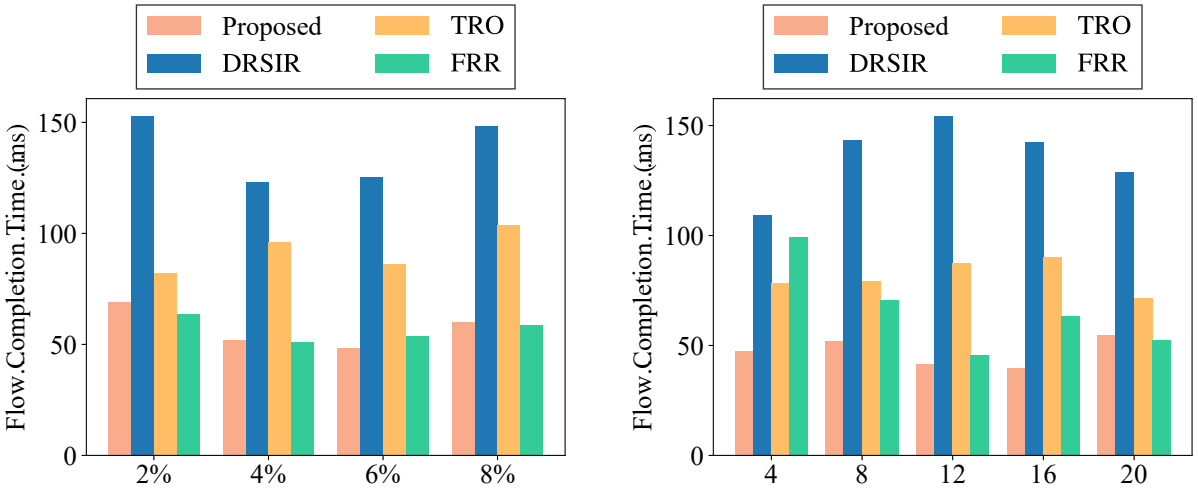
From the perspective of the average packet loss rate of flows, the results in Fig. 6.6 show that the proposed method significantly improves compared to DRSIR, TRO and FRR. Under different scenarios of link failure rate and average fault duration, the average packet loss rate of flows in this work can be managed under 5%. However, the average packet loss rates for DRSIR, TRO and FRR reach around 10% to 15%. Compared to DRSIR, the average packet

loss rates in Fig. 6.6a and Fig. 6.6b of this work have been reduced by 80.46% and 85.52% respectively. Compared to TRO, this work's average packet loss rates have been reduced by 81.43% and 82.57%, respectively. For FRR, the values are 48.13% and 67.87%.

There are two main reasons why this approach is significantly better than other methods. First, the proposed method integrates MPNN into the deep reinforcement learning framework to enhance the responsiveness of the scheduling algorithm to changing environments. Leveraging the generalization capability of MPNN, this method successfully incorporates fault topology as an input for decision-making, further enhancing its adaptability and robustness when facing network faults. In contrast, DRSIR and TRO, as traditional deep reinforcement learning algorithms, are relatively less competent at dealing with faults or responding to dynamic changes in network topology. DRSIR can dynamically collect new topology information during operation and retrain the model based on the updated topology, exhibiting a certain degree of fault awareness. TRO is deployed directly after training without the capability to adapt based on network faults. Moreover, the low load on faulty nodes could mislead the model and result in making worse choices. While FRR provides fast reaction times by implementing rerouting mechanisms entirely in the data plane, its reliance on only local failure information limits its effectiveness. When multiple link failures occur, FRR's lack of global network state awareness means its failover routes may not be optimal, as they must be pre-configured based on local information alone. Although FRR uses randomization to achieve better resilience-load tradeoffs compared to deterministic approaches, its localized decision-making still results in higher packet loss rates compared to our MPNN-based method which leverages global topology information.

Secondly, there is also a significant difference in the deployment between the proposed method and DRSIR. DRSIR is an offline deep reinforcement learning scheduling algorithm that requires periodic collection of topology and statistical information for online training before deploying the trained model into the network. Although DRSIR can accommodate network changes to some extent, the 4-second training process makes it unable to respond promptly and effectively to rapidly occurring network faults. In contrast, the proposed method can be deployed directly after training to respond in real time to changes in the network environment, including various link failures and traffic conditions. Therefore, this immediate response and decision-making capability make our method significantly superior to DRSIR regarding network scheduling efficiency and adaptability.

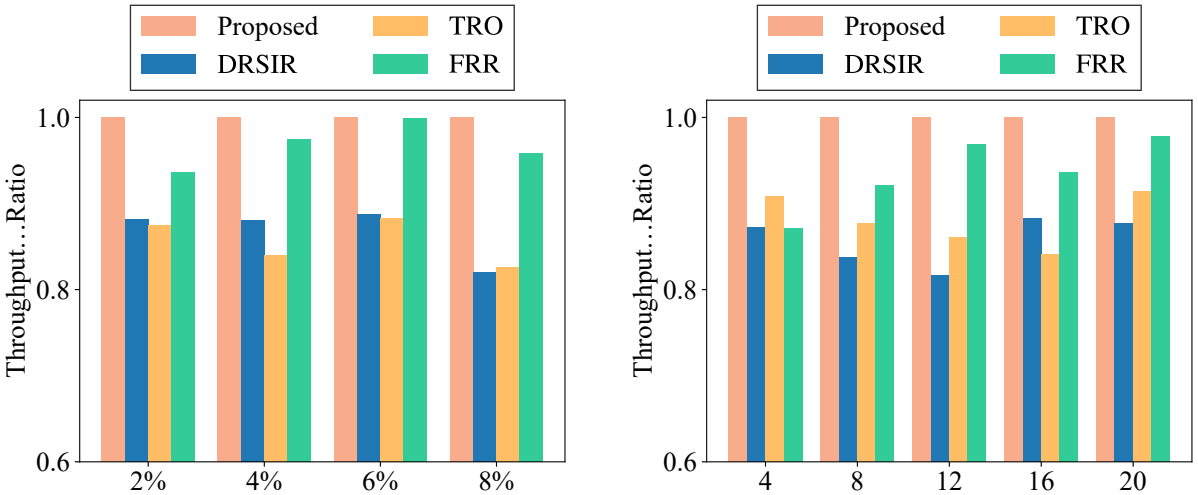
In summary, by integrating MPNN and adopting online training and deployment, the proposed method outperforms the traditional deep reinforcement learning scheduling algorithms DRSIR and TRO both in handling network faults and ensuring service quality.



(a) Link fault rates

(b) Average durations of link faults (s)

Figure 6.7: The comparison of the average flow completion times of the proposed method with DRSIR, TRO and FRR under different link fault rates and different average durations of link faults.



(a) Link fault rates

(b) Average durations of link faults (s)

Figure 6.8: The ratio of throughput of the proposed method with DRSIR, TRO and FRR under different link fault rates and different average durations of link faults.

6.3.3 Average Flow Completion Time and Throughput

We also conducted an in-depth evaluation of the average flow completion time and throughput under different link fault probabilities and average fault durations. The flow completion time and throughput are important metrics for evaluating flow scheduling algorithms. The evaluation results show that this model still outperforms the baseline methods on both metrics. The comparative results for average flow completion time and throughput are shown in Fig. 6.7 and Fig. 6.8.

Specifically, in Fig. 6.7(a) the average flow completion time of the proposed method is 58.38% and 36.97% lower than that of DRSIR and TRO, while achieving comparable performance with FRR. In Fig. 6.7(b), when varying the average fault duration, the proposed method demonstrates even more significant improvements. This indicates that our method can maintain efficient flow transmission even under prolonged fault conditions. Regarding throughput performance, our method consistently outperforms the baselines, with DRSIR, TRO and FRR achieving only 80-90% of our method's throughput under various fault scenarios. These results demonstrate our method's superior capability in resource utilization and service quality maintenance during network failures.

Link failures have significant negative impacts on both flow completion time and throughput. Network failures force data flows to reroute, extending flow completion times and potentially decreasing overall network throughput. While FRR's data plane implementation enables quick local responses to failures, its pre-configured nature may lead to suboptimal routing decisions when facing complex failure patterns, potentially impacting overall throughput. In contrast, by integrating MPNN and real-time decision-making, our method enables the network scheduler to respond to failures promptly and accurately, considering both local and global network states. This comprehensive awareness allows for more intelligent routing decisions, thereby reducing the delay in flow completion time and minimizing throughput degradation caused by failures.

In summary, despite the significant impact of link failures and durations on the flow completion time and throughput, our integration of MPNN and real-time decision-making successfully addresses these challenges and delivers superior performance over baseline methods. Compared to DRSIR and TRO, our method shows substantial improvements in both metrics. While matching FRR's performance in flow completion time, our method achieves better throughput, demonstrating a better balance between quick failure response and optimal resource utilization.

These results validate the advantages of our method in handling faults and guaranteeing service quality, as well as the great potential of deep learning for solving network scheduling problems.

6.4 Summary

This chapter first conducts a theoretical analysis and mathematical modeling of traffic scheduling issues under data center network fault scenarios. We also propose the optimal problem of fault-aware traffic scheduling. This model fills the gap in the theoretical analysis of fault-aware traffic scheduling.

Subsequently, this chapter presents the system architecture design of the fault-aware traffic scheduling system, built on graph neural networks and deep reinforcement learning, as well as the environment and policy models for reinforcement learning. This algorithm utilizes the ability of graph neural networks to generalize on network topology structures, solving the issue of model invalidation caused by topology changes due to faults.

Finally, we carried out experiments and evaluations on the designed method. The results show that, compared to other non-fault-aware traffic scheduling algorithms, the proposed method achieves superior performance in terms of flow completion rate, average packet loss rate, average flow completion time, and throughput.

Chapter 7

Conclusions and Future Work

7.1 Summary of Outcomes

This thesis explores and optimizes data center network technologies from three aspects that impact the efficient and stable operation of data center networks: routing management, fault detection, and traffic scheduling.

This work begins by addressing the foundational aspect of routing functionality and proposing centralized and semi-centralized routing methods, cRetor and sRetor, to tackle the issues of high control overhead and long flow startup times in software-defined data center networks. In the realm of fault detection mechanisms, the thesis addresses problems of high detection overhead and low accuracy by introducing two algorithms for generating fault detection matrices: one heuristic-based and the other powered by deep reinforcement learning. For traffic scheduling, particularly in fault scenarios, the thesis presents fault-aware traffic scheduling algorithms that leverage graph neural networks and deep reinforcement learning.

The innovative contributions of this research include:

1. *High-Performance, Low-Overhead Routing in Software-Defined Data Center Networks with Regular Topologies*: While software-defined networking introduces significant flexibility and centralized management capabilities to data center networks, it also brings inherent SDN issues such as aggregated control plane overhead, controller bottleneck effects, and longer flow setup times. This study aims to address these issues by leveraging the regularity of network topologies and innovatively proposing two topology-aware

routing methods. Experimental results demonstrate that the centralized cRetor method effectively reduces the controller's workload and enhances network scalability while maintaining a simple switch architecture. The semi-centralized sRetor method goes further by offloading some controller functions to switches, solving inherent SDN flaws while retaining flexibility, and making SDN deployments in data center networks more feasible.

2. *Active Failure Probing Matrix Construction in Data Center Networks*: Faults in data center networks can cause service interruptions and data loss, making timely and accurate fault detection crucial. While active fault detection mechanisms offer high timeliness and controllable frequency, they also risk imposing additional loads on network performance. This thesis first theoretically analyzes and models active fault detection mechanisms, studying the relationship between probing matrices and detection accuracy. Based on this analysis, two algorithms for generating fault detection matrices are proposed, utilizing heuristic and deep reinforcement learning approaches to create reasonable matrices that reduce detection overhead and improve performance. Simulation results verify the accuracy of the theoretical model and the effectiveness of the proposed matrix construction algorithms.

3. *Traffic Scheduling Optimization in Fault Scenarios for Data Center Networks*: Proper traffic scheduling is crucial for ensuring the efficient operation of data center networks. Traditional traffic scheduling methods may perform poorly in fault scenarios due to changes in topology. This thesis models and analyzes traffic scheduling issues in fault scenarios, identifying optimization variables and objectives. It then proposes a method combining the generalized capabilities of graph neural networks with deep reinforcement learning to address traffic scheduling in fault scenarios. Experimental results show that the proposed fault-aware traffic scheduling method outperforms conventional algorithms in key metrics such as flow completion rates and throughput.

In summary, this thesis establishes a comprehensive solution system for data center networks by innovating and optimizing the three critical areas of routing management, fault detection, and traffic scheduling within data center networks. The methods and technologies proposed in this work strongly support enhancing the scalability, reliability, and performance of data center networks. The effectiveness of these methods has been validated through simulation experiments, making significant contributions to the development and application of data center

networks.

The research presented in this thesis ensures the efficient and stable operation of data centers and lays the groundwork for further advancing data center networks toward automation and intelligence. This work addresses current challenges in data center network operations and opens new avenues for future research, aiming at creating more resilient, efficient, and self-managing data center environments.

7.2 Future Works

This thesis has researched some key areas of data center networks and achieved certain results. However, as data center networks serve as crucial infrastructure, many challenges remain to be addressed. In connection with the work of this thesis, the following issues are deemed worthy of further exploration:

- Automated Generation of Topology Description Files: In Chapter 4, topology description languages were used to describe regular network topologies, allowing routing algorithms to grasp the regularity of network topologies. Currently, topology description files still require manual summarization and writing by network administrators based on the network topology structure. With the continuous development of machine learning and graph neural networks, it is envisioned that future approaches could use these technologies to automatically generalize and summarize topology description files, or even directly replace topology description files with corresponding pre-trained models.

- Generalized Fault Detection Matrix Generation: In Chapter 5, the deep reinforcement learning method employed only utilized ordinary fully connected networks. As a result, this method requires training for each specific network topology. Introducing technologies like graph neural networks in the future could potentially enable the capability to generate universal failure probing matrices applicable to any topology.

- Multi-objective Optimization in Traffic Scheduling: In the traffic scheduling method of Chapter 6, the model training process mainly focused on fault-related metrics, without incorporating other network performance metrics into the optimization objectives. It is proposed that future work could consider including metrics such as link utilization and throughput into the optimization objectives to design more optimized and ratio-

nal multi-objective optimization problems, thereby further improving traffic scheduling effectiveness.

By addressing these forward-looking challenges, future research can continue to enhance the operational efficiency, reliability, and intelligence of data center networks, paving the way for more advanced and self-adaptive network infrastructures.

Bibliography

[1] M. Naumov, J. Kim, D. Mudigere, *et al.*, "Deep learning training in facebook data centers: Design of scale-up and scale-out systems," *arXiv preprint arXiv:2003.09518*, 2020.

[2] W. Wang, M. Khazraee, Z. Zhong, *et al.*, "TopoOpt: Co-optimizing network topology and parallelization strategy for distributed training jobs," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, Boston, MA: USENIX Association, Apr. 2023, pp. 739–767, ISBN: 978-1-939133-33-5.

[3] Y. Cai, J. Llorca, A. M. Tulino, and A. F. Molisch, "Compute- and data-intensive networks: The key to the metaverse," in *2022 1st International Conference on 6G Networking (6GNet)*, 2022, pp. 1–8. DOI: 10.1109/6GNet54646.2022.9830429.

[4] T.-J. Sun, S.-W. Jeong, H.-J. Jeong, C. S. Hong, S.-B. Park, and E.-N. Huh, "Metaops: Metaverse operations for large-scale metaverse services on distributed cloud," in *Advances in Computer Science and Ubiquitous Computing*, J. S. Park, L. T. Yang, Y. Pan, and J. H. Park, Eds., Singapore: Springer Nature Singapore, 2023, pp. 779–787, ISBN: 978-981-99-1252-0.

[5] C. L. Stergiou and K. E. Psannis, "Digital twin intelligent system for industrial internet of things-based big data management and analysis in cloud environments," *Virtual Reality & Intelligent Hardware*, vol. 4, no. 4, pp. 279–291, 2022, ISSN: 2096-5796. DOI: https://doi.org/10.1016/j.vrih.2022.05.003.

[6] A. Mohiyuddin, A. R. Javed, C. Chakraborty, M. Rizwan, M. Shabbir, and J. Nebhen, "Secure Cloud Storage for Medical IoT Data using Adaptive Neuro-Fuzzy Inference

System," *International Journal of Fuzzy Systems*, vol. 24, no. 2, pp. 1203–1215, Mar. 2022, ISSN: 2199-3211. DOI: 10.1007/s40815-021-01104-y.

[7] H. Woisetschläger, A. Isenko, S. Wang, R. Mayer, and H.-A. Jacobsen, "Federated fine-tuning of llms on the very edge: The good, the bad, the ugly," *arXiv Preprint arXiv:2310.03150*, 2023.

[8] B. Lin, T. Peng, C. Zhang, *et al.*, "Infinite-llm: Efficient llm service for long context with distattention and distributed kvcache," *arXiv Preprint arXiv:2401.02669*, 2024.

[9] S. Jabeen, X. Li, M. S. Amin, O. Bourahla, S. Li, and A. Jabbar, "A review on methods and applications in multimodal deep learning," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 19, no. 2s, Feb. 2023, ISSN: 1551-6857. DOI: 10.1145/3545572.

[10] L. Che, J. Wang, Y. Zhou, and F. Ma, "Multimodal federated learning: A survey," *Sensors*, vol. 23, no. 15, 2023, ISSN: 1424-8220. DOI: 10.3390/s23156986.

[11] K. Hazelwood, S. Bird, D. Brooks, *et al.*, "Applied machine learning at facebook: A datacenter infrastructure perspective," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018, pp. 620–629. DOI: 10.1109/HPCA.2018.00059.

[12] A. de Vries, "The growing energy footprint of artificial intelligence," *Joule*, vol. 7, no. 10, pp. 2191–2194, 2023.

[13] J. Pan, L. Cai, S. Yan, and X. S. Shen, "Network for ai and ai for network: Challenges and opportunities for learning-oriented networks," *IEEE Network*, vol. 35, no. 6, pp. 270–277, 2021. DOI: 10.1109/MNET.101.2100118.

[14] Z. Zhou, M. Shojafar, M. Alazab, J. Abawajy, and F. Li, "Afed-ef: An energy-efficient vm allocation algorithm for iot applications in a cloud data center," *IEEE Transactions on Green Communications and Networking*, vol. 5, no. 2, pp. 658–669, 2021. DOI: 10.1109/TGCN.2021.3067309.

[15] M. Mohammed Sadeeq, N. M. Abdulkareem, S. R. M. Zeebaree, D. Mikaeel Ahmed, A. Saifullah Sami, and R. R. Zebari, "Iot and cloud computing issues, challenges and opportunities: A review," *Qubahan Academic Journal*, vol. 1, no. 2, pp. 1–7, Mar. 2021. DOI: 10.48161/qaj.v1n2a36.

[16] J. Son and R. Buyya, "Latency-aware virtualized network function provisioning for distributed edge clouds," *Journal of Systems and Software*, vol. 152, pp. 24–31, 2019, ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2019.02.030.

[17] G. Kumar, N. Dukkipati, K. Jang, *et al.*, "Swift: Delay is simple and effective for congestion control in the datacenter," in *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '20, Virtual Event, USA: Association for Computing Machinery, 2020, pp. 514–528, ISBN: 9781450379557. DOI: 10.1145/3387514.3406591.

[18] W. Wang, M. Moshref, Y. Li, *et al.*, "Poseidon: Efficient, robust, and practical datacenter CC via deployable INT," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, Boston, MA: USENIX Association, Apr. 2023, pp. 255–274, ISBN: 978-1-939133-33-5.

[19] W. Xia, P. Zhao, Y. Wen, and H. Xie, "A survey on data center networking (dcn): Infrastructure and operations," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 1, pp. 640–656, 2017. DOI: 10.1109/COMST.2016.2626784.

[20] J. T. Moy, *OSPF: Anatomy Of An Internet Routing Protocol*. Addison-Wesley Professional, 1998.

[21] H. Gredler and W. Goralski, *The Complete IS-IS Routing Protocol*. Springer Science & Business Media, 2005.

[22] Y. Rekhter, T. Li, and S. Hares, "A border gateway protocol 4 (bgp-4)," Tech. Rep., 2006.

[23] T. Wang, Z. Su, Y. Xia, and M. Hamdi, "Rethinking the data center networking: Architecture, network protocols, and resource sharing," *IEEE Access*, vol. 2, pp. 1481–1496, 2014. DOI: 10.1109/ACCESS.2014.2383439.

[24] B. Dai, G. Xu, B. Huang, P. Qin, and Y. Xu, "Enabling network innovation in data center networks with software defined networking: A survey," *Journal of Network and Computer Applications*, vol. 94, pp. 33–49, 2017, ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2017.07.004.

[25] X. Fei, F. Liu, Q. Zhang, H. Jin, and H. Hu, "Paving the way for nfv acceleration: A taxonomy, survey and future directions," *ACM Comput. Surv.*, vol. 53, no. 4, Aug. 2020, ISSN: 0360-0300. DOI: 10.1145/3397022.

[26] N. McKeown, T. Anderson, H. Balakrishnan, *et al.*, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008, ISSN: 0146-4833. DOI: 10.1145/1355734.1355746.

[27] C. Guo, L. Yuan, D. Xiang, *et al.*, "Pingmesh: A large-scale system for data center network latency measurement and analysis," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, London United Kingdom: ACM, Aug. 2015, pp. 139–152, ISBN: 978-1-4503-3542-3. DOI: 10.1145/2785956.2787496.

[28] Y. Peng, J. Yang, C. Wu, C. Guo, C. Hu, and Z. Li, "Detector: A topology-aware monitoring system for data center networks," in *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, Santa Clara, CA: USENIX Association, Jul. 2017, pp. 55–68, ISBN: 978-1-931971-38-6.

[29] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, ser. SIGCOMM '08, Seattle, WA, USA: Association for Computing Machinery, 2008, pp. 63–74, ISBN: 978-1-60558-175-0. DOI: 10.1145/1402958.1402967.

[30] C. Guo, G. Lu, D. Li, *et al.*, "Bcube: A high performance, server-centric network architecture for modular data centers," in *Proceedings of the ACM SIGCOMM 2009 Conference*

on Data Communication - SIGCOMM '09, Barcelona, Spain: ACM Press, 2009, p. 63, ISBN: 978-1-60558-594-9. DOI: 10.1145/1592568.1592577.

[31] X. Wang, J.-X. Fan, C.-K. Lin, J.-Y. Zhou, and Z. Liu, "Bcdc: A high-performance, server-centric data center network," *Journal of Computer Science and Technology*, vol. 33, no. 2, pp. 400–416, Mar. 2018, ISSN: 1000-9000, 1860-4749. DOI: 10.1007/s11390-018-1826-3.

[32] Z. Chkirbene, R. Hadjidj, S. Foufou, and R. Hamila, "Lascada: A novel scalable topology for data center network," *IEEE/ACM Transactions on Networking*, vol. 28, no. 5, pp. 2051–2064, Oct. 2020, ISSN: 1063-6692, 1558-2566. DOI: 10.1109/TNET.2020.3008512.

[33] H. Feng, Y. Deng, X. Qin, and G. Min, "Criso: An incremental scalable and cost-effective network architecture for data centers," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 2016–2029, 2021. DOI: 10.1109/TNSM.2020.3036875.

[34] M. Al-Makhlafi, H. Gu, A. Almuaalemi, E. Almekhlafi, and M. M. Adam, "Ribsnet: A scalable, high-performance, and cost-effective two-layer-based cloud data center network architecture," *IEEE Trans. on Netw. and Serv. Manag.*, vol. 20, no. 2, pp. 1676–1690, Jun. 2023, ISSN: 1932-4537. DOI: 10.1109/TNSM.2022.3218127. [Online]. Available: https://doi.org/10.1109/TNSM.2022.3218127.

[35] G. Qu and W. Chen, "Constructing a large-scale data center network structure using regular graphs," in *2019 IEEE International Conferences on Ubiquitous Computing & Communications (IUCC) and Data Science and Computational Intelligence (DSCI) and Smart Computing, Networking and Services (SmartCNS)*, 2019, pp. 809–812. DOI: 10.1109/IUCC/DSCI/SmartCNS.2019.00164.

[36] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "Dcell: A scalable and fault-tolerant network structure for data centers," in *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, ser. SIGCOMM '08, Seattle, WA, USA: Association for Computing Machinery, 2008, p. 75.

[37] A. Greenberg, J. R. Hamilton, N. Jain, *et al.*, "Vl2: A scalable and flexible data center network," in *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, ser. SIGCOMM '09, Barcelona, Spain: Association for Computing Machinery, 2009, pp. 51–62, ISBN: 9781605585949. DOI: 10.1145/1592568.1592576.

[38] D. Lin, Y. Liu, M. Hamdi, and J. Muppala, "Hyper-bcube: A scalable data center network," in *2012 IEEE International Conference on Communications (ICC)*, 2012, pp. 2918–2923. DOI: 10.1109/ICC.2012.6363759.

[39] Y. Sun, Q. Liu, and W. Fang, "Diamond: An improved fat-tree architecture for large-scale data centers.," *Journal of Communication*, vol. 9, no. 1, pp. 91–98, 2014.

[40] D.-m. Yu, Z. Zhang, Y.-h. Deng, L.-x. Lin, T.-j. He, and G.-l. He, "Flexible, highly scalable and cost-effective network structures for data centers," *Journal of Network and Computer Applications*, vol. 210, p. 103 542, 2023, ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2022.103542.

[41] Z. Liu, A. Zhao, and M. Liang, "A port-based forwarding load-balancing scheduling approach for cloud datacenter networks," *Journal of Cloud Computing*, vol. 10, no. 1, p. 13, Dec. 2021, ISSN: 2192-113X. DOI: 10.1186/s13677-021-00226-w.

[42] E. Nepolo and G.-A. Lusilao Zodi, "A predictive ecmp routing protocol for fat-tree enabled data centre networks," in *2021 15th International Conference on Ubiquitous Information Management and Communication (IMCOM)*, 2021, pp. 1–8. DOI: 10.1109/IMCOM51814.2021.9377396.

[43] S. Hu, X. Wang, and Z. Shi, "A software defined network scheme for intra datacenter network based on fat-tree topology," *Journal of Physics: Conference Series*, vol. 2025, no. 1, p. 012 106, Sep. 2021. DOI: 10.1088/1742-6596/2025/1/012106.

[44] W. Lin, X.-Y. Li, J.-M. Chang, and X. Jia, "Constructing multiple cists on bcube-based data center networks in the occurrence of switch failures," *IEEE Transactions on Computers*, vol. 72, no. 7, pp. 1971–1984, 2023. DOI: 10.1109/TC.2022.3230288.

[45] W. Fan, F. Xiao, H. Cai, X. Chen, and S. Yu, "Disjoint paths construction and fault-tolerant routing in bcube of data center networks," *IEEE Transactions on Computers*, vol. 72, no. 9, pp. 2467–2481, 2023. DOI: 10.1109/TC.2023.3251849.

[46] M. Lv, J. Fan, W. Fan, and X. Jia, "Fault diagnosis based on subsystem structures of data center network bcube," *IEEE Transactions on Reliability*, vol. 71, no. 2, pp. 963–972, 2022. DOI: 10.1109/TR.2021.3140069.

[47] B. Lan, F. Lei, K. Wu, and D. Dong, "Dfr: Dynamic-thresold fault-tolerant routing for fat tree," in *Proceedings of the 7th Asia-Pacific Workshop on Networking*, ser. APNET '23, New York, NY, USA: Association for Computing Machinery, 2023, pp. 180–181, ISBN: 9798400707827. DOI: 10.1145/3600061.3603125.

[48] D. D. Robin and J. I. Khan, "P4te: Pisa switch based traffic engineering in fat-tree data center networks," *Computer Networks*, vol. 215, p. 109 210, 2022, ISSN: 1389-1286. DOI: 10.1016/j.comnet.2022.109210.

[49] X. Zhou, X. Peng, T. Xie, *et al.*, "Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study," *IEEE Transactions on Software Engineering*, vol. 47, no. 2, pp. 243–260, 2021. DOI: 10.1109/TSE.2018.2887384.

[50] A. Roy, H. Zeng, J. Bagga, and A. C. Snoeren, "Passive realtime datacenter fault detection and localization," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, Boston, MA: USENIX Association, Mar. 2017, pp. 595–612, ISBN: 978-1-931971-37-9.

[51] A. Roy, R. Das, H. Zeng, J. Bagga, and A. C. Snoeren, "Understanding the limits of passive realtime datacenter fault detection and localization," *IEEE/ACM Transactions on Networking*, vol. 27, no. 5, pp. 2001–2014, 2019. DOI: 10.1109/TNET.2019.2938228.

[52] M. Cociglio, G. Fioccola, G. Marchetto, A. Sapio, and R. Sisto, "Multipoint passive monitoring in packet networks," *IEEE/ACM Transactions on Networking*, vol. 27, no. 6, pp. 2377–2390, 2019. DOI: 10.1109/TNET.2019.2950157.

[53] X. Gang, W. Zhan, Z. Dawei, and A. Xuejun, "Anomaly detection algorithm of data center network based on lsdb," *Journal of Computer Research and Development*, vol. 55, no. 4, pp. 815–830, 2018, ISSN: 1000-1239. DOI: 10.7544/issn1000-1239.2018.20160970.

[54] P. K. Pandey, B. Adhikari, and S. Chakraborty, "A diffusion protocol for detection of link failure and utilization of resources in multi-agent systems," *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 3, pp. 1493–1507, 2020. DOI: 10.1109/TNSE.2019.2936468.

[55] D. A. Popescu and A. W. Moore, "Measuring network conditions in data centers using the precision time protocol," *IEEE Transactions on Network and Service Management*, vol. 18, no. 3, pp. 3753–3770, 2021. DOI: 10.1109/TNSM.2021.3081536.

[56] H. Hao, R. Yu, and J. Guo, "A Probe Set Determination Method Based on Spanning Tree Algorithm," in *Proceedings of the 2023 7th International Conference on High Performance Compilation, Computing and Communications*, Jinan China: ACM, Jun. 2023, pp. 15–20, ISBN: 978-1-4503-9988-3. DOI: 10.1145/3606043.3606046.

[57] K. Xie, J. Tian, G. Xie, G. Zhang, and D. Zhang, "Low cost sparse network monitoring based on block matrix completion," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, IEEE, 2021, pp. 1–10.

[58] Y. Lin, Y. Zhou, Z. Liu, *et al.*, "Netview: Towards on-demand network-wide telemetry in the data center," *Computer Networks*, vol. 180, p. 107 386, 2020, ISSN: 1389-1286. DOI: 10.1016/j.comnet.2020.107386.

[59] C. Jia, T. Pan, Z. Bian, *et al.*, "Rapid detection and localization of gray failures in data centers via in-band network telemetry," in *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, 2020, pp. 1–9. DOI: 10.1109/NOMS47738.2020.9110326.

[60] S. Xie, G. Hu, C. Xing, J. Zu, and Y. Liu, "Fint: Flexible in-band network telemetry method for data center network," *Computer Networks*, vol. 216, p. 109 232, 2022, ISSN: 1389-1286. DOI: 10.1016/j.comnet.2022.109232.

[61] Y. Su, "P4-based grey failure detection and location in fat-tree data center networks," Ph.D. dissertation, University of Electronic Science and Technology of China, Chengdu, 2022.

[62] W. Stallings, *SNMP, SNMPv2, and CMIP: The Practical Guide to Network Management*. Addison-Wesley Longman Publishing Co., Inc., 1993.

[63] R. Sommer and A. Feldmann, "Netflow: Information loss or win?" In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurment*, 2002, pp. 173–174.

[64] J. Wang, H. Qi, K. Li, and X. Zhou, "Real-time link fault detection as a service for datacenter network," *Journal of Computer Research and Development*, vol. 55, no. 04, pp. 704–716, 2018.

[65] L. Tan, W. Su, W. Zhang, H. Shi, J. Miao, and P. Manzanares-Lopez, "A packet loss monitoring system for in-band network telemetry: Detection, localization, diagnosis and recovery," *IEEE Transactions on Network and Service Management*, vol. 18, no. 4, pp. 4151–4168, 2021. DOI: 10.1109/TNSM.2021.3125012.

[66] S. Vladyslav, "The traffic engineering in data center networks based on inband network telemetry," Ph.D. dissertation, Beijing University of Posts and Telecommunications, Beijing, 2022.

[67] G. Bankhamer, R. Elsässer, and S. Schmid, "Local fast rerouting with low congestion: A randomized approach," *IEEE/ACM Transactions on Networking*, vol. 30, no. 6, pp. 2403–2418, 2022. DOI: 10.1109/TNET.2022.3174731.

[68] F. L. Verdi and G. V. Luz, "Infarr: In-network fast rerouting," *IEEE Transactions on Network and Service Management*, vol. 20, no. 3, pp. 2319–2330, 2023. DOI: 10.1109/TNSM.2023.3283459.

[69] K.-T. Foerster, A. Kamisinski, Y.-A. Pignolet, S. Schmid, and G. Tredan, "Improved fast rerouting using postprocessing," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 1, pp. 537–550, 2022. DOI: 10.1109/TDSC.2020.2998019.

[70] K. H. K. Reddy, A. K. Luhach, V. V. Kumar, S. Pratihar, D. Kumar, and D. S. Roy, "Towards energy efficient smart city services: A software defined resource management scheme for data centers," *Sustainable Computing: Informatics and Systems*, vol. 35, p. 100 776, 2022, ISSN: 2210-5379. DOI: 10.1016/j.suscom.2022.100776.

[71] Z. Guo, Y. Xu, Y.-F. Liu, *et al.*, "Aggreflow: Achieving power efficiency, load balancing, and quality of service in data center networks," *IEEE/ACM Transactions on Networking*, vol. 29, no. 1, pp. 17–33, 2021. DOI: 10.1109/TNET.2020.3026015.

[72] N. Rikhtegar, M. Keshtgari, O. Bushehrian, and G. Pujolle, "Bite: A dynamic bi-level traffic engineering model for load balancing and energy efficiency in data center networks," *Applied Intelligence*, vol. 51, no. 7, pp. 4623–4648, Jul. 2021, ISSN: 1573-7497. DOI: 10.1007/s10489-020-02003-9.

[73] P. Kamboj and S. Pal, "Energy-aware routing in sdn enabled data center network," in *2022 IEEE 21st International Symposium on Network Computing and Applications (NCA)*, vol. 21, 2022, pp. 123–130. DOI: 10.1109/NCA57778.2022.10013588.

[74] R. Dai, H. Li, and X. Fu, "Data center flow scheduling mechanism based on differential evolution and ant colony optimization algorithm," *Journal of Computer Applications*, vol. 42, no. 12, pp. 3863–3869, 2022.

[75] D. M. Casas-Velasco, O. M. C. Rendon, and N. L. S. Da Fonseca, "Intelligent routing based on reinforcement learning for software-defined networking," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 870–881, Mar. 2021, ISSN: 1932-4537, 2373-7379. DOI: 10.1109/TNSM.2020.3036911. (visited on 05/23/2023).

[76] J. Zhang, M. Ye, Z. Guo, C.-Y. Yen, and H. J. Chao, "Cfr-rl: Traffic engineering with reinforcement learning in sdn," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 10, pp. 2249–2259, Oct. 2020, ISSN: 0733-8716, 1558-0008. DOI: 10.1109/JSAC.2020.3000371. (visited on 07/22/2022).

[77] R. Jin, J. Li, X. Tuo, W. Wang, and X. Li, "A congestion control method of sdn data center based on reinforcement learning," *International Journal of Communica-*

tion Systems, vol. 31, no. 17, e3802, 2018. DOI: 10.1002/dac.3802. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/dac.3802.

[78] P. Sun, Z. Guo, S. Liu, J. Lan, J. Wang, and Y. Hu, "Smartfct: Improving power-efficiency for data center networks with deep reinforcement learning," *Computer Networks*, vol. 179, p. 107 255, 2020, ISSN: 1389-1286. DOI: 10.1016/j.comnet.2020.107255.

[79] D. M. Casas-Velasco, O. M. C. Rendon, and N. L. S. Da Fonseca, "Drsir: A deep reinforcement learning approach for routing in software-defined networking," *IEEE Transactions on Network and Service Management*, vol. 19, no. 4, pp. 4807–4820, Dec. 2022, ISSN: 1932-4537, 2373-7379. DOI: 10.1109/TNSM.2021.3132491. (visited on 06/28/2023).

[80] W.-X. Liu, J. Lu, J. Cai, Y. Zhu, S. Ling, and Q. Chen, "Drl-plink: Deep reinforcement learning with private link approach for mix-flow scheduling in software-defined data-center networks," *IEEE Transactions on Network and Service Management*, vol. 19, no. 2, pp. 1049–1064, Jun. 2022, ISSN: 1932-4537, 2373-7379. DOI: 10.1109/TNSM.2021.3128267. (visited on 01/16/2023).

[81] X. Pei, P. Sun, Y. Hu, D. Li, B. Chen, and L. Tian, "Enabling efficient routing for traffic engineering in sdn with deep reinforcement learning," *Computer Networks*, vol. 241, p. 110 220, 2024, ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2024.110220.

[82] L. Chen, J. Lingys, K. Chen, and F. Liu, "Auto: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, 2018, pp. 191–205.

[83] W.-X. Liu, J. Cai, Y. Wang, Q. C. Chen, and D. Tang, "Mix-flow scheduling using deep reinforcement learning for software-defined data-center networks," *Internet Technology Letters*, vol. 2, no. 3, e99, 2019. DOI: 10.1002/itl2.99. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/itl2.99.

[84] Y. Tang, H. Guo, T. Yuan, *et al.*, "Flow splitter: A deep reinforcement learning-based flow scheduler for hybrid optical-electrical data center network," *IEEE Access: Practical*

Innovations, Open Solutions, vol. 7, pp. 129 955–129 965, 2019. DOI: 10.1109/ACCESS.2019.2940445.

[85] Y. Gao, X. Gao, and G. Chen, "Metisrl: A reinforcement learning approach for dynamic routing in data center networks," in *Database Systems for Advanced Applications*, A. Bhattacharya, J. Lee Mong Li, D. Agrawal, *et al.*, Eds., vol. 13246, Cham: Springer International Publishing, 2022, pp. 615–622, ISBN: 978-3-031-00125-3 978-3-031-00126-0. DOI: 10.1007/978-3-031-00126-0_44. (visited on 01/12/2023).

[86] J. Zhao, S. Zhang, Y. Zhang, L. Zhang, and H. Long, "Deep reinforcement learning-based routing optimization algorithm for edge data center," in *2021 IEEE Symposium on Computers and Communications (ISCC)*, Athens, Greece: IEEE, Sep. 2021, pp. 1–7, ISBN: 978-1-66542-744-9. DOI: 10.1109/ISCC53001.2021.9631254. (visited on 01/16/2023).

[87] Y.-R. Chen, A. Rezapour, W.-G. Tzeng, and S.-C. Tsai, "Rl-routing: An sdn routing algorithm based on deep reinforcement learning," *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 4, pp. 3185–3199, Oct. 2020, ISSN: 2327-4697, 2334-329X. DOI: 10.1109/TNSE.2020.3017751. (visited on 10/07/2022).

[88] Z. Jia, Y. Sun, Q. Liu, S. Dai, and C. Liu, "Cretor: An sdn-based routing scheme for data centers with regular topologies," *IEEE Access: Practical Innovations, Open Solutions*, vol. 8, pp. 116 866–116 880, 2020. DOI: 10.1109/ACCESS.2020.3004609.

[89] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, 2013. DOI: 10.1109/MCOM.2013.6461195.

[90] T. D. Nadeau and K. Gray, *SDN: Software Defined Networks: An Authoritative Review of Network Programmability Technologies*. O'Reilly Media, Inc., 2013.

[91] C. Caba and J. Soler, "Mitigating sdn controller performance bottlenecks," in *2015 24th International Conference on Computer Communication and Networks (ICCCN)*, 2015, pp. 1–6. DOI: 10.1109/ICCCN.2015.7288429.

[92] E. Akanksha, Jyoti, N. Sharma, and K. Gulati, "Review on reinforcement learning, research evolution and scope of application," in *2021 5th International Conference on Computing Methodologies and Communication (ICCMC)*, 2021, pp. 1416–1423. DOI: 10.1109/ICCMC51019.2021.9418283.

[93] E. Levin, R. Pieraccini, and W. Eckert, "Using markov decision process for learning dialogue strategies," in *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP '98 (cat. No.98CH36181)*, vol. 1, 1998, 201–204 vol.1. DOI: 10.1109/ICASSP.1998.674402.

[94] J. Clifton and E. Laber, "Q-learning: Theory and applications," *Annual Review of Statistics and Its Application*, vol. 7, pp. 279–301, 2020.

[95] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017. DOI: 10.1109/MSP.2017.2743240.

[96] N. C. Luong, D. T. Hoang, S. Gong, *et al.*, "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3133–3174, 2019. DOI: 10.1109/COMST.2019.2916583.

[97] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv Preprint arXiv:1707.06347*, 2017.

[98] M. Gori, G. Monfardini, and F. Scarselli, "A new model for learning in graph domains," in *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, vol. 2, 2005, 729–734 vol. 2. DOI: 10.1109/IJCNN.2005.1555942.

[99] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009. DOI: 10.1109/TNN.2008.2005605.

[100] J. Bruna, W. Zaremba, A. Szlam, and Y. Lecun, "Spectral networks and locally connected networks on graphs," English (US), in *International Conference on Learning Representations (ICLR2014), CBLS, April 2014*, 2014.

[101] M. Tang, B. Li, and H. Chen, "Application of message passing neural networks for molecular property prediction," *Current Opinion in Structural Biology*, vol. 81, p. 102 616, 2023.

[102] P. Tam, I. Song, S. Kang, S. Ros, and S. Kim, "Graph neural networks for intelligent modelling in network management and orchestration: A survey on communications," *Electronics*, vol. 11, no. 20, p. 3371, 2022.

[103] T. Liu, A. Jiang, J. Zhou, M. Li, and H. K. Kwan, "Graphsage-based dynamic spatial–temporal graph convolutional network for traffic prediction," *IEEE Transactions on Intelligent Transportation Systems*, 2023.

[104] C. Wan, Y. Li, A. Li, N. S. Kim, and Y. Lin, "Bns-gcn: Efficient full-graph training of graph convolutional networks with partition-parallelism and random boundary node sampling," *Proceedings of Machine Learning and Systems*, vol. 4, pp. 673–693, 2022.

[105] S. Brody, U. Alon, and E. Yahav, "How attentive are graph attention networks?" *arXiv preprint arXiv:2105.14491*, 2022.

[106] A. Singla, P. B. Godfrey, and A. Kolla, "High throughput data center topology design," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, Seattle, WA: USENIX Association, Apr. 2014, pp. 29–41, ISBN: 978-1-931971-09-6.

[107] S. I. Alliance, "Whitepaper on sdn controller performance in data center scenario (in chinese)," SDN/NFV Industry Alliance, Tech. Rep., 2017.

[108] Y. Liu, J. K. Muppala, M. Veeraraghavan, D. Lin, and M. Hamdi, *Data Center Networks: Topologies, Architectures and Fault-Tolerance Characteristics*. Springer Science & Business Media, 2013.

[109] J. W. Guck, A. Van Bemten, M. Reisslein, and W. Kellerer, "Unicast qos routing algorithms for sdn: A comprehensive survey and performance evaluation," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 1, pp. 388–415, 2018, ISSN: 1553-877X. DOI: 10.1109/COMST.2017.2749760.

[110] P. Zeng, Y. Shen, Z. Qiu, Z. Qiu, and M. Guo, "Srp: A routing protocol for data center networks," in *The 16th Asia-Pacific Network Operations and Management Symposium*, Hsinchu: IEEE, Sep. 2014, pp. 1–6, ISBN: 978-4-88552-288-8. DOI: 10.1109/APNOMS.2014.6996564.

[111] M. N. Pathan, M. Muntaha, S. Sharmin, *et al.*, "Priority based energy and load aware routing algorithms for SDN enabled data center network," *Computer Networks*, vol. 240, p. 110 166, Feb. 2024, ISSN: 13891286. DOI: 10.1016/j.comnet.2023.110166.

[112] *Ryu sdn framework*. [Online]. Available: https://ryu-sdn.org.

[113] *Open vswitch*. [Online]. Available: http://www.openvswitch.org/.

[114] T. Parr, *Antlr (another tool for language recognition)*. [Online]. Available: https://www.antlr.org/index.html.

[115] *Mininet: An instant virtual network on your laptop (or other pc)*. [Online]. Available: http://mininet.org/.

[116] *Quagga routing suite*. [Online]. Available: https://www.nongnu.org/quagga/index.html.

[117] *Dcell data center network structure implemented with software-defined networking (sdn)*. [Online]. Available: https://github.com/chuyangliu/dcell.

[118] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using networkx," in *Proceedings of the 7th python in science conference*, G. el Varoquaux, T. Vaught, and J. Millman, Eds., Pasadena, CA USA, 2008, pp. 11–15.

[119] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: Measurement, analysis, and implications," in *Proceedings of the ACM SIGCOMM 2011 Conference*, 2011, pp. 350–361.

[120] L. D. Ghein, *Change of default ospf and is-is spf and flooding timers and ispf removal*. [Online]. Available: https://www.cisco.com/c/en/us/support/docs/ip/ip-routing/211432-Change-of-Default-OSPF-and-IS-IS-SPF-and.html.

[121] A. Mathew, M. Srinivasan, and C. S. R. Murthy, "Packet generation schemes and network latency implications in sdn-enabled 5g c-rans: Queuing model based analysis," in *2019 IEEE 30th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, Istanbul, Turkey: IEEE, Sep. 2019, pp. 1–7, ISBN: 978-1-5386-8110-7. DOI: 10.1109/PIMRC.2019.8904151.

[122] A. Darabseh, M. Al-Ayyoub, Y. Jararweh, E. Benkhelifa, M. Vouk, and A. Rindos, "Sddc: A software defined datacenter experimental framework," in *2015 3rd International Conference on Future Internet of Things and Cloud*, 2015, pp. 189–194. DOI: 10.1109/FiCloud.2015.127.

[123] C.-R. Lin, Y.-J. Chen, and L.-C. Wang, "Handoff delay analysis in sdn-enabled mobile networks: A network calculus approach," in *2017 IEEE 86th Vehicular Technology Conference (VTC-Fall)*, Toronto, ON: IEEE, Sep. 2017, pp. 1–5, ISBN: 978-1-5090-5935-5. DOI: 10.1109/VTCFall.2017.8288202.

[124] B. Vidalenc, L. Noirie, S. Ghamri-Doudane, and E. Renault, "Adaptive failure detection timers for igp networks," in *2013 IFIP Networking Conference*, 2013, pp. 1–9.

[125] D. Kotani and Y. Okabe, "Packet-in message control for reducing cpu load and control traffic in openflow switches," in *2012 European Workshop on Software Defined Networking*, Darmstadt, 2012, pp. 42–47. DOI: 10.1109/EWSDN.2012.23.

[126] Shie-Yuan Wang, Chih-Liang Chou, and Chun-Ming Yang, "Estinet openflow network simulator and emulator," *IEEE Communications Magazine*, vol. 51, no. 9, pp. 110–117, Sep. 2013, ISSN: 0163-6804. DOI: 10.1109/MCOM.2013.6588659.

[127] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15, New York, NY, USA: Association for Computing Machinery, 2015, pp. 123–137, ISBN: 978-1-4503-3542-3. DOI: 10.1145/2785956.2787472.

[128] Z. Eitan, *Dctg data center traffic generator library*, 2018. [Online]. Available: https://github.com/Mellanox/DCTrafficGen.

[129] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th Annual Conference on Internet Measurement - IMC '10*, Melbourne, Australia: ACM Press, 2010, p. 267, ISBN: 978-1-4503-0483-2. DOI: 10.1145/1879141.1879175.

[130] M. Khairy, A. Alawneh, A. Barnes, and T. G. Rogers, "SIMR: Single instruction multiple request processing for energy-efficient data center microservices," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022, pp. 441–463. DOI: 10.1109/MICRO56248.2022.00040.

[131] R. Ranjan, "Streaming big data processing in datacenter clouds," *IEEE Cloud Computing*, vol. 1, no. 01, pp. 78–83, May 2014, ISSN: 2372-2568. DOI: 10.1109/MCC.2014.22.

[132] J. Park, M. Naumov, P. Basu, *et al.*, "Deep learning inference in facebook data centers: Characterization, performance optimizations and hardware implications," *arXiv preprint arXiv:1811.09886*, 2018.

[133] H. Wang, P. Nguyen, J. Li, *et al.*, "Grano: Interactive graph-based root cause analysis for cloud-native distributed data platform," *Proc. VLDB Endow.*, vol. 12, no. 12, pp. 1942–1945, Aug. 2019, ISSN: 2150-8097. DOI: 10.14778/3352063.3352105.

[134] Z. Chen, J. Xu, T. Peng, and C. Yang, "Graph convolutional network-based method for fault diagnosis using a hybrid of measurement and prior knowledge," *IEEE Transactions on Cybernetics*, vol. 52, no. 9, pp. 9157–9169, 2022. DOI: 10.1109/TCYB.2021.3059002.

[135] X. Qi, J. Li, Z. Wang, and L. Liu, "Probabilistic probe selection algorithm for fault diagnosis in communication networks," *Computer Networks*, vol. 198, p. 108 365, 2021, ISSN: 1389-1286. DOI: 10.1016/j.comnet.2021.108365.

[136] X. Guo, X. Peng, H. Wang, *et al.*, "Graph-based trace analysis for microservice architecture understanding and problem diagnosis," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 1387–1397.

[137] D. Bhamare, A. Kassler, J. Vestin, *et al.*, "Intopt: In-band network telemetry optimization framework to monitor network slices using p4," *Computer Networks*, vol. 216, p. 109 214, 2022, ISSN: 1389-1286. DOI: 10.1016/j.comnet.2022.109214.

[138] C. Kim, A. Sivaraman, N. Katta, A. Bas, A. Dixit, and L. J. Wobker, "In-band network telemetry via programmable dataplanes," in *ACM SIGCOMM*, vol. 15, 2015.

[139] S. Narayana, A. Sivaraman, V. Nathan, *et al.*, "Language-directed hardware design for network performance monitoring," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17, New York, NY, USA: Association for Computing Machinery, 2017, pp. 85–98, ISBN: 978-1-4503-4653-5. DOI: 10.1145/3098822.3098829.

[140] T. Pan, X. Lin, H. Song, *et al.*, "Int-probe: Lightweight in-band network-wide telemetry with stationary probes," in *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, 2021, pp. 898–909. DOI: 10.1109/ICDCS51616.2021.00090.

[141] S. R. Saufi, Z. A. B. Ahmad, M. S. Leong, and M. H. Lim, "Challenges and opportunities of deep learning models for machinery fault detection and diagnosis: A review," *IEEE Access : Practical Innovations, Open Solutions*, vol. 7, pp. 122 644–122 662, 2019. DOI: 10.1109/ACCESS.2019.2938227.

[142] Y. Bejerano and Rajeev Rastogi, "Robust monitoring of link delays and faults in ip networks," in *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No.03CH37428)*, vol. 1, San Francisco, CA, USA: IEEE, 2003, pp. 134–144, ISBN: 978-0-7803-7752-3. DOI: 10.1109/INFCOM.2003.1208666.

[143] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev, "Reinforcement learning for combinatorial optimization: A survey," *Computers and Operations Research*, vol. 134, p. 105 400, 2021, ISSN: 0305-0548. DOI: https://doi.org/10.1016/j.cor.2021.105400.

[144] N. Saha, M. Zangooei, M. Golkarifard, and R. Boutaba, "Deep reinforcement learning approaches to network slice scaling and placement: A survey," *IEEE Communications Magazine*, vol. 61, no. 2, pp. 82–87, 2023. DOI: 10.1109/MCOM.006.2200534.

[145] S. Manna, T. D. Loeffler, R. Batra, *et al.*, "Learning in continuous action space for developing high dimensional potential energy models," *Nature Communications*, vol. 13, no. 1, p. 368, Jan. 2022, ISSN: 2041-1723. DOI: 10.1038/s41467-021-27849-6.

[146] W. Xia, P. Zhao, Y. Wen, and H. Xie, "A Survey on Data Center Networking (DCN): Infrastructure and Operations," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 1, pp. 640–656, 2017. DOI: 10.1109/COMST.2016.2626784.

[147] C. Dong, W. Wen, T. Xu, and X. Yang, "Joint Optimization of Data-Center Selection and Video-Streaming Distribution for Crowdsourced Live Streaming in a Geo-Distributed Cloud Platform," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 729–742, 2019. DOI: 10.1109/TNSM.2019.2907785.

[148] B. He, X. Zheng, Y. Chen, *et al.*, "DxPU: Large Scale Disaggregated GPU Pools in the Datacenter," *ACM Trans. Archit. Code Optim.*, Oct. 2023, Just Accepted, ISSN: 1544-3566. DOI: 10.1145/3617995. [Online]. Available: https://doi.org/10.1145/3617995.

[149] G. S. Begam, M. Sangeetha, and N. R. Shanker, "Load balancing in dcn servers through sdn machine learning algorithm," *Arabian Journal for Science and Engineering*, vol. 47, pp. 1423–1434, 2021.

[150] S. Bharany, S. Badotra, S. Sharma, *et al.*, "Energy efficient fault tolerance techniques in green cloud computing: A systematic survey and taxonomy," *Sustainable Energy Technologies and Assessments*, vol. 53, p. 102 613, 2022, ISSN: 2213-1388. DOI: https://doi.org/10.1016/j.seta.2022.102613. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2213138822006634.

[151] Y. He, B. A. Jayawickrama, E. Dutkiewicz, S. Srikanteswara, and M. Mueck, "Priority access and general authorized access interference mitigation in the spectrum access system," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 6, pp. 4969–4983, 2018.

[152] E. Dutkiewicz, B. A. Jayawickrama, and Y. He, "Radio spectrum maps for emerging iot and 5g networks: Applications to smart buildings," in *2017 International Conference on Electrical Engineering and Computer Science (ICECOS)*, IEEE, 2017, pp. 7–9.

[153] Y. He, E. Dutkiewicz, G. Fang, and M. D. Mueck, "Licensed shared access in distributed antenna systems enabling network virtualization," in *1st International Conference on 5G for Ubiquitous Connectivity*, IEEE, 2014, pp. 76–80.

[154] J. Li, P. Sun, and Y. Hu, "Traffic modeling and optimization in datacenters with graph neural network," *Computer Networks*, vol. 181, p. 107 528, 2020, ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2020.107528. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128620311865.

[155] J. Xie, F. R. Yu, T. Huang, *et al.*, "A survey of machine learning techniques applied to software defined networking (sdn): Research issues and challenges," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 393–430, 2019. DOI: 10.1109/COMST.2018.2866942.

[156] M. MAJDOUB, A. EL KAMEL, and H. YOUSSEF, "Dqr: An efficient deep q-based routing approach in multi-controller software defined wan (sd-wan)," *Journal of Interconnection Networks*, vol. 20, no. 04, p. 2 150 002, 2020. DOI: 10.1142/S021926592150002X. eprint: https://doi.org/10.1142/S021926592150002X.

[157] S. Munikoti, D. Agarwal, L. Das, M. Halappanavar, and B. Natarajan, "Challenges and opportunities in deep reinforcement learning with graph neural networks: A comprehensive review of algorithms and applications," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–21, 2023. DOI: 10.1109/TNNLS.2023.3283523.

[158] P. Almasan, J. Suárez-Varela, K. Rusek, P. Barlet-Ros, and A. Cabellos-Aparicio, "Deep reinforcement learning meets graph neural networks: Exploring a routing optimization use case," *Computer Communications*, vol. 196, pp. 184–194, Dec. 2022.

[159] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, p. 61, 2009.

[160] B. Yan, Q. Liu, J. Shen, and D. Liang, "Flowlet-level multipath routing based on graph neural network in openflow-based sdn," *Future Generations Computer Systems: FGCS*, p. 134, 2022.

[161] W. Jiang, "Graph-based deep learning for communication networks: A survey," *Computer Communications*, vol. 185, pp. 40–54, 2022.

[162] W. Quattrociocchi, G. Caldarelli, and A. Scala, "Self-healing networks: Redundancy and structure," *PLOS ONE*, vol. 9, no. 2, pp. 1–7, Feb. 2014. DOI: 10.1371/journal.pone.0087986.