

Article

Research on a Computing First Network Based on Deep Reinforcement Learning

Qianwen Xu ^{1,2}, Jingchao Wang ^{3,*}, Shuangyin Ren ^{4,*}, Zhongbo Li ⁴ and Wei Gao ⁴

¹ Institute of Systems Engineering, Military Academy of Science, Beijing 100141, China; xuqianwen2024@126.com

² The 63880 Troops of PLA, Luoyang 471000, China

³ Peng Cheng Laboratory, Shenzhen 518000, China

⁴ The 32008 Troops of PLA, Beijing 100141, China

* Correspondence: wangjc.2000@tsinghua.org.cn (J.W.); renshuangyin@126.com (S.R.)

Abstract

The joint optimization of computing resources and network routing constitutes a central challenge in Computing First Networks (CFNs). However, existing research has predominantly focused on computation offloading decisions, whereas the cooperative optimization of computing power and network routing remains underexplored. Therefore, this study investigates the joint routing optimization problem within the CFN framework. We first propose a computing resource scheduling architecture for CFN, termed SICRSA, which integrates Software-Defined Networking (SDN) and Information-Centric Networking (ICN). Building upon this architecture, we further introduce an ICN-based hierarchical naming scheme for computing services, design a computing service request packet format that extends the IP header, and detail the corresponding service request identification process and workflow. Furthermore, we propose Computing-Aware Routing via Graph and Long-term Dependency Learning (CRGLD), a Graph Neural Network (GNN), and Long Short-Term Memory (LSTM)-based routing optimization algorithm, within the SICRSA framework to address the computing-aware routing (CAR) problem. The algorithm incorporates a decision-making framework grounded in spatiotemporal feature learning, thereby enabling the joint and coordinated selection of computing nodes and transmission paths. Simulation experiments conducted on real-world network topologies demonstrate that CRGLD enhances both the quality of service and the intelligence of routing decisions in dynamic network environments. Moreover, CRGLD exhibits strong generalization capability when confronted with unfamiliar topologies and topological changes, effectively mitigating the poor generalization performance typical of traditional Deep Reinforcement Learning (DRL)-based routing models in dynamic settings.

Keywords: Computing First Networks; computing-aware routing; Graph Neural Network; Long Short-Term Memory; resource scheduling; Software-Defined Networking



Academic Editor: Valentina Emilia Balas

Received: 11 December 2025

Revised: 19 January 2026

Accepted: 21 January 2026

Published: 2 February 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

Driven by innovations in digital twins, autonomous vehicles, and artificial intelligence, a wide array of new services and applications continues to emerge. These novel services impose stringent requirements on computational power and latency, while also triggering an explosive growth in data volume [1]. Such a dramatic surge in data presents unprecedented challenges to cloud computing data centers. In this context, edge computing has emerged as a distributed computing paradigm. The network architecture has consequently

evolved from a single, large-scale cloud data center to widely distributed multi-access edge computing servers, and further to mobile intelligent devices, forming a collaborative three-tier architecture encompassing cloud, edge, and end devices. This architecture effectively alleviates the pressure on cloud data centers by extending computational capabilities to the network edge, thereby enabling local data processing. However, in current systems, the computational capacity of individual nodes remains limited, and nodes lack the ability to perceive and coordinate with one another, leading to issues such as low resource utilization and inefficient scheduling.

While cloud–network integration has realized the coordinated delivery of “cloud” and “network” resources, this model still treats computing and networking as separate optimization domains, hindering the achievement of the global optimum. Meanwhile, the evolution of future networks demands that computing and networking be tightly coupled—capable of mutual awareness and efficient collaboration. Motivated by these considerations, CFN has been proposed as a promising paradigm. CFN represents a new type of network that flexibly orchestrates computing, storage, and network resources of service nodes through a centralized control plane, enabling efficient resource sharing across nodes. By interconnecting distributed computing nodes within the network, CFN demonstrates significant potential for on-demand task scheduling and flexible resource sharing in computing systems [2].

The resource scheduling architecture constitutes the core of CFN, responsible for the awareness, allocation, and optimization of global resources. A well-designed architecture directly determines system throughput, response latency, and resource utilization, while also accommodating the diverse requirements across different application scenarios. Currently, with the rapid expansion of cloud–edge–end computing nodes and the increasing diversification of user service demands, existing CFN architectures built upon traditional TCP/IP networks often adopt centralized resource scheduling. This approach typically relies on a unified computing resource management center to maintain and update global resource status information, which suffers from limitations in scheduling flexibility and scalability. Furthermore, in the event of failures in the centralized management entity, the asynchronous updating of resource status can easily lead to scheduling breakdowns. Such centralized methods lack both scalability and fault tolerance, and cannot effectively handle faults in resource scheduling scenarios, thereby increasing the risk of service disruptions. In contrast, fully distributed scheduling approaches face challenges in terms of coordination complexity and consistency assurance, often struggling to achieve timely global optimization decisions.

As a core enabling technology of CFN, CAR has garnered significant attention from both academia and industry in recent years. Extending beyond traditional routing mechanisms, CAR integrates multidimensional resource metrics—such as computational capacity, network bandwidth, and latency—into the routing decision process, aiming to achieve integrated scheduling and coordination of computing and network resources. From a computational perspective, the real-time, on-demand, and dynamic scheduling across entire network domains constitutes an NP-hard problem. The “state-space explosion” inherent in large-scale complex networks further prevents the system from obtaining real-time optimal solutions efficiently. Current control schemes for CFN primarily fall into centralized, distributed, and hybrid categories. Regardless of the adopted scheme, it remains essential to select suitable computing nodes and determine optimal transmission paths for computational tasks based on the status of both computing and network resources. Consequently, the central challenge for CFN lies in achieving globally coordinated scheduling of computing and network resources.

However, most existing studies have focused on localized optimization—such as computational offloading—and have yet to realize deep, systemic coupling between network routing and computational demands. The decision-making process of computing power routing needs to comprehensively consider multiple factors such as the status of computing resources, network performance, and task characteristics. Traditional routing algorithms are unable to achieve joint optimization of computing resources and network status, making it difficult to achieve global optimality for the “computing node-transmission path.” In practice, computing networks are subject to continuous topological evolution due to node additions or failures, link variations, and changes in traffic patterns. Static routing schemes or those optimized only for local conditions fail to adapt in real time to fluctuating resource availability, shifting workloads, and heterogeneous service requirements. Although DRL shows potential in routing optimization, traditional routing algorithms based on DRL have the problem of weak generalization ability, especially when facing unknown topologies and changes in topologies, they often require a large amount of re-training to maintain performance.

In this study, we propose a computing resource scheduling architecture for CFN, named SICRSA, which integrates SDN and ICN. Building upon this architecture, we introduce a hierarchical ICN-based naming scheme for computing services, design an IP-header-extended packet format for computing service requests, and detail the corresponding service request identification procedure. Furthermore, we develop a GNN- and LSTM-based routing optimization algorithm, termed CR-GLD, within the SICRSA framework. This algorithm facilitates intelligent routing decisions by holistically integrating computational and network resource states. Simulation results indicate that CR-GLD outperforms several classical intelligent routing algorithms in performance metrics and demonstrates stronger generalization capability across diverse network topologies and under dynamic topological changes.

The main contributions of this work are summarized as follows:

- (1) To address the limitations of existing CFN resource scheduling architectures—specifically, insufficient scalability and fault tolerance, along with high deployment costs—we propose SICRSA, a novel resource scheduling architecture for CFN that integrates SDN and ICN.
- (2) Building upon SICRSA, we design an ICN-inspired hierarchical naming scheme for computing services, develop a computing service request packet format that extends the IP header, and specify in detail the associated service request identification process and workflow.
- (3) To tackle the challenges of co-optimizing computing nodes and transmission paths in dynamic network environments, as well as the poor generalization exhibited by many DRL-based routing models, we propose CRGLD—a GNN- and LSTM-driven computing-aware routing optimization algorithm—within the SICRSA framework.

The remainder of this paper is organized as follows: Section 2 reviews the related work in this field. Section 3 details the proposed SICRSA architecture. Section 4 elaborates on the CRGLD computational-aware routing algorithm. Section 5 provides a comprehensive evaluation of the algorithm’s performance through simulation experiments. Finally, Section 6 concludes the paper and outlines potential directions for future research.

2. Related Work

This section reviews relevant research pertaining to the design of computing resource scheduling architectures for CFN and the optimization of CAR. Section 2.1 categorizes existing methods for constructing computing resource scheduling architectures based on SDN and ICN frameworks. Section 2.2 surveys related works on computational-aware rout-

ing, discusses their key characteristics, and provides a concise overview of both traditional routing algorithms and routing optimization approaches based on reinforcement learning.

2.1. Research Status of Computing First Network Architecture Design

Owing to the inherent characteristics of Computing First Networks—such as resource heterogeneity, geographical distribution, dynamic network conditions, and load imbalance—conventional network architectures, which implement network functions through distributed control and data planes, present notable limitations. In such architectures, network devices (e.g., switches and routers) independently handle data forwarding and processing, while control logic is distributed across multiple nodes. This decentralized control paradigm increases the complexity of network management and configuration, consequently limiting the flexibility and programmability required for effective resource orchestration.

In recent years, emerging network paradigms have been increasingly applied to Computing First Networks, with SDN serving as a prominent example. Unlike traditional architectures, SDN introduces a centralized controller that decouples network control and management logic from the underlying physical devices. Within the SDN framework, the controller maintains centralized, programmable oversight of the entire network, using standardized protocols to communicate with switches and direct their data-forwarding and processing behavior. This design provides a unified global view of both computing and network resources, enabling direct programmatic control over the data plane.

For instance, Gong et al. [3] incorporated SDN into the CFN framework by introducing a centralized controller and proposed a software-defined, computing-aware architecture. This architecture facilitates the perception and modeling of computing and network resources, offering a novel approach to computational resource awareness in CFN.

However, the centralized controller, which acts as the so-called “central brain” of the network, introduces critical vulnerabilities. In the event of controller failure or cyberattacks, the control plane becomes susceptible to a single point of failure, potentially resulting in widespread network outages, service disruption, and inherent limitations in scalability and reliability. In contrast, computing resource scheduling architectures based on ICN offer a promising alternative to address these challenges. ICN employs content names—rather than IP addresses—as the primary identifier for routing and data delivery. It natively supports in-network caching, multipath distribution, and dynamic resource discovery, thereby enhancing routing flexibility and resilience in CFN environments.

Recent years have witnessed growing research interest in integrating SDN and ICN within CFN. Several researchers have proposed architectural solutions that combine ICN with CFN. This approach leverages content names as unique resource identifiers, enabling flexible network management while supporting efficient transmission and processing of massive data volumes. Two primary implementation approaches for SDN- and ICN-based CFN architectures have been explored, as follows [4]: (i) employing dedicated ICN-compatible hardware to deploy SDN-based networks, and (ii) using SDN technology to realize ICN functionality over existing network infrastructure without modification [5,6]. Representative efforts include the work of Imran et al. [7], who introduced a joint communication and computing resource allocation framework for sixth-generation information-centric networks, featuring a microservice-centric interest naming structure to facilitate name-based communication. For computing resource allocation, their scheme incorporates a priority-based mechanism to ensure effective distribution in multi-access edge computing environments. Ni et al. [8] proposed a path-based computing resource scheduling architecture within ICN, integrating TOPSIS (Technique for Order Preference by Similarity to Ideal Solution) and an entropy weight-based scheduling strategy to compre-

hensively evaluate computing nodes, combined with a roulette algorithm for probabilistic node selection. Similarly, Krol et al. [9] developed a novel architectural mechanism for CFN convergence using ICN, adopting computational graph representations for distributed programs to achieve computing awareness and resource scheduling, with ICN's remote method call serving as the execution environment.

However, while these architectural designs offer certain advantages in terms of simplicity, performance, and fault tolerance, they still face notable practical limitations. Existing implementations predominantly depend on specialized hardware that natively supports ICN functionality, resulting in relatively high deployment costs and constrained scalability in real-world scenarios.

All of the aforementioned network architecture schemes successfully achieve separation between the control and data planes and implement essential functionalities such as in-network caching and packet forwarding. However, several critical challenges remain unresolved:

- (1) While [3]'s SDN-based architecture provides a unified view, it inherits the inherent single-point-of-failure and scalability limitations of a central controller. Our proposed SICRSA architecture integrates ICN principles with SDN, creating a hybrid control plane. This design mitigates the central controller's brittleness by distributing resource awareness and name resolution (leveraging ICN's strengths) while retaining SDN's programmable control for global optimization. This directly addresses the reliability and scalability challenge noted as a limitation in centralized approaches.
- (2) Several cited works [4–6] explore ICN-SDN integration, and others [8,9] propose ICN-based compute scheduling, but their practical deployment often relies on specialized ICN-native hardware. SICRSA innovates by proposing a software-defined, IP-compatible ICN overlay. Our IP-header-extension-based packet format and naming scheme enable computation-aware, ICN-inspired semantics to run on standard Open-Flow switches, significantly reducing deployment barriers and enhancing practicality compared to hardware-dependent solutions.

2.2. Research Status of Computational-Aware Routing Optimization

Unlike traditional IP routing, in which routers forward packets by consulting internally maintained routing tables to determine the next-hop address, CAR in CFN introduces fundamentally distinct requirements. In conventional IP networks, routers periodically exchange routing information—such as topology, address prefixes, and link state—with neighboring devices to maintain forwarding efficiency. In contrast, the core determinant for routing in CFN shifts from connectivity alone to computational capability. The principal objective of CAR is to accurately map diverse computational service requests to their optimal computing nodes, a task that poses significant challenges to existing network architectures. Integrating computational resource and service information directly into traditional routing tables would impose excessive overhead for updates and maintenance, consuming considerable network and computational resources. Instead, CFN must dynamically synthesize the real-time status of computing nodes with current network forwarding conditions to enable efficient resource utilization and orchestration. This routing and scheduling process must jointly consider multi-dimensional factors, including the state of computational resource nodes (e.g., computing type, capacity, and load) and the quality of network services (e.g., latency, bandwidth, and packet loss). Consequently, a key technical challenge lies in effectively integrating computational and network resource information to co-optimize the selection of computing nodes and transmission paths—a capability essential for realizing the full potential of Computing First Networks.

A variety of approaches for CAR have been progressively developed. Traditional routing algorithms are significantly limited in meeting the multi-dimensional and dynamic resource scheduling demands of Computing First Networks, particularly in efficiently coordinating computational and network resources. In response, the research community has proposed a range of innovative CAR schemes, which can be broadly classified into three categories: cost-based algorithms, optimization theory-based algorithms, and heuristic algorithms. Substantial progress has been made in recent years to advance these routing methodologies.

First, cost-based algorithms reduce complex multi-objective routing problems into a unified optimization framework by constructing a composite cost metric. For instance, CFN-dyncast [10] employs a distributed mechanism that schedules user requests to the optimal computing node based on real-time node load and network conditions. Compared to centralized approaches, CFN-dyncast reduces service response time, though its performance in environments where CFN routers are connected under homogeneous network conditions requires further validation.

Second, optimization-theory-based algorithms formulate the routing problem as a mathematical programming model to systematically search for global or local optimal solutions. Han et al. [11] introduced a utility-optimization-based method for joint computational resource and bandwidth allocation, selecting optimal service nodes and routing paths to improve overall computational resource utilization. Yao et al. [12] designed a novel routing protocol that holistically integrates both computational and network resource states, thereby scheduling service requests to optimal nodes while improving system capacity and end-to-end latency. Ding et al. [13] proposed a joint routing and resource allocation model in Computing-Aware Networks, aiming to minimize total operational cost and computation time, although the model exhibits certain limitations in reliability.

Finally, heuristic and system-level approaches offer practical implementations and experimental validation for these complex challenges. Zhao et al. [14] developed a CAN-based simulation prototype, verifying the effectiveness of its CAR mechanism in improving overall computational resource utilization. Lin et al. [15] proposed a computational-aware routing strategy based on heuristic algorithms. This strategy determines the optimal routing path by comprehensively considering the dynamic network performance and the status of computing resources. Liu et al. [16] built a Kubernetes-based prototype platform for Computing First Networks, enabling dynamic resource allocation and flexible request scheduling, which effectively reduces response latency and enhances load balancing.

While these methods offer foundational insights and co-optimization frameworks, they often struggle with the real-time adaptability and computational complexity inherent in large-scale, dynamic CFN environments. Their performance frequently relies on accurate global models and may not generalize efficiently under rapidly changing network states or unseen topological conditions.

In addition, intelligent routing algorithms have emerged as promising approaches for optimizing computing-aware routing strategies, primarily categorized into RL-based and DRL-based methods. As an effective decision-making and control technique, RL has demonstrated considerable success in domains such as network routing and resource allocation. For example, Dai et al. [17] proposed a transfer learning-enhanced routing mechanism for named-data Computing First Networks, incorporating the accuracy of pre-trained classification models as prior knowledge to significantly reduce training time, though their study was confined to classification tasks in specific application scenarios. Xie et al. [18] introduced a non-dominated sorting genetic algorithm integrated with an RL-based parameter adjustment strategy, which holistically optimizes multiple objectives to enhance routing performance and efficiency while notably reducing client-side latency

and cost. Gong et al. [19] designed a software-defined CAR algorithm based on Q-Learning, improving computational resource utilization, optimizing network communication, and effectively coordinating computational and network resources. Qi et al. [20] focused on a single-source, single-destination system, introducing timeliness- and value-aware metrics and developing an RL-based update strategy to maximize the overall task success rate while minimizing unnecessary communication overhead. Xie et al. [18] further proposed a multi-objective optimization routing planning method that combines a non-dominated sorting genetic algorithm with RL-driven parameter tuning, effectively addressing routing challenges in CFN and substantially reducing latency and cost. Although these methods offer innovative solutions to performance bottlenecks and resource inefficiencies in large-scale CFN, challenges remain—particularly concerning model generalization and scalability in extensive, dynamic network environments.

Building upon recent advances in DRL for network control, a growing body of research has begun to adopt emerging DRL techniques to achieve model-free optimization in networking. Consequently, a variety of DRL-based CAR algorithms have rapidly developed. For instance, Li et al. [21] proposed a DRL-based method for joint optimization of computational and network resources in machine-to-machine networks, though it is primarily tailored for Internet of Things (IoT) environments. Yu et al. [22] integrated federated learning with DRL to enable stable, reliable, and real-time coordination between edge nodes and associated multi-access edge computing nodes in 5G ultra-dense networks. Bai et al. [23] addressed the underutilization of network resources in existing routing schemes by introducing a DQN-based routing optimization algorithm, which improves average network throughput while reducing latency and packet loss. Feng et al. [24] constructed a collaborative scheduling framework that supports the transmission of calculation results, and further proposes an algorithm based on Deep Reinforcement Learning to optimize the long-term scheduling problem of balanced computing-aware routing, thereby improving the success rate of computing tasks and ensuring the balanced utilization of computing and network resources. Asmat et al. [25] developed a Deep Reinforcement Learning-based caching framework that optimizes edge caching decisions by incorporating real-time network conditions, content popularity, and resource utilization, thereby enhancing cache hit rates, reducing latency, improving bandwidth efficiency, and ensuring secure caching under dynamic network states.

However, most existing computing power routing optimization algorithms based on reinforcement learning/Deep Reinforcement Learning are trained and evaluated on specific, static network topologies. Its decision-making mechanism is usually closely coupled with the topological features encountered during the training process. Therefore, when deployed to unfamiliar network topologies or dynamically changing network topologies (such as due to link failures or node additions), these algorithms exhibit limited generalization ability. This often requires time-consuming and laborious retraining, severely limiting its generalization in dynamic network environments.

Current research on computational-aware routing continues to confront the following key challenges:

- (1) The decision-making process of CAR requires the simultaneous consideration of multidimensional factors, including computational resource status, network performance, and task characteristics. However, conventional routing algorithms are unable to achieve true joint optimization of computational and network resources, making it difficult to attain a global optimum in the coupled selection of computing nodes and transmission paths.
- (2) Although Deep Reinforcement Learning shows promise in routing optimization, existing DRL-based routing algorithms suffer from limited generalization capability.

Specifically, when encountering previously unseen topologies or dynamic topological changes, these models often require extensive retraining to maintain performance, limiting their applicability in real-world, evolving network environments.

3. Architecture

This section presents the proposed computing resource scheduling architecture for CFN, termed SICRSA. First, Section 3.1 outlines the overall architectural design, including the topology and the functional relationships among its core entities. Subsequently, Sections 3.2–3.4 detail the computing service naming scheme, the corresponding packet format design, and the service request identification process, respectively. Finally, Section 3.5 elaborates on the end-to-end workflow for computing service request scheduling within the SICRSA framework.

3.1. Overview

This section details SICRSA, a novel computing resource scheduling architecture designed for CFN. As shown in Figure 1, the SICRSA framework is hierarchically organized into three distinct planes from bottom to top: the service plane, the Computing First Network plane, and the control plane. The following subsections elaborate on the functional components and their respective roles within each plane.

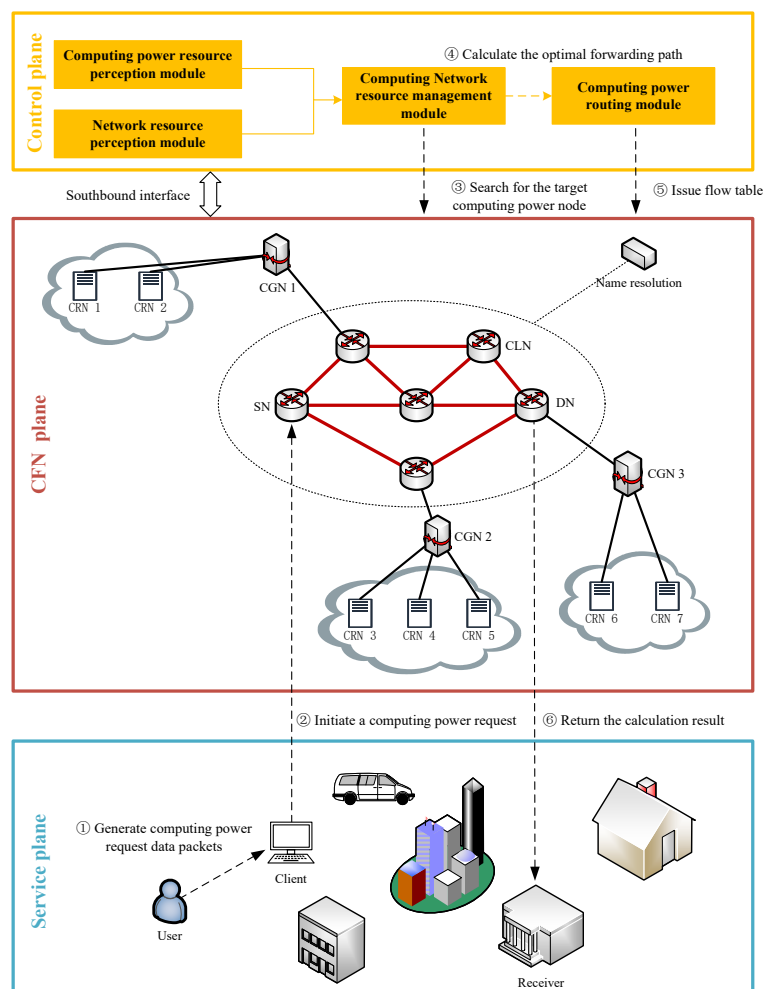


Figure 1. SICRSA overview.

3.1.1. Service Plane

To accommodate diverse types of computational tasks and associated resource requirements, the hierarchical naming mechanism of ICN is adopted to uniquely identify computational services. This allows computational service request packets to operate directly at the service-logic level, bypassing the limitations of traditional IP address-based forwarding. After a user-initiated service request is named and resolved, trigger packets are forwarded along pre-established computational-aware routing paths determined by the control plane. The uploading of computational tasks and the return of results are implemented via the OpenFlow protocol to ensure low latency and high reliability. Tailored computing resource scheduling strategies are designed according to differentiated user needs, thereby effectively fulfilling varied service requests. This architecture can be efficiently deployed in application scenarios such as the Internet of Vehicles, AI model training, and distributed scientific computing.

3.1.2. Computing First Network Plane

The Computing First Network plane follows a hierarchical structure, which consists of the following layers from bottom to top: computational resource nodes (CRNs), computing gateway nodes (CGNs), resource localization nodes (CLNs), computational-aware routing nodes (CANs), and name resolution nodes (NRNs).

Within the CFN architecture, computational resource nodes constitute the fundamental physical or virtualized units that deliver computing capabilities. Distributed across diverse clusters or geographical regions, these nodes encompass resources such as CPUs, GPUs, memory, and container clusters, and act as the actual providers of computational power for user-driven service requests. In the SICRSA framework, computational resource nodes are classified into three categories based on their task profiles: lightweight computing nodes, intelligent computing nodes, and supercomputing nodes. Lightweight computing nodes are primarily deployed at the network edge to handle latency-sensitive and data-proximate tasks; intelligent computing nodes support computationally intensive workloads such as AI model training and graphics rendering; and supercomputing nodes are dedicated to large-scale parallel scientific computations. These heterogeneous nodes are interconnected through a unified networking protocol and scheduling mechanism, collectively forming an elastic, distributed computational resource pool that enables dynamic on-demand allocation. Furthermore, computing resources can be containerized and encapsulated as standardized APIs, allowing flexible invocation by end users or applications.

The computing gateway node functions as a critical entity that provides access and management capabilities for both users and computational resource nodes. On the resource side, it monitors the real-time status of each computational resource node, identifies their specific types, and generates corresponding computational service identifiers. The connection between a computing gateway node and computational resource nodes follows a one-to-many topology. Through computational awareness, the gateway collects resource information from computational nodes and uploads status updates to the controller while maintaining local copies, thereby reducing network overhead caused by frequent state announcements. This design simultaneously fulfills the requirement for real-time maintenance of computational resource status. By establishing a distributed computational resource status table, the system enables real-time updates of node states, coordinates local edge nodes, and supports globally optimized routing strategies. This distributed approach effectively mitigates the single points of failure inherent in centralized resource management architectures.

The architecture of a CAR node is analogous to that of a routing node in an ICN network and is implemented using an OpenFlow switch. After computing service request

packets are distributed according to the principle of proximity, the CAR module in the control plane translates routing policies into OpenFlow flow tables, which are then delivered to the CAR nodes via the southbound interface. These nodes ultimately perform packet matching and forwarding based on the installed flow-table rules.

The name resolution node maintains mapping relationships between content identifiers and IP addresses for all entities within its resolution domain, providing name resolution services. In contrast to the end-to-end transmission paradigm of traditional IP networks, Information-Centric Networking employs content names as unique identifiers for resource localization, enabling content-based lookup to identify publishers of network resources. Current content naming mechanisms primarily include self-certifying and hierarchical schemes; this study adopts the latter. When a user initiates a computational task request, the network selects an appropriate idle computing node based on the task type to provide the service. Within the SICRSA framework, each computational service request is assigned a unique content identifier. The client receives user requests, generates corresponding data packets, interacts with the ICN name resolution system to resolve content identifier-to-IP address mappings, and randomly distributes these requests to the nearest computing-aware routing nodes.

3.1.3. Control Plane

The control plane integrates four core modules: the computing resource awareness module, the network resource awareness module, the computing and network resource management module, and the CAR module. Among them, the computing and network resource awareness module collects the real-time status of global computing and network resources. The computing and network resource management module monitors and processes global computing and network information, maintaining an updated unified view of the global resource environment. The CAR module dynamically selects optimal computing nodes and transmission paths based on the integrated resource information, thereby achieving global resource-demand matching. The detailed design and implementation of each module are described in the following subsections.

The computing and network resource management module acquires resource information from both the computing resource awareness and network resource awareness modules. This information comprises two categories of metrics, as follows: computing resource metrics (e.g., type and quantity of computational units) and network resource metrics (e.g., link available bandwidth, delay, and packet loss rate). These metrics are stored within the management module and, after intelligent aggregation and processing, serve as inputs to the routing optimization algorithm. In this work, the CAR module leverages these processed metrics—along with cached routing strategies and the proposed intelligent algorithm (CRGLD)—to generate optimal routing plans. The controller subsequently conveys these paths to the Computing First Network plane by installing corresponding flow-table entries into the relevant switches.

Unlike traditional routing mechanisms that focus exclusively on network conditions, this study proposes a routing mechanism that integrates computational resource information with network-layer metrics. During routing and scheduling, it simultaneously considers both computational resource awareness and link-state information to enable efficient computational task management. The computational resource awareness module employs a hybrid announcement scheme combining directed and flood-based dissemination. By extending the Border Gateway Protocol (BGP), computational resource information is embedded into protocol messages. Through active reporting, computational status is propagated across the network, achieving real-time awareness of computational resources. Computational resource nodes notify the computational resource awareness module and

the computational-network resource management module of the status information layer by layer via the southbound interface. This information is then associated with the network resource awareness module for further dissemination.

As illustrated in Figure 2, each computational resource node first models and measures its resource status, defines a new IP message format to carry this information, and reports it upward. The status information is transmitted through the northbound interface on the computational side to the computational-network resource management module (C1→C2). Following intelligent processing, it is subsequently used for routing decisions.

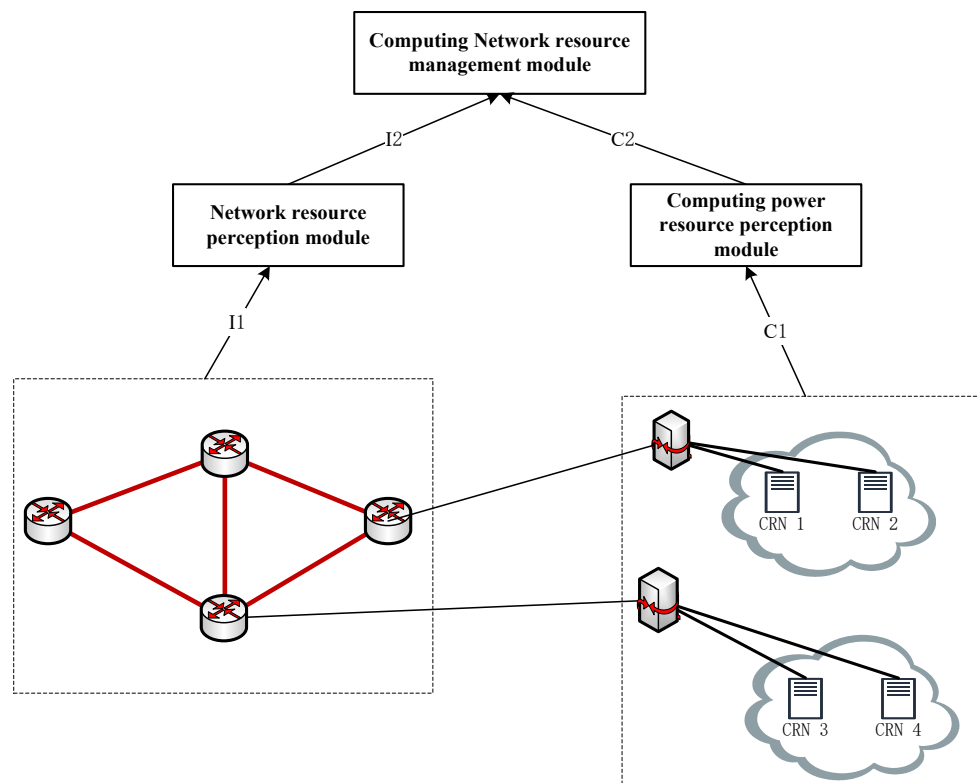


Figure 2. Computing network perception process.

The network resource awareness module detects and collects information from each connected device in the network, including port connectivity, bandwidth statistics, and inter-device link latency. This information is subsequently flooded to all computational-aware routing nodes. The computing-network resource management module is responsible for aggregating and storing metrics such as link bandwidth, latency, and packet loss rate. The controller actively issues data-request packets and periodically monitors the status of network link resources, with the network topology maintained in a four-tuple data structure. As shown in Figure 2, network status information is disseminated to all computational-aware routing nodes via flooding and is forwarded to the computing-network resource management module (I1→I2) through the northbound interface on the network side. Based on the acquired computational and network resource information, the computational-aware routing module performs a comprehensive evaluation of both computational and network metrics to conduct joint routing optimization. This process determines next-hop addresses and updates routing tables. When new computational service requests arrive, they can be forwarded efficiently via routing-table lookup, enabling rapid packet-forwarding decisions.

The network resource awareness module is primarily responsible for sensing both network topology and network quality of service. The latter encompasses performance metrics such as bandwidth, delay, and packet loss rate, while topology awareness includes the identification of link connectivity, switch locations, and host positions within the network.

The estimation of available link bandwidth is performed through real-time analysis of switch port byte-count statistics. The controller periodically, at a default interval of 2 s, queries bandwidth metrics from all ports, including current byte counts and port capacities. For each link identified by the network topology discovery module, the controller determines the corresponding switches and terminals at both ends and selects the minimum of the two directional available bandwidth values as the link's effective residual capacity. This approach ensures that routing decisions remain within the actual carrying capability of each link, thereby preventing congestion from over-subscription.

The residual bandwidth of a link is computed as follows:

$$bw_{link} = \min(B_{src}^{total} - B_{src}^{used}, B_{dst}^{total} - B_{dst}^{used}) \quad (1)$$

Here, B_{src}^{total} represents the total bandwidth of the source port, B_{src}^{used} represents the used bandwidth of the source port, $B_{dst}^{total} = \frac{(b_c - b_l)}{t}$ represents the total bandwidth of the destination port, B_{dst}^{used} represents the used bandwidth of the destination port, b_c represents the current byte count, b_l represents the previous period byte, and represents the counting time interval.

For the given path p , the network computing management module performs information processing, using $i \in p$ of each link, calculate the available bandwidth bw_{linki} for each path:

$$bw_p = \min_{i \in p}(bw_{linki}) \quad (2)$$

The measurement of network delay consists of two main parts, namely the Echo delay between the controller and the switch, and the link LLDP delay between the switches. The controller sends Echo requests to all switches and receives their responses, calculates the round-trip transmission time from the controller to the switches, and then divides it by 2 to obtain the one-way delay, that is, the Echo delay. The one-way delay between switches is measured by sending LLDP data packets. When the target switch receives the LLDP packet, it will send it to the controller through the Packet-In event. The controller obtains the LLDP delay by subtracting the timestamp in the packet from the current time. The calculation of the actual link delay requires eliminating the delay influence between the controller and the switch. Suppose there is a link from the source switch A to the destination switch B. The LLDP delay from A to B is measured, denoted as $lldp_{ab}$, and the LLDP delay from B to A is denoted as $lldp_{ba}$. At the same time, the Echo delay from the controller to A is obtained, denoted as $echo_a$, and the Echo delay from the controller to B is denoted as $echo_b$.

The formula for calculating the actual link delay is as follows:

$$delay_{link} = \frac{(lldp_{ab} + lldp_{ba} - echo_a - echo_b)}{2} \quad (3)$$

For the given path p , the computing network management module performs information processing, using $i \in p$ of each link, calculate the delay $delay_{linki}$ of each path:

$$delay_p = \sum_{i \in p} delay_{linki} \quad (4)$$

The measurement of network packet loss rate is achieved by analyzing the packet counters on the switch ports. The controller periodically acquires the port statistics of all switches, including the number of sent packets and received packets. It traverses each link discovered by the network topology awareness module to obtain the information of the switches and ports at both ends of the link. The forward packet loss rate of the link is $\frac{tp_s - rp_d}{tp_s}$, and the reverse packet loss rate of the link is $\frac{tp_d - rp_s}{tp_d}$. Ensure that the routing decision is

based on the worst-case scenario, and take the maximum value of the bidirectional packet loss rate as the final packet loss rate of the link.

The packet loss rate of a link is given by the following:

$$loss_{link} = \max\left(\frac{tp_s - rp_d}{tp_s}, \frac{tp_d - rp_s}{tp_d}\right) \quad (5)$$

Among them, tp_s is the number of packets sent at the source port, rp_d is the number of packets received at the destination port, tp_d is the number of packets sent at the destination port, and rp_s is the number of packets received at the source port.

For the given path p , the computing network management module performs information processing, using $i \in p$ of each link, calculate the packet loss rate $loss_{linki}$ of each path:

$$loss_p = 1 - \prod_{i \in p} (1 - loss_{linki}) \quad (6)$$

The computational-aware routing module formulates scheduling strategies for computational tasks based on the processed computing-network resource information provided by the computing-network resource management module. These strategies are then translated into OpenFlow messages and delivered to the Computing First Network layer to update the relevant routing tables, thereby guiding the flow-table distribution process. The specific computational-aware routing optimization algorithm employed will be detailed in Section 5.

3.2. Naming Mechanism for Computational Service Requests

The ICN network uses a hierarchical naming mechanism to name data, similar to the URL on the Internet. The difference lies in that the ICN name directly identifies the data itself rather than its location. The basic structure of an ICN name consists of two parts—the prefix and the suffix. The prefix usually includes one or more identifiers to indicate the source or relevance of the data. Suffixes include the actual name or identifier of the data and are used to uniquely identify specific data. Academic resources, for example, data/edu/ucl/cs/lectures/network-icn-2023. The mp4, /edu/ucl/cs/lectures on behalf of the prefix, network-icn-2023. Mp4 on behalf of the suffix. In the actual network environment, ICN clients or network nodes usually map the various parts of the content name to fixed-length hash values for forwarding.

In this work, the aforementioned hierarchical naming mechanism is applied to describe service requests in Computing First Network. A typical service request includes both computational requirements and network requirements. Specifically, computational requirements comprise the type of computing resource, the minimum expected computational capacity, memory, and storage, among other parameters. Moreover, this naming scheme offers strong extensibility, allowing additional QoS constraints to be embedded according to diverse user scenarios. These constraints include task deadlines, reliability specifications, cost budgets, and location preferences such as proximity to users or data sources.

3.3. Design of Computational Service Request Data Packets

In a TCP/IP-compatible ICN environment, to allow OpenFlow switches to reliably distinguish between conventional IP packets and ICN messages carrying computational service requests, this paper proposes a semantic extension of the protocol field in the IP header. When the controller parses an incoming packet, it can identify whether the packet contains a computational service request by examining the value of this field.

In the IP protocol, the protocol field identifies the type of upper-layer protocol, with its value range and usage being uniformly assigned and managed by IANA. Specifically, val-

ues 0–143 are assigned protocol identifiers, 144–252 are currently unallocated, 253–254 are reserved for experimentation and testing, and 255 is a reserved value. Based on this allocation scheme, this work selects two values—152 and 153—from the unallocated range to respectively identify ICN interest packets and data packets that carry computational service requests. When a user initiates a computational service request, the protocol field is set to 152. When an ICN publisher responds to the request and returns the computation result, the protocol field is set to 153. Through this mechanism, network devices can unambiguously distinguish between computational service requests and their corresponding responses, thereby providing foundational protocol support for computational-aware routing and resource scheduling in ICN-enabled environments.

Building on the aforementioned mechanism, SICRSA further designs a dedicated packet format for computational service requests based on the IPv6 extension header, illustrated in Figure 3. The design introduces two key fields into the packet header—the source ID and the destination ID. The source ID identifies the end-user initiating the computational request, while the destination ID specifies the type of computational resource service being requested, for example, CRN1 or CRN2. Following these identifier fields are the Resource Request Quantity field and the QoS field. The resource request quantity field details the specific resource requirements, including the types and amounts of computational resources such as CPU cores, GPU units, and memory capacity. The QoS field defines the network performance constraints that must be satisfied, including bandwidth, latency, packet-loss rate, and maximum completion time. This structured design enables each computational service request to be precisely identified, classified, and scheduled within the IP network, thereby establishing a solid foundation for subsequent service-semantics-based routing and resource matching.

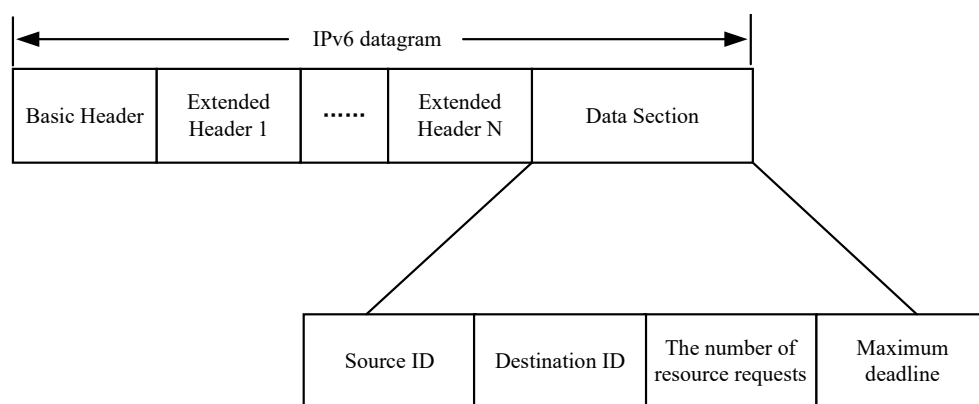


Figure 3. The design of computing power service request data packets.

3.4. Computational Service Request Identification Process

When an OpenFlow switch receives a computational resource request message, it executes the computational service request identification procedure illustrated in Figure 4. This process begins by matching the protocol field to determine whether the message represents a computational service request, and then applies the corresponding forwarding logic. Specifically, when a user device sends a computational request to the nearest routing node, if the switch fails to match the destination address in its local flow table, it triggers a Packet-In message and reports it to the Ryu controller. Then, the Ryu controller parses the protocol number field in the IP header to check if it matches the predefined 152 field. If the protocol number matches the preset value, the controller invokes the computational-aware routing module to further parse the computational resource requirements and service constraints carried in the IP extension header. Based on this information, it identifies a suitable target computational node and computes an optimal forwarding path that satisfies

the specified quality-of-service requirements. If the protocol number does not match, the controller treats the message as a non-computational service request and discards it. In the SICRSA architecture, every computational-aware routing node is capable of receiving computational resource request messages. By reading the embedded resource-demand fields in the message, these nodes can make intelligent forwarding decisions according to the system's routing policy, thereby enabling efficient resource scheduling and service delivery.

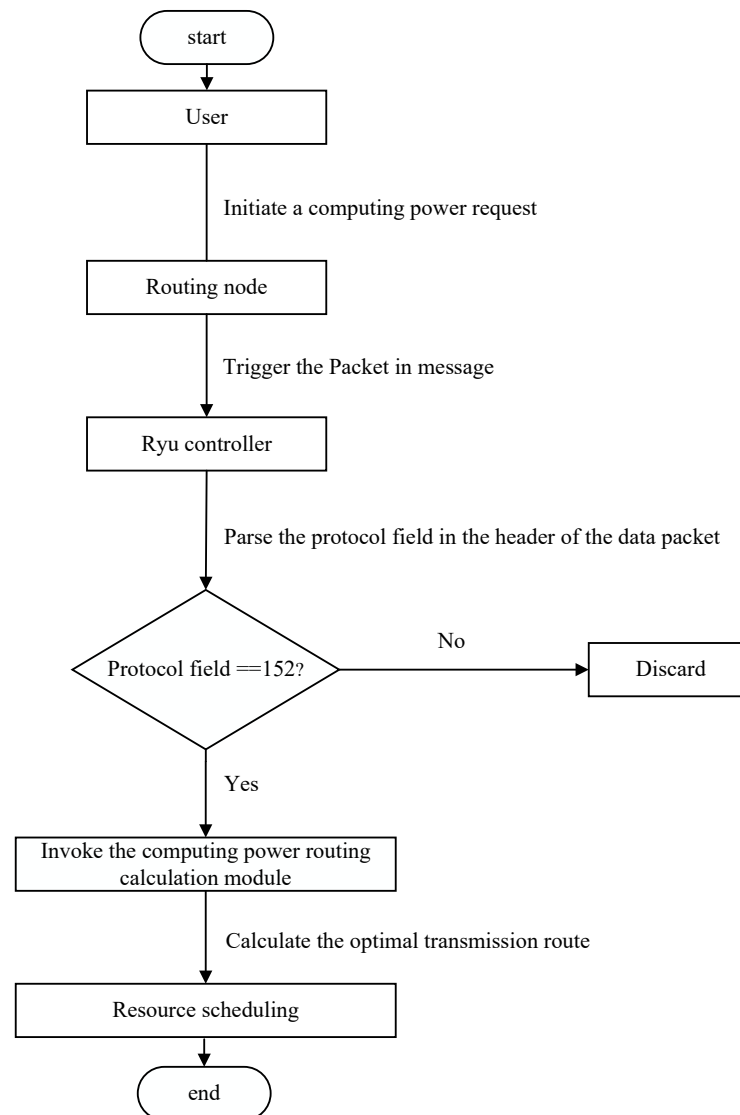


Figure 4. The process of identifying requests for computing power services.

3.5. System Workflow

This section describes the end-to-end workflow for computational resource scheduling. As indicated by the arrows in Figure 5, the diagram illustrates the complete lifecycle of a computational task, from its initiation to final completion.

1. The client generates a computational service request packet according to the resource requirements specified by the user.
2. The client initiates the computational task request. Following the principle of proximity-based scheduling, the service request is forwarded to a nearby computing-aware routing node. By parsing the packet header, the node extracts parameters such as the computational demand and the quality-of-service (QoS) constraints of the task.

3. Identify target computational nodes that satisfy the requirements. The computing-network resource management module maintains the real-time status of computational and network resources, including node availability, bandwidth, latency, packet-loss rate, and link connectivity. Based on this information, the computing-aware routing module performs packet matching and processing to locate suitable target computational nodes.
4. Compute the optimal forwarding path. After intelligently processing the collected resource status, the computing-aware routing module employs an optimization algorithm to generate a routing strategy for the task. The controller pre-computes the corresponding resource scheduling policy and generates the routing flow table.
5. Install and execute the flow table. The computing-aware routing module translates the routing strategy into OpenFlow messages and, via the southbound interface protocol, delivers them to the relevant switches in the Computing First Network plane. The switches then process and forward the computational service request packets according to the installed flow-table rules.
6. Return the computation result. Once the computation is complete, the result is transmitted back along the designated path to the destination computing-aware routing node and finally delivered to the receiving end-user.

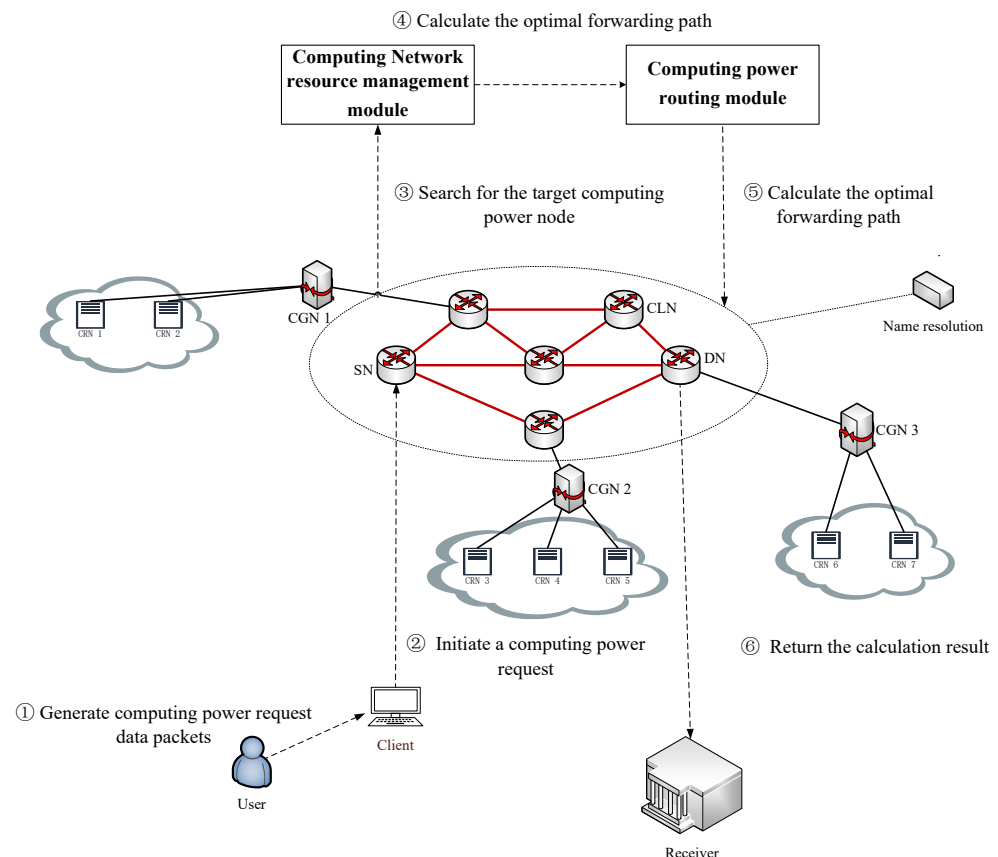


Figure 5. The resource scheduling process of computing power requests.

4. Computational-Aware Routing Optimization Algorithm Based on Deep Reinforcement Learning

In Computing First Networks, user-initiated computational service requests follow random access patterns, and the network state itself is subject to continuous dynamic change. In such an environment, the routing strategy adopted at any given moment strongly influences subsequent network states, a characteristic that exhibits the Markov property. Thus, the long-term computational resource scheduling process can be formally modeled

as a Markov Decision Process (MDP) to estimate how current decisions affect future system states. Within the SICRSA architecture, this MDP framework captures the interaction between an intelligent agent—the Computing-Aware Routing via Graph and Long-term Dependency Learning (CRGLD) algorithm implemented in the computational-aware routing module of the control plane—and the dynamic Computing First Network environment.

In emerging CFN scenarios, the network must simultaneously optimize resource allocation and schedule service requests along transmission paths that meet specified QoS constraints. While existing reinforcement learning methods, such as Q-Learning and Deep Q-Networks (DQNs), can effectively solve Markov Decision Processes, they are inherently limited to learning from Euclidean data representations. In contrast, network topologies constitute non-Euclidean graph-structured data. Graph Neural Networks (GNNs) excel at processing such relational data, as network topologies can be naturally represented as graphs with node- and link-level features, thereby overcoming this fundamental limitation of conventional approaches.

GNNs exhibit distinct advantages in processing graph-structured data. By representing network topology as a graph with node and edge attributes, GNNs demonstrate strong generalization capability in the face of topological changes, effectively addressing the challenges posed by dynamic network structural variations. Meanwhile, Long Short-Term Memory (LSTM) networks excel at learning temporal dynamics from link performance metrics. The integration of GNN and LSTM enables the simultaneous learning of structural relationships among nodes and links in the topology, along with time-varying dynamic features, thereby supporting a more comprehensive extraction of spatiotemporal fusion characteristics from the network. To this end, this section proposes a GNN- and LSTM-based computing-aware routing optimization algorithm within the computational-aware routing module of the SICRSA control plane.

This section begins by outlining the overall design of the algorithm. It then details the extraction of spatiotemporal fusion features, summarizes the key components and workflow of the proposed method, and concludes with experimental validation of the algorithm's effectiveness and generalization capability across diverse network topologies.

4.1. The Design of CR-GLD

Building upon the global computing-network resource view, the proposed algorithm employs Graph Neural Networks (GNNs) and Long Short-Term Memory (LSTM) networks to extract spatiotemporal features of the network topology. These fused features serve as the state input for a Deep Reinforcement Learning agent. The agent then performs a collaborative two-stage decision-making process—first, it allocates suitable computing nodes to user service requests to meet computational demands; second, it determines optimal transmission paths between each source–destination pair, that is, between the user device and the assigned computing node. This approach aims to jointly optimize computing and network resources, thereby enhancing the dynamic allocation and load balancing of overall network resources.

Furthermore, the algorithm utilizes the hierarchical naming mechanism inherent in Information-Centric Networking to identify computational service request packets. Requests of the same service type share a unified computational resource identifier, which enables the computing-aware routing module to implement efficient resource allocation and routing strategies. The workflow of the CR-GLD algorithm is illustrated in Figure 6, which outlines the following key steps:

- (1) **Computing-Network Information Update:** The computational resource awareness module and the network resource awareness module continuously monitor the status of computational and network resources to determine whether an information update

is warranted. The system refreshes computing-network data at regular intervals and also triggers real-time updates upon major network events, ensuring that all service requests are processed based on the latest resource state.

- (2) **Candidate Computational Node and Path Generation:** When each request arrives, the system generates a candidate set, including m available computational nodes and k feasible transmission paths. Based on the parameters listed in Section 5.3, adjustments were made by conducting grid searches on the reserved validation set (subset of Geant topology traffic). The final values are those that maximize the stability and convergence speed of accumulated rewards during the training process. Based on preliminary simulations across our training topologies, we found that $m = 5$ provided a good balance. We observed that in the topology we evaluated, the main QoS metrics of the optimal path (latency, bandwidth) were almost always within the first three shortest paths (by hop count). Increasing k above 3 will result in a diminishing return on solution quality, while linearly increasing the state action space of the DRL agent slows down convergence speed. Therefore, we set $k = 3$. This ensures multiple potential candidate solutions are considered under varying network conditions, thereby optimizing routing efficiency.
- (3) **Spatiotemporal Feature Extraction:** The algorithm utilizes GNN to capture the spatial dependencies among network nodes and LSTM networks to model the temporal dynamics of link parameters—such as bandwidth, latency, and packet loss rate. The extracted spatial and temporal features are then concatenated and fused into a unified representation, which serves as the input state for the deep reinforcement learning agent.
- (4) **Optimal Routing Strategy Computation:** By leveraging the integrated computing and network resource information, the agent performs a dual decision-making process—first, it selects the optimal computational resource nodes based on the specific requirements of each task request; second, it determines the most efficient transmission path for each computational task. This dual process enables the intelligent joint optimization of computational and network resources.

4.2. The Process of Feature Extraction Through Spatiotemporal Fusion

Unlike traditional Euclidean data, network topology is inherently graph-structured. To process such non-Euclidean data effectively, the proposed algorithm incorporates a GNN-LSTM module designed to jointly capture both the spatial layout and temporal dynamics of the network. In its spatial component, the model adopts a Message-Passing Neural Network (MPNN), which iteratively updates node representations by propagating information between each node and its neighbors, thereby effectively extracting topological structural features.

However, network states—such as link bandwidth, latency, and packet loss rate—typically exhibit strong temporal variations that pure GNN architectures struggle to model fully. To address this, the algorithm integrates LSTM as its temporal component. Leveraging its gating mechanism, LSTM learns long-range dependencies in link performance parameters over time. By fusing the spatial features from GNN with the temporal features from LSTM, the model constructs a unified spatiotemporal representation, providing a more comprehensive basis for subsequent routing decisions. Figure 7 illustrates the computational flow of the GNN-LSTM module within the CR-GLD framework.

In network modeling, topology can be defined as $G = (V, E)$, where V represents the set of nodes and E is the set of edges connecting the nodes. GNN uses message-passing functions to propagate information between nodes and aggregate and update the states of each element. This process will be iteratively executed K times. The working process

of GNN consists of two stages, namely the message passing stage and the readout stage. The following sections respectively introduce the spatial message passing stage and the spatiotemporal feature aggregation stage.

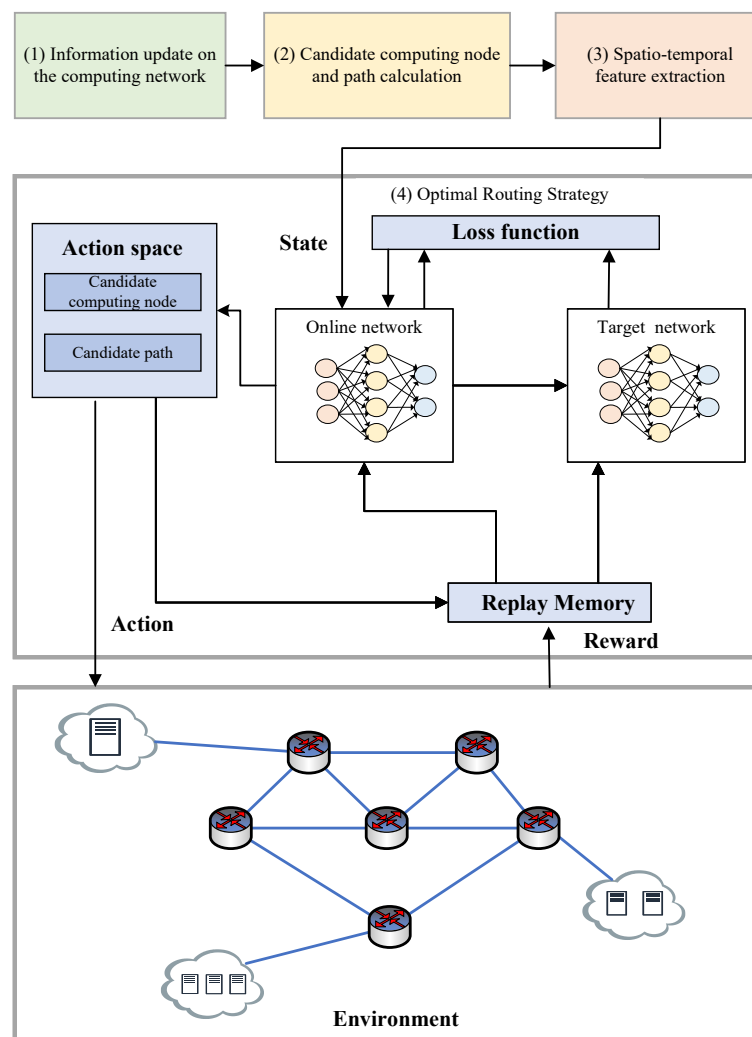


Figure 6. Overview of the CR-GLD Algorithm.

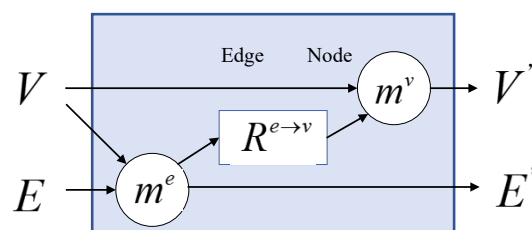


Figure 7. The calculation process of the GNN-LSTM module.

4.2.1. Spatial Message Passing Stage

On the network topology graph G_t of a single time step t , perform K GNN iterations to extract the spatial features of the current moment. Node i aggregates information from neighboring node j . The calculation formula of the message aggregation function is as follows:

$$m_i^{k+1,t} = \sum_{v_j \in N(v_i)} M^k(h_i^{k,t}, h_j^{k,t}, e_{ij}^t) \quad (7)$$

Here $m_i^{k+1,t}$ represents the total number of messages received by the node from its neighboring nodes at the time step t during the $k + 1$ iteration. $h_i^{k,t}, h_j^{k,t}$ represents the hidden state of the node i and its neighbors j at the time step t during the k iteration. e_{ij}^t represent the characteristics of the edge connecting the node i and j at the time step t ; M^k represent learnable message functions.

Node i combines its own information with aggregated information. The calculation formula for the state update function is as follows:

$$h_i^{k+1,t} = U^k(h_i^{k,t}, m_i^{k+1,t}) \quad (8)$$

Here, U^k represents the update function. It combines the current state of the node itself and the received messages to generate a new state of the node.

After K iterations, we obtain the final spatial state representation of the node i at the time step t as follows:

$$h_i^t = h_i^{K,t} \quad (9)$$

4.2.2. The Stage of Spatiotemporal Feature Aggregation

Firstly, the node states calculated by the GNN at each time step t are input as vector sequences into the LSTM to learn the dynamic link state information related to time. This stage adopts an LSTM unit with a forgetting gate. If time series modeling is conducted separately for each node i , an independent LSTM unit is maintained for each node. The calculation formula is as follows:

$$(c_i^t, h_i^{lstm,t}) = LSTM_i(h_i^t, h_i^{lstm,t-1}, c_i^{t-1}) \quad (10)$$

Among them, h_i^t is from the feature vector of node i of GNN at time step t , serving as the input of LSTM at time t . $h_i^{lstm,t-1}$ represents the hidden state of the LSTM unit of node i in the previous time step $t - 1$. c_i^{t-1} represents the cell state of the LSTM unit of node i at the previous time step $t - 1$. $h_i^{lstm,t}, c_i^t$ represent the hidden state and cell state of the current time step t obtained after the update of the LSTM unit, which capture the spatiotemporal fusion features of node i up to time t .

Secondly, to model the entire network topology graph, a readout function R is used to aggregate the features of all nodes and generate a state vector g^t representing the entire graph at time t . In the spatiotemporal feature aggregation stage, the readout function R is implemented as a global mean pooling operation. After the GNN completes its message-passing iterations for a given time step, each node in the network graph has a final updated feature vector. The readout function takes the set of all these node feature vectors and computes their element-wise average. This produces a single, fixed-length graph-level embedding vector that summarizes the overall state of the entire network at that moment. This pooled vector is then passed as input to the global LSTM module to capture temporal dynamics. The reason for making this design choice is that mean pooling is a simple, efficient, permutation-invariant operation that makes it very suitable for aggregating information from graphs. The calculation formula is as follows:

$$g^t = R(\{h_i^t \mid i \in G_t\}) \quad (11)$$

Then input it into an LSTM, and the calculation formula is as follows:

$$(c^t, h^{lstm,t}) = LSTM_{graph}(g^t, h^{lstm,t-1}, c^{t-1}) \quad (12)$$

Among them, the global graph embedding vector g^t representing the time step t is used as the input of LSTM at time t ; $h^{lstm,t}$ represents the spatiotemporal integration characteristics of the entire network's historical dynamics.

Finally, LSTM takes the output of the last time step as the input $h_i^{lstm,t}$ of the DRL agent. This feature simultaneously includes network topology spatial information and dynamic time series information.

Algorithm 1 illustrates the process of spatiotemporal fusion feature extraction. The input is a set of nodes, V^t ; an adjacency matrix sequence, $\{A^1, A^2, \dots, A^T\}$; and a node feature sequence, $\{X^1, X^2, \dots, X^T\}$. The output is the spatiotemporal fusion feature h_{lstm}^T . The initialization of the node state combines the original node feature x_i^t and zero-padding as the input of the message-passing neural network (line 4). The forward propagation process of GNN is as follows. In each round of iteration k , each node aggregates message M^k from its neighbor nodes and then updates its current state U^k in combination with its previous state (lines 8–11). The readout function R aggregates node representation h_i^t into a single graph embedding vector g^t , which represents the global network state at time step t (line 15). Take the graph embedding g^t at the current moment as input and combine it with the previous hidden state h_{lstm}^{t-1} and cell state c_{lstm}^{t-1} of LSTM to update and obtain the spatiotemporal state (h_{lstm}^t, c_{lstm}^t) at the current moment. During this process, LSTM learned the dynamic characteristics of the evolution of network link states over time (line 16). The algorithm finally outputs the LSTM hidden state h_{lstm}^T of the last time step T . This vector is a rich representation that integrates historical spatiotemporal information and will be used as the input of the Deep Reinforcement Learning agent for the DRL-based routing optimization decision in the subsequent Algorithm 2 (line 18).

Algorithm 1 The process of feature extraction through spatiotemporal fusion

Input: Number of Episodes: n ; Node set: V^t ; Adjacency matrix sequence: $\{A^1, A^2, \dots, A^T\}$; Node feature sequence: $\{X^1, X^2, \dots, X^T\}$.

Output: spatiotemporal fusion characteristics: h_{lstm}^T .

```

1:  $h_{lstm}^0, c_{lstm}^0 \leftarrow \text{Initialize LSTM State}()$ 
2: for  $t$  in 1 to  $T$  do
3:   for each node  $i$  in  $V^t$  do
4:      $h_i^{0,t} \leftarrow [x_i^t, 0, \dots, 0]$ 
5:   end for
6:   for  $k = 0$  to  $K - 1$  do
7:     for each node  $i$  in  $V^t$  do
8:        $m_i^{k+1,t} \leftarrow \sum_{j \in N(i)} M^k(h_i^{k,t}, h_j^{k,t}, e_{ij}^t)$ 
9:        $h_i^{k+1,t} = U^k(h_i^{k,t}, m_i^{k+1,t})$ 
10:    end for
11:  end for
12:  for each node  $i$  in  $V^t$  do
13:     $h_i^t \leftarrow h_i^{K,t}$ 
14:  end for
15:   $g^t \leftarrow \mathbb{R}(\{h_i^t \mid i \in V_t\})$ 
16:   $(h_{lstm}^t, c_{lstm}^t) \leftarrow \text{LSTM}_{\text{cell}}(g^t, h_{lstm}^{t-1}, c_{lstm}^{t-1})$ 
17: end for
18: return  $h_{lstm}^T$ 

```

The GNN-LSTM module effectively learns edge features by introducing the hidden states of all edges in the graph, thereby achieving efficient reasoning and modeling of the complex relationships between links and nodes. This module not only learns the structural information of nodes and links in the network topology, but also encodes the hidden states. The hidden state is shown in as Table 1 and is represented as $h_i = (x_1, x_2, \dots, x_N)$.

Among them, x_1 to x_3 represent the status of computing resources, including the type of computing resource service, node computing capacity, and load. The network resource status is stored in x_4 – x_6 , including available link bandwidth, delay, and packet loss rate. x_7 – x_N is reserved as a field and is currently zeroed in.

Table 1. The corresponding meaning of the hidden state.

Hidden State	Meaning
x_1	Computing resource service type
x_2	Node computing capacity
x_3	Node load
x_4	Link available bandwidth
x_5	Time delay
x_6	Packet loss rate

4.3. The Basic Components of CR-GLD

CR-GLD employs an SDN architecture and operates within a DRL framework to generate routing policies. The SDN controller first collects global computing-network resource states, which are processed by the GNN-LSTM module to extract spatiotemporal features. The resulting feature vector is provided as the state input to the DRL agent. Based on the current resource state, the agent computes reward signals, selects corresponding actions, and iteratively updates its parameters driven by these rewards. The agent's output actions are defined as optimal path sets between source–destination node pairs. Finally, the SDN controller translates these policy sets into concrete flow-table rules, which are deployed to the Computing First Network plane for execution. The CRGLD intelligent agent operates based on the global resource state and makes routing decisions for all pending requests at each decision point.

The CR-GLD routing process is structured around eight core components, as follows: (1) environment, (2) agent, (3) state, (4) action, (5) reward function, (6) neural network, (7) experience replay buffer, and (8) exploration strategy. During learning, the agent operates as the central decision-making entity—it observes the current state from the environment, evaluates the reward associated with a chosen action, selects the next state to transition to, executes an action according to its policy, and subsequently receives a reward as feedback from the environment.

(1) Environment

In this context, the environment is defined as the Computing First Network plane within the SICRSA architecture. Through continuous interaction with this environment, the agent learns the topological relationships among network nodes and links, thereby establishing a foundation for making informed computational-aware routing decisions.

(2) Agent

As the decision-making entity, the agent engages with the environment through exploratory interactions and observes the resulting outcomes. After executing an action, it receives an immediate reward signal and an updated state of the environment. Based on these observations, the agent evaluates the long-term value—typically expressed as the expected cumulative return—of each state–action pair and accordingly updates the parameters of its internal neural network.

(3) State space

Suppose a network topology has a total of n nodes, where the number of switches (source nodes) connected to the user end is r , the number of computing power routing nodes is r' , the number of switches (destination nodes) connected to the computing power nodes is d , and the number of source–destination node pairs is $r \cdot d$, satisfying $r + r' + d = n$. The agent defines the state space S based on the network topology. Each state contains the state information of the current node and link, as well as the source–object node pair (user request end–computing power node pair). Among them, the status of nodes includes the

type of computing resources, the quantity of computing power resources, etc., while the status of links includes link bandwidth, delay, and packet loss rate, etc.

(4) Action space

During the learning process, the agent maps the state space to the action space. Action Space A consists of a set of actions taken on all states within the state space. Sort all computing nodes in descending order of available computing power. From the sorted list, select the set $C = (c_0, c_1, c_2, \dots, c_m)$ of the top m nodes with the highest available computing power as the candidate nodes for this decision. The computing power routing module randomly selects the k -path set $P = (p_0, p_1, p_2, \dots, p_k)$ using the k -shortest path algorithm and calculates the k -shortest path for each source–destination node pair. For the state $s_t \in S$, the agent can take any action. Each action will cause state s_t to transition to its corresponding state. The agent selects the path corresponding to the state from the i paths connecting the source node and the destination node. Based on the observed states, the agent formulates routing strategies for all newly arrived computing power requests.

(5) Reward function

In the CFN scenario, the agent learns to select the appropriate computing node and the best transmission path. Taking into account the joint optimization of computing resources and network resources comprehensively, computing capacity T_{comp} and network service quality S are two key components in the resource scheduling of CFN. Therefore, the optimization objective is constructed as maximizing the multi-objective utility function composed of computing power and network service quality, expressed as $R = u_1 T_{\text{comp}} + u_2 I$, among them, u_1 and u_2 are normalization coefficients, and u_1 takes a negative value.

Computing capability is quantified as the amount of data processed per second, with reference to the service offloading paradigm in mobile edge computing. Accordingly, the computational time for a single task can be expressed as follows:

$$T_{\text{comp}} = \frac{S_{\text{task}}}{C_{\text{node}}} \quad (13)$$

Here, S_{task} represents the size of the computing task, with the unit being bits (bit); C_{node} represents the computing power of a computing node, with the unit being bits per second (bit/s).

Network service quality I is calculated based on the network resource metric value I_{source} . The network computing management module collects network resource information for all paths through the network perception module and calculates the available bandwidth bw_p , delay $delay_p$, and packet loss rate $loss_p$ of each path, respectively. The calculation method follows the process described in Section 3.1.3.

The setting of the reward function is to increase the available bandwidth, reduce the path delay, and reduce the packet loss rate. Therefore, the setting of network quality of service I is directly proportional to the available bandwidth bw_p of the link and inversely proportional to the latency $delay_p$ and packet loss rate $loss_p$. This effectively guides the agent to learn long-term strategies by maximizing the reward value, adjusting its own behavior, and keeping it consistent with the set goals. The reward function is defined as follows:

$$I = \beta_1 \cdot bw_p + \beta_2 \cdot \frac{1}{delay_p} + \beta_3 \cdot \frac{1}{loss_p} \quad (14)$$

Among them, bw_p , $delay_p$, and $loss_p$ represent the available bandwidth, delay, and packet loss rate of the path respectively, and β_1 , β_2 , and β_3 denote the weights in the reward function, with the value range being $[0, 1]$ for all.

In order to eliminate the dimensional and magnitude differences among different link state indicators and make them comparable. In this paper, the Min–Max interval scaling

method is adopted to normalize the index values, and the range of network resource measurement values is rescaled to the target interval. The Min–Max scaling method maps the original data to the interval $[0, 1]$ through a linear transformation. For the given value x , the transformation is carried out through the following formula:

$$x_{norm} = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \quad (15)$$

Among them, x_{norm} is the normalized value, $x_{\min}$ is the minimum value in the original data, and $x_{\max}$ is the maximum value in the original data.

The reward function after Min–Max scaling is defined as follows:

$$I_{norm} = \beta_1 \cdot bw'_p + \beta_2 \cdot \frac{1}{delay'_p} + \beta_3 \cdot \frac{1}{loss'_p} \quad (16)$$

Among them, bw'_p , $delay'_p$, and $loss'_p$, respectively represent the available bandwidth, delay, and packet loss rate of the normalized path.

Over the long term, the CR-GLD process operates iteratively and adaptively, accommodating the continuous arrival of computational service requests and the dynamic evolution of network states. Through ongoing interaction and learning between the DRL agent and the environment, the routing strategy is progressively refined, converging toward an optimal policy that enhances resource utilization and network service quality—ultimately maximizing the cumulative reward R .

(6) Neural network

The agent uses a dual-neural network structure for reinforcement learning, namely the online network and the target network. Among them, the online network estimates the Q value of the current state S_t . The parameters of this network are updated at each learning step, and the target network is responsible for estimating the Q value of the next state S_{t+1} . After the pre-set number of learning steps, the online network will regularly update the weights of the target network. The weight Settings of the two neural networks are the same, thereby improving the stability of learning. The calculation formula is as follows:

$$Q'(S_t, A) = R + \gamma \cdot \max_A Q(S_{t+1}, A) \quad (17)$$

The loss function is used to train online networks. The current value network is trained at each learning step to reduce the loss function. The parameters of the online network are updated through gradient descent to make the predicted Q value as close as possible to the “target Q value” calculated through the target network. The calculation formula of the loss function is as follows:

$$Loss = (Q'(S_t, A) - Q(S_t, A))^2 \quad (18)$$

The online network and the target network have exactly the same network architecture, including the input layer, hidden layer, and output layer. Each source–object node pair represents a state, and each neuron in the input layer receives a state as the input of the neural network. The number of neurons in the output layer is equal to the size of the action space, and each neuron estimates the Q value corresponding to action a_i , that is, $Q(s, a_i)$.

(7) Memory replay unit

The agent stores past experiences in a dataset of replay memory, represented as $\langle S_t, A_t, R_t, S_{t+1} \rangle$, allowing for small-batch sampling of these experiences and offline training on previously obtained data. This process can reduce the number of interactions and the variance of updates when the agent is learning, thereby promoting the stability of the training process.

(8) Explore and utilize strategies

The agent adopts the $\varepsilon - greedy$ exploration strategy. This strategy uses an adjustable parameter $\varepsilon \in [0, 1]$ to determine whether the agent selects the action with the highest reward value with probability $p = 1 - \varepsilon$ or randomly selects any action with probability $p = \varepsilon$. The value of ε is updated according to the formula, usually set to start from the maximum value $\varepsilon_{\max}$ and gradually decrease at a linear decay rate $decr$ throughout the learning process until it reaches the minimum value $\varepsilon_{\min}$.

$$\varepsilon = \varepsilon_{\max} - steps \cdot decr \quad (19)$$

The agent selects the action in the current state according to the formula. This method allows the agent to try more actions and eventually find the optimal strategy. For each learning process, a random number $x \in [0, 1]$ will be generated. If $x < 1 - \varepsilon$, the agent will choose the action with the smallest reward function value; otherwise, the agent will randomly select an action.

$$A = \begin{cases} \max_A Q_t(S_t, A), & x < 1 - \varepsilon, \\ \text{random}, & \text{else.} \end{cases} \quad (20)$$

4.4. The Algorithm Flow of CR-GLD

The training of the CR-GLD algorithm proceeds over successive rounds. In each round, the GNN-LSTM module embedded in the DRL framework acts as a feature encoder that extracts compact spatiotemporal representations from the raw network state, forming an intelligent routing agent endowed with spatiotemporal awareness. The agent selects actions based on the current environment state; the environment then returns a reward and transitions to a new state. Through repeated iterations of the “state–action–reward–next state” cycle, the agent continuously refines the parameters of its target network, ultimately converging to an optimal policy capable of dynamically adapting to environmental changes.

Algorithm 2 implements the learning process of finding the optimal computing power node and transmission path for all user request ends. The input to Algorithm 2 includes the following parameters: the number of episodes n in the learning period; the maximum value $\varepsilon_{\max}$ of the parameter; attenuation rate $decr$; the number of steps before starting the training using the data stored in the memory playback unit, recorded as the playback start size l_{rss} ; the update frequency f_{tup} of the target neural network; a set of m computing nodes c_m ; a list of k paths p_k ; and network resource measurement values I_{source} . The output is the optimal path set from the user request end to the computing power node. Each path in this set is composed of the state–action pair corresponding to the maximum value in the m table.

First, perform environment initialization, initializing the online network, target network, and memory replay unit, respectively (line 1). The agent conducts multiple cycles of learning according to the steps from line 2 to line 21. During each learning cycle, Algorithm 1 is first invoked to obtain the spatiotemporal fusion features h_{lstm}^T , which are then concatenated with the requested features to construct the state S_t . The agent starts from the initial random state S_t and randomly selects a candidate computing node and path (line 3). After a finite number of operations, it enters the next state until the final state ends (lines 4 to 20). Therefore, a learning cycle consists of a series of steps, traversing each state between the initial state and the final state. Among them, each step includes selecting and performing an action, updating the status, and receiving a reward.

The second stage is the exploration and storage phase. The cycle steps (lines 4 to 20) within each learning period are as follows. The agent first updates the value of parameter ε based on the attenuation rate $decr$ (line 5), and then adopts strategy $\varepsilon - greedy$ to select a

node from the m candidate computing nodes corresponding to the current state and choose a transmission path from the k optional paths (line 6). After performing the selected action, the agent calculates the reward value R_t for selecting the computing node and path in the current state S_t , and then calls Algorithm 1 again to generate a new spatiotemporal fusion feature S_{t+1} , thereby forming the next state h_{lstm}^{T+1} (line 8). Subsequently, the experience of this decision-making is stored in the memory replay unit in the form of $\langle S_t, A_t, S_{t+1}, R_t \rangle$ (line 9). After the agent has completed the storage, it checks whether the number of steps set for the online network training has been reached (line 10).

Algorithm 2 CR-GLD algorithm

Input: number of episodes: n ; explore parameters: decr ; attenuation rate: decr ; playback start size: l_{rss} ; target network update frequency: f_{tup} ; a set of m computing nodes: c_m ; a list of k paths: p_k ; network resource measurement values: l_{source} .

Output: The set of optimal transmission paths for all user request end-computing power nodes.

```

1: Initialize Online network, target network, memory playback unit
2: for episode  $\leftarrow 1$  to  $n$  do
3:    $h_{lstm}^T \leftarrow \text{Algorithm 1}(V, A_{seq}, X_{seq})$ 
4:   Agent acquires the initial state
5:   while  $S_{t+1}$  is not dst do
6:     Update  $\epsilon = \epsilon_{\max} - (\text{steps} \times \text{decr})$ 
7:     Use the  $\epsilon$ -greedy exploration strategy to select action  $A_t$  in state  $S_t$ 
8:     Agent calculates the value  $R_t$  of the reward function based on the current state  $S_t$ 
9:      $h_{lstm}^{T+1} \leftarrow \text{Algorithm 1}(A_{seq+1}, X_{seq+1})$ 
10:    Save Experience  $\langle S_t, A_t, S_{t+1}, R_t \rangle$  to the memory replay unit
11:    if  $\text{steps} > l_{rss}$ 
12:      Randomly extract a small batch of experience from the memory replay unit
13:      Estimate  $Q(S_t, A_t)$  with Online NN
14:      Calculate  $Q'(S_t, A) = R + \gamma \cdot \max_A Q(S_{t+1}, A)$ 
15:      Update the parameters of the online network
16:      if  $\text{steps} \% f_{tup} = 0$ 
17:        Update the parameters of the target network
18:      end
19:      Update the next status  $S_t \leftarrow S_{t+1}$ 
20:    end
21:  end
22: end
  
```

Then comes the network iterative update stage. In each round of training, when the pre-set step count requirement is reached, the agent randomly samples a small batch of data from the memory replay unit to train the online network (line 11). Specifically, the agent uses an online network to estimate the $Q(S_t, A_t)$ of the current state–action pair (line 12) and calculates the target value b based on the target network (line 13). Then, the gradient descent method and the backpropagation algorithm are adopted to adjust the parameters of the online network to minimize the loss value (line 14). If the current number of steps executed meets the update frequency of the target network and a certain amount of experience is stored, the agent uses the weight θ of the online network to update the weight $\hat{\theta}$ of the target network (lines 15 to 17). Subsequently, the intelligent device will enter the next state (line 18) and repeat the above iterative process. After the model training is completed, the agent retrieves the Q values corresponding to all actions in each state and stores the path with the largest Q value as the optimal path between the source–object node pairs in the path library. Each entry in this path library is a triple, recording the source node, the target node, and their corresponding optimal transmission path (line 22).

Once the optimal routing strategy is generated by the computational-aware routing module, the SDN controller translates the corresponding path set into flow-table entries and distributes them to the relevant switches within a configured installation window. The duration of this window depends mainly on the number of forwarding devices and flow entries

being updated. Simultaneously, the DRL agent periodically performs path computation and flow-table installation, synchronized with the regularly updated network resource metrics obtained from the computing-network monitoring cycle. This enables continuous routing-strategy optimization and dynamic adaptation to evolving network conditions.

The complexity of CRGLD can be analyzed in three parts—the GNN module, the LSTM module, and the DRL strategy network. During the inference phase, the trained CRGLD agent must handle a new service request. It involves a forward propagation of the intelligent agent neural network. GNN module: For a network topology graph $G = (V, E)$ with $|V| = N$ nodes and $|E|$ edges, and K message-passing iterations, the time complexity for a single forward pass is $O(K \cdot |E|)$, as each iteration involves aggregating information across edges. This is linear with respect to the number of edges, which is favorable. The space complexity for storing node embeddings is $O(N \cdot d)$, where d is the embedding dimension. LSTM module: Processing the temporal sequence of length T (historical states) for the graph-level embedding has a time complexity of $O(T \cdot H^2)$, where H is the hidden state size. In our architecture, this is applied once to the aggregated graph embedding g^t , not per node, keeping this cost manageable and independent of network size N during inference. DRL Policy Network: The final multi-layer perceptron that maps the fused spatiotemporal features to a Q-value distribution over actions has a complexity dependent on its fixed architecture (e.g., layers, neurons), yielding a constant time complexity $O(1)$ relative to N and $|E|$.

The dominant cost is the GNN forward pass; the time complexity of the CR-GLD algorithm is $O(K \cdot |E|)$. This indicates that the online decision time scales linearly with the number of links in the network, which is a scalable property for large networks. The space requirement is primarily for the trained model parameters and node embeddings.

5. Experimental Results

This section first describes the experimental setup, including the network topologies and traffic generation methods employed. It then outlines the process for determining appropriate parameter values for the CR-GLD algorithm model. Finally, it evaluates the effectiveness of CR-GLD in optimizing network performance and assesses its generalization capability.

5.1. Test Environment

For the experiments, network topologies were selected from the publicly available SNDlib (Survivable Network Design Library) dataset to train and evaluate the CR-GLD algorithm. The selected topologies—Geant, Cost266, and Germany50—were emulated using the Mininet tool. The Geant topology comprises 22 switches and 36 links, representing a large-scale pan-European research network. It tests the algorithm's performance in a complex, well-connected backbone topology with many possible routing paths; the Cost266 topology consists of 37 switches and 57 links, a reference topology for core network planning. It provides a different structural profile, testing the algorithm in a network with potentially different connectivity patterns and bottleneck links compared to Geant, and the Germany50 topology includes 50 switches and 88 links, a national-level topology. It evaluates performance in a denser, more regional network, which is highly relevant for edge computing scenarios where computation and users are geographically closer. Within each topology, client nodes and computing nodes were randomly assigned, with their numbers proportional to the size of the topology; the remaining nodes functioned as routing nodes.

In this experimental setup, the Computing First Network environment was implemented using Python 3.11, while the proposed CR-GLD algorithm was developed through the PyTorch 2.0 framework. The Ryu 3.24 controller was employed to create application pro-

grams for network topology discovery and link state information collection. Computational resources were configured within a range of [50, 200] GHz. Each CFN router processes an average of one computing task per scheduling time slot, with task computational demands ranging from [5, 50] GCPU. The experiment considers an extended scheduling process comprising 30 consecutive time slots, each lasting 1 s. Service requests are generated within this environment, with each request specifying the source node and computational requirements. These computational resource requests are initiated from user terminals and include both the type of computational service required and the specific computational resource demands.

This experiment conducts a comparative evaluation of three models: the proposed CR-GLD algorithm, an ablated variant (CR-GLD w/o LSTM), and a classical Deep Q-Network (DQN) baseline. CR-GLD w/o LSTM is a routing algorithm that combines GNN with DQN: it extracts spatial features of the network topology through GNN to capture node-link relationships, and uses these features as input to DQN. Unlike the full CR-GLD model, this variant does not incorporate LSTM to learn temporal dynamics in the network. DQN serves as a standard Deep Reinforcement Learning baseline. All three algorithms share the same agent design and decision-making scope, including computing-node selection and path planning.

5.2. Traffic Generation

To simulate dynamic network conditions, this experiment employs the real traffic matrix from the SNDlib dataset. This matrix consists of fine-grained measurements of actual IP network traffic, formatted as an array that records traffic volumes between node pairs at successive time points, thereby exhibiting both temporal and spatial characteristics. During the experiments, the obtained traffic is injected into the network to induce dynamic variations in resource states, ensuring that all compared algorithms are evaluated under identical traffic scenarios.

5.3. Parameter Setting

When designing the training strategy of the agent, this algorithm model adopts the Adaptive moment estimation optimizer [26]. The Adam optimizer improves the gradient descent algorithm, enabling the training of deep networks to converge rapidly. In addition, the Adam optimizer is also very easy to implement and requires only a small amount of memory. This algorithm model uses the Rectified Linear Unit function [27] to initialize the network parameters. During the training and evaluation phases, this experiment did not take into account different QoS requirements. Therefore, setting the adjustable parameters in the reward function of Equation (14) to 1 gives the same importance to the optimization of the three indicators that affect the reward calculation. It creates a neutral, balanced optimization target, allowing us to fairly demonstrate and compare the inherent ability of CRGLD versus baseline algorithms (DQN, CRGLD w/o LSTM) to improve across all three fundamental network QoS dimensions simultaneously. The equal-weight reward served as a stable and unambiguous objective function to train and evaluate this new architecture, ensuring that performance differences stemmed from the learning mechanism itself rather than a complex, tuned reward shaping. In the process of intelligent agent training, in order to determine specific learning parameters, the convergence performance of reward values in each training set is used as a key comparison indicator to observe the influence of different parameters on the convergence of reward values. At the same time, common DRL design principles are followed, and it is observed that they contribute to achieving stable convergence in our framework. The parameter values of the

model were ultimately determined. Table 2 summarizes the parameter values used in this algorithm model.

Table 2. The parameter values of the CR-GLD algorithm.

Parameter	Value
The number of hidden layers in DQN	1
The number of neurons	128
Number of LSTM layers	1
Attenuation rate	1/400
Discount rate	0.0001
The size of the experience playback buffer	50,000
Batch size	32

5.4. Performance Evaluation

The simulation experiment evaluates the proposed CR-GLD algorithm along two dimensions—effectiveness and generalization capability. Specifically, effectiveness refers to the algorithm’s ability—when tested on multiple known training topologies—to schedule computational services to suitable computing nodes and improve the quality of network services. Generalization capability measures the stability and adaptability of the algorithm’s performance when applied to entirely new network topologies that were not encountered during the training phase.

5.4.1. Validity

To verify the effectiveness of the proposed algorithm, during the training phase, this experiment uses three classic topologies and real traffic matrices from the SNDlib dataset. The agent is trained for 2000 sets, with 50 iterations per set. Based on the current model, multiple experiments are conducted to verify convergence and performance. Experiments show that 50 iterations are sufficient to obtain effective and stable results.

Once the model reaches convergence, the experiment proceeds to the testing phase. The test evaluates whether the proposed CR-GLD algorithm can effectively schedule computational tasks to appropriate computing nodes and select the optimal transmission path by maximizing cumulative reward. By comparing its performance against two benchmark algorithms, the proposed model is shown to achieve superior results.

During the testing phase, the agent is evaluated over 100 episodes. The cumulative reward obtained after 50 iterations within each episode serves as the primary performance metric. This metric captures the total return achievable by the model’s policy over a sequence of 50 consecutive, complete decisions, directly reflecting the quality of the learned strategy. A higher cumulative reward indicates more effective decisions overall, corresponding to larger available link bandwidth, lower latency and packet loss, and reduced computational service time. For a clearer comparison, cumulative rewards are normalized. These values represent the integrated performance considering both computational and network resources, with the mean reward across episodes indicated by a dashed line.

As shown in Figure 8, the proposed CR-GLD algorithm delivers significantly higher cumulative reward than both the CR-GLD w/o LSTM and DQN baselines on the Geant topology, with a particularly pronounced gap relative to DQN. Specifically, under the Geant topology, the average normalized cumulative reward reaches 0.582 for CR-GLD, compared to 0.503 for CR-GLD w/o LSTM and 0.351 for DQN. This corresponds to a performance improvement of 15.71% over CR-GLD w/o LSTM and 65.81% over DQN.

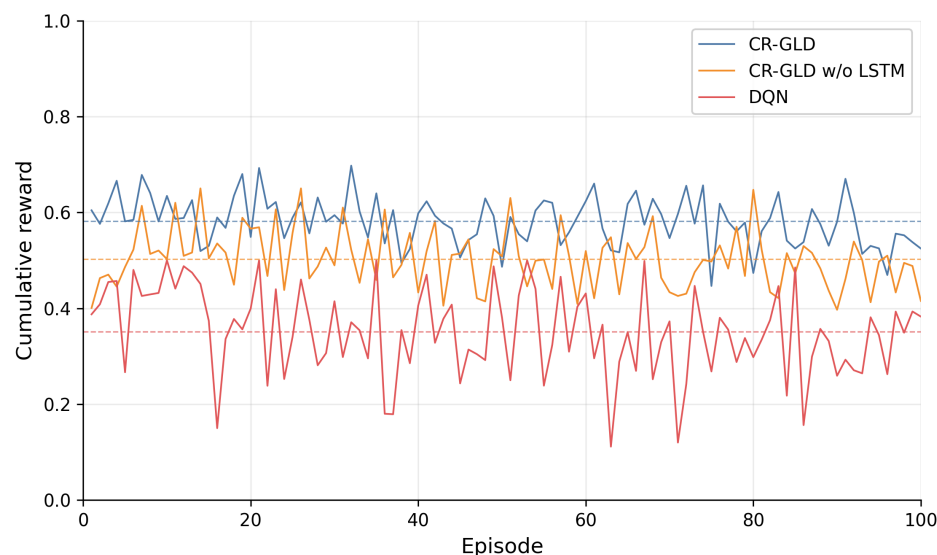


Figure 8. Algorithm comparison chart under the Geant topology.

Under the Cost266 topology shown in Figure 9, the CR-GLD algorithm achieves an average cumulative reward of 0.626, compared to 0.527 for CR-GLD w/o LSTM and 0.324 for DQN. This corresponds to a performance improvement of 16.67% over CR-GLD w/o LSTM and an approximate 93% improvement over DQN.

As illustrated in Figure 10 for the Germany50 topology, the CR-GLD algorithm attains an average cumulative reward of 0.588, whereas CR-GLD w/o LSTM and DQN yield 0.504 and 0.324, respectively. This translates to a performance improvement of 18.79% over CR-GLD w/o LSTM and approximately 81.5% over DQN.

The analysis of the experimental results indicates that the proposed CR-GLD algorithm outperforms both the CR-GLD w/o LSTM variant and the DQN baseline. This superiority stems from CR-GLD's ability to jointly learn spatial topology features via GNN and temporal dynamics via LSTM, achieving effective spatiotemporal fusion. Consequently, it selects more suitable computing nodes and transmission paths, thereby providing stronger guarantees on link-level quality of service. By contrast, the CR-GLD w/o LSTM variant lacks the temporal modeling capability of LSTM and cannot fully capture time-varying network dynamics, which results in moderately lower performance. DQN, although a Deep Reinforcement Learning method, does not incorporate explicit spatiotemporal feature extraction and cannot directly learn from non-Euclidean graph data; it therefore exhibits the weakest performance among the three approaches.

5.4.2. Generalization Ability

To evaluate the generalization capability of the CR-GLD algorithm across different network topologies, this experiment adopts the Eastern Massachusetts (EMA) topology and the corresponding real traffic matrix from the publicly available Transportation Networks dataset. The EMA topology comprises 74 switches and 258 links. The agent performed 100 test episodes under this previously unseen topology. As shown in Figure 11, CR-GLD consistently outperforms both reference algorithms on the EMA topology. Specifically, it achieves an average cumulative reward of 0.518, compared to 0.404 for CR-GLD w/o LSTM and 0.174 for DQN.

To verify the generalization ability of the CR-GLD algorithm on dynamically changing network topologies, in this experiment, the India35 network topology and the real traffic matrix were selected from the SNDlib dataset, and the agent conducted 50 sets of tests. To simulate the scenario when network links are interrupted or fail, in this experiment,

some links are randomly deleted. When the agent is trained on the network topology, the environment will change dynamically. It can be seen from Figure 12 that the original average cumulative reward value of the CR-GLD algorithm is 0.524. When one link is randomly deleted, the average cumulative reward value is 0.533. When two links are randomly deleted, the average cumulative reward value is 0.549. When three links are randomly deleted, the average cumulative reward value is 0.55. Therefore, the cumulative reward value of the agent remains nearly at the same level.

The above experimental results show that the proposed CR-GLD algorithm has a better generalization ability when facing different network topologies or even when the network topology links change. This is because the algorithm adopts a DRL agent based on GNN to learn the graph relationship between nodes and links. Therefore, even when encountering topologies not seen during training, the agent can still conduct effective reasoning based on the learned graph structure relationships.

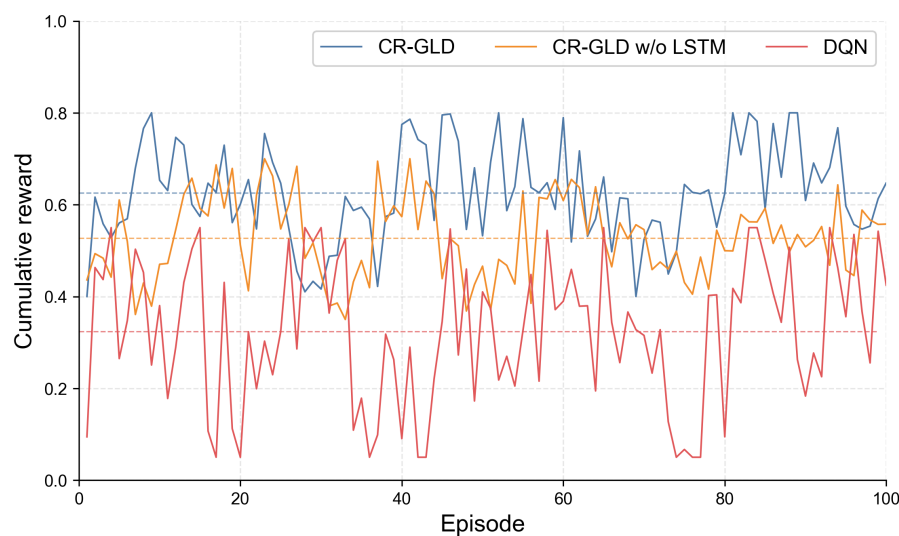


Figure 9. Algorithm comparison chart under Cost266 topology.

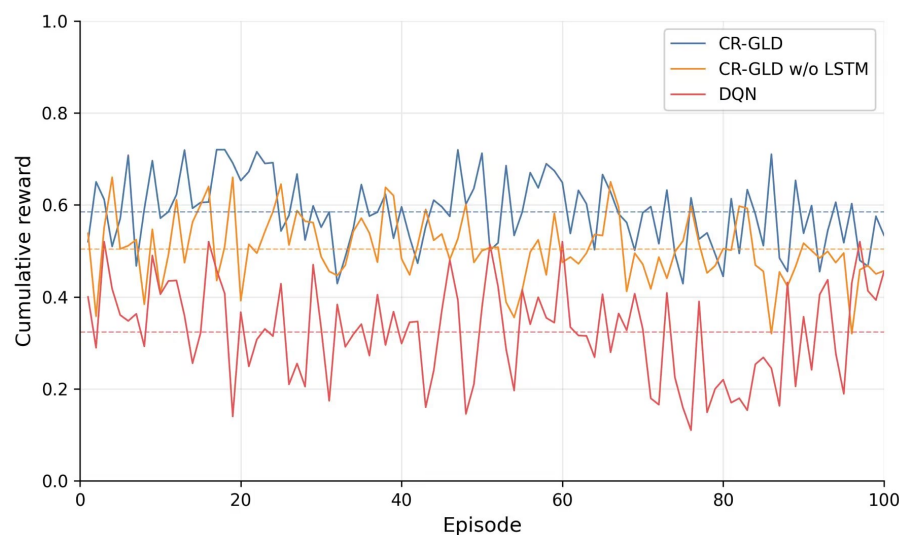


Figure 10. Algorithm comparison chart under Germany50 topology.

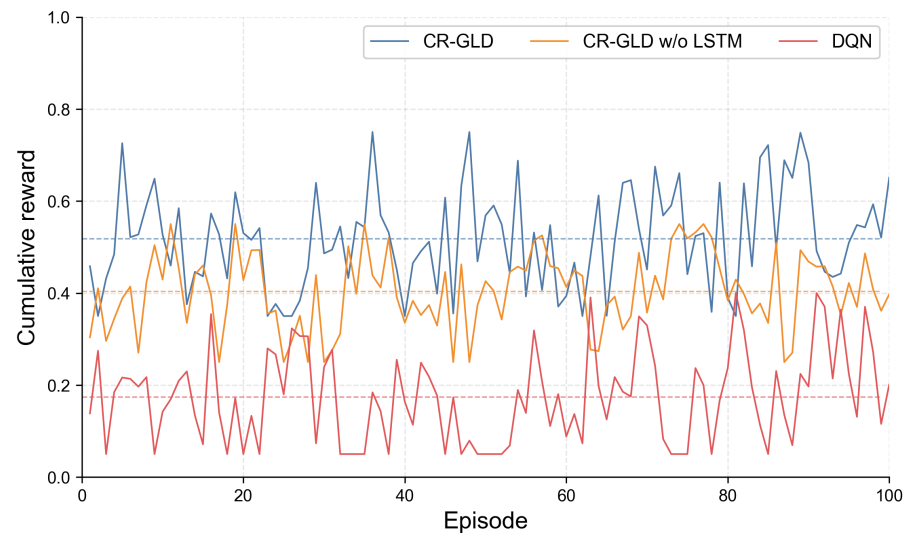


Figure 11. Algorithm comparison chart under EMA topology.

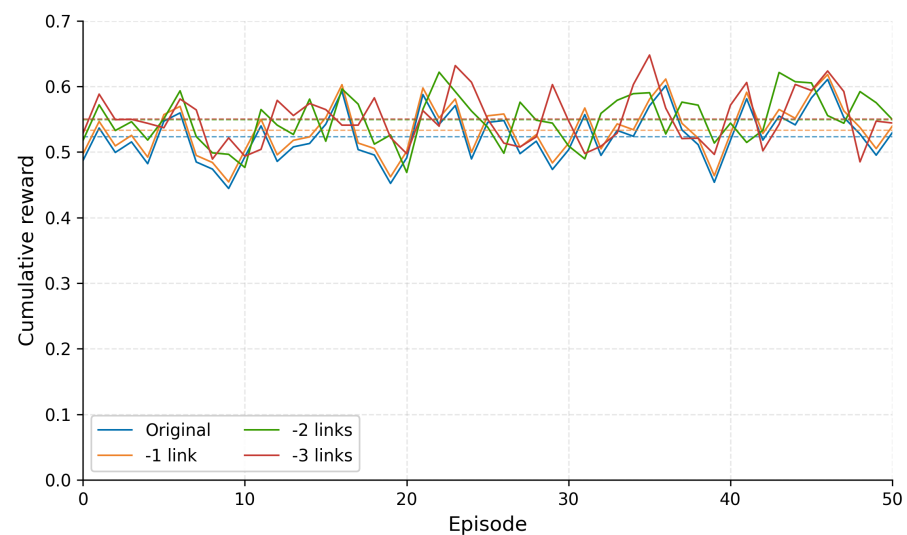


Figure 12. Experiment on randomly deleting links in the India35 topology.

6. Conclusions and Future Work

This paper proposes SICRSA, a computing resource scheduling architecture for CFN that integrates SDN and ICN. The architecture's modular design and end-to-end workflow are described in detail. Building on this foundation, we further introduce an ICN-based hierarchical naming scheme for computational services, design an IP-header-extended packet format for service requests, and specify the corresponding request identification procedure, thereby implementing a user-oriented resource scheduling mechanism. The architecture effectively avoids single points of failure and alleviates scalability bottlenecks, thus enhancing the reliability and flexibility of resource scheduling in CFN.

In addition, we present CR-GLD, a computational-aware routing optimization algorithm developed within the SICRSA framework. The algorithm incorporates a GNN-LSTM module to effectively extract spatiotemporal fusion features from network topology. Within a Deep Reinforcement Learning framework, it employs a multi-dimensional reward function that holistically considers both computational and network resources, guiding the agent toward cooperative resource optimization during learning. Simulation experiments conducted across diverse network topologies systematically validate CR-GLD in terms of both performance optimization and generalization capability. The results demonstrate that

CR-GLD outperforms two benchmark algorithms in comprehensive performance and exhibits superior generalization and robustness when confronted with unfamiliar topologies and dynamic link variations. The proposed approach thus offers an effective and practical solution for routing optimization in Computing First Networks.

Building upon these findings, we hope that future work can be implemented and evaluated in testing environments with real hardware, traffic changes, and concurrent scheduling to further validate the generalization ability of the SICRSA architecture and CRGLD algorithm. We also suggest that future work conduct a comprehensive parameter sensitivity analysis and explore more intelligent fusion models to quickly adapt to dynamic topology changes.

Author Contributions: Conceptualization, Q.X. and S.R.; formal analysis, software, writing, Q.X. and S.R.; supervision, Z.L. and W.G.; investigation and methodology, J.W.; validation and visualization, Q.X. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The code of this article is open source at <https://github.com/xqw62/CRGLD/> (and will be available online in 26 March 2026).

Conflicts of Interest: The authors declare no conflicts of interest.

References

- Esposito, C.; Castiglione, A.; Pop, F.; Choo, K.-K.R. Challenges of Connecting Edge and Cloud Computing: A Security and Forensic Perspective. *IEEE Cloud Comput.* **2017**, *4*, 13–17. [\[CrossRef\]](#)
- Ahmed, M.; Ashraf Hossain, M. Cloud Computing and Security Issues in the Cloud. *Int. J. Netw. Secur. Appl.* **2014**, *6*, 25–36. [\[CrossRef\]](#)
- Gong, X.; Bai, C.; Ren, S.; Wang, J.; Wang, C. Cloud-Network Resource Perception and Modeling Technology Based on Software-Defined Computing First Network. In Proceedings of the 2023 3rd International Conference on Computer Science, Electronic Information Engineering and Intelligent Control Technology (CEI), Wuhan, China, 15–17 December 2023; pp. 6–12.
- Aldaoud, M.; Al-Abri, D.; Awadalla, M.; Kausar, F. Leveraging ICN and SDN for Future Internet Architecture: A Survey. *Electronics* **2023**, *12*, 1723. [\[CrossRef\]](#)
- Guesmi, T.; Kalghoum, A.; Alshammari, B.M.; Alsaif, H.; Alzamil, A. Leveraging Software-Defined Networking Approach for Future Information-Centric Networking Enhancement. *Symmetry* **2021**, *13*, 441. [\[CrossRef\]](#)
- Zali, Z.; Hashemi, M.R.; Cianci, I.; Grieco, A.; Boggia, G. A Controller-Based Architecture for Information Centric Network Construction and Topology Management. *China Commun.* **2018**, *15*, 131–145. [\[CrossRef\]](#)
- Imran, M.; Ali, M.N.; Din, M.S.U.; Rehman, M.A.U.; Kim, B.-S. An Efficient Communication and Computation Resources Sharing in Information-Centric 6G Networks. *IEEE Internet Things J.* **2024**, *11*, 27275–27294. [\[CrossRef\]](#)
- Ni, Z.; You, J.; Li, Y. An ICN-Based On-Path Computing Resource Scheduling Architecture with User Preference Awareness for Computing Network. *Electronics* **2024**, *13*, 933. [\[CrossRef\]](#)
- Król, M.; Mastorakis, S.; Oran, D.; Kutscher, D. Compute First Networking: Distributed Computing meets ICN. In Proceedings of the 6th ACM Conference on Information-Centric Networking, Macau, China, 24–26 September 2019; ACM: New York, NY, USA, 2019; pp. 67–77.
- Liu, B.; Mao, J.; Xu, L.; Hu, R.; Chen, X. CFN-dyncast: Load Balancing the Edges via the Network. In Proceedings of the 2021 IEEE Wireless Communications and Networking Conference Workshops (WCNCW), Nanjing, China, 9 March–1 April 2021; pp. 1–6.
- Han, X.; Zhao, Y.; Yu, K.; Huang, X.; Xie, K.; Wei, H. Utility-Optimized Resource Allocation in Computing-Aware Networks. In Proceedings of the 2021 13th International Conference on Communication Software and Networks (ICCSN), Chongqing, China, 4–7 June 2021; pp. 199–205.
- Yao, H.; Duan, X.; Fu, Y. A Computing-Aware Routing Protocol for Computing Force Network. In Proceedings of the 2022 International Conference on Service Science (ICSS), Shenzhen, China, 2–4 July 2022; pp. 137–141.
- Ding, H.; He, F.; Zhang, P.; Zhang, L.; Zhang, X.; Qi, M. Optimal Routing and Heterogeneous Resource Allocation for Computing-Aware Networks. *ICT Express* **2024**, *10*, 614–619. [\[CrossRef\]](#)

14. Zhao, Y.; Chong, Z.; Han, X.; Du, Z.; Yu, K.; Huang, X. Simulation Study of Routing Mechanism in the Computing-Aware Network. In Proceedings of the 2021 the 10th International Conference on Networks, Communication and Computing, Virtual, 10–12 December 2021; ACM: New York, NY, USA, 2021; pp. 126–134.
15. Lin, Z.; Wang, L.; Ning, W.; Zhao, Y.; Yu, L.; Jiang, J. Heuristic-Based Computing-Aware Routing for Dynamic Networks. *Electronics* **2025**, *14*, 3724. [[CrossRef](#)]
16. Liu, J.; Sun, Y.; Su, J.; Li, Z.; Zhang, X.; Lei, B.; Wang, W. Computing Power Network: A Testbed and Applications with Edge Intelligence. In Proceedings of the IEEE INFOCOM 2022—IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), Online, 2–5 May 2022; pp. 1–2.
17. Dai, Y.; Zhu, Y. A Transferable-Experience-Based Routing Mechanism for Named Computing Power Network. *IAENG Int. J. Comput. Sci.* **2025**, *52*, 1235.
18. Xie, Y.; Huang, X.; Li, J.; Liu, T. Computing Power Network: Multi-Objective Optimization-Based Routing. *Sensors* **2023**, *23*, 6702. [[CrossRef](#)] [[PubMed](#)]
19. Gong, X.; Ren, S.; Wang, C.; Wang, J. Research on Computing Resource Measurement and Routing Methods in Software Defined Computing First Network. *Sensors* **2024**, *24*, 1086. [[CrossRef](#)]
20. Qi, J.; Liu, C.; Xu, C.; Wang, R.; Dong, J.; Yu, Y. Efficient Information Updates in Compute-First Networking via Reinforcement Learning with Joint AoI and VoI. *arXiv* **2025**, arXiv:2505.06025. [[CrossRef](#)]
21. Li, M.; Yang, L.; Yu, F.R.; Wu, W.; Wang, Z.; Zhang, Y. Joint Optimization of Networking and Computing Resources for Green M2M Communications Based on DRL. In Proceedings of the 2019 IEEE Global Communications Conference (GLOBECOM), Big Island, HI, USA, 9–13 December 2019; pp. 1–6.
22. Yu, S.; Chen, X.; Zhou, Z.; Gong, X.; Wu, D. When Deep Reinforcement Learning Meets Federated Learning: Intelligent Multi-Timescale Resource Management for Multi-Access Edge Computing in 5G Ultra-Dense Network. *IEEE Internet Things J.* **2020**, *8*, 2238–2251. [[CrossRef](#)]
23. Bai, C.; Gong, X.; Ren, S.; Xu, T.; Wang, J. A Routing Algorithm Based on Software-Defined Computing First Network. *J. Phys. Conf. Ser.* **2023**, *2670*, 012017. [[CrossRef](#)]
24. Feng, L.; Xie, R.; Tang, Q.; Huang, T.; Xiong, Z.; Chen, T.; Zhang, R.; Tan, S.; Fang, Z. CaRCS: Joint Optimization of Computing-Aware Routing and Collaborative Scheduling in Computing Power Networks. *IEEE Netw.* **2025**, *39*, 270–278. [[CrossRef](#)]
25. Asmat, H.; Din, I.U.; Almogren, A.; Rodrigues, J.J.P.C. Reinforcement Learning-Based Edge Caching for IoT in Compute-First Networks. *IEEE Trans. Consum. Electron.* **2025**, early access. [[CrossRef](#)]
26. Kingma, D.P.; Ba, J. A Method for Stochastic Optimization. *arXiv* **2014**, arXiv:1412.6980. [[CrossRef](#)]
27. Hara, K.; Saito, D.; Shouno, H. Analysis of Function of Rectified Linear Unit Used in Deep Learning. In Proceedings of the 2015 International Joint Conference on Neural Networks (IJCNN), Killarney, Ireland, 12–17 July 2015; pp. 1–8.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.