Routing Planning for Video Transmission in Cloud Content Delivery Networks Based on Q-Learning

Pingshan Liu[1,2], Yemin Sun[2(✉)], Chunyan Qin[1], and Yiqian Liu[1]

[1] Business School, Guilin University of Electronic Technology, Guilin, China
[2] Guangxi Key Laboratory of Trusted Software, Guilin University of Electronic Technology, Guilin, China
2198371783@qq.com

Abstract. In the cloud content delivery networks, when an edge CDN node lacks the video requested by a user, it needs to send video requests to the origin server or other edge CDN nodes. To enhance user experience quality, the target node that receives the video request needs to establish a low-latency video transmission path to send videos to the requesting node. However, existing video transmission strategies have not fully considered the dynamic network congestion status within the cloud content delivery network. Therefore, this paper proposes a Q-learning based Adaptive Video Routing algorithm (Q-AVR) specifically for video transmission issues within cloud content delivery networks. It aims to reduce end-to-end video transmission latency and improve network bandwidth utilization. In the video transmission path construction process, edge CDN nodes exchange information through sending data packets. Each node stores the information in a Q-table and makes routing decisions based on the Q values. This algorithm optimizes end-to-end video transmission by learning network status in real-time. After simulation validation, the results show that the Q-AVR algorithm can effectively reduce end-to-end video transmission latency and improve network bandwidth utilization.

Keywords: Cloud Content Delivery Networks · Video Transmission · Q-learning · End-to-End delay

1 Introduction

With the rapid development of the Internet and the swift growth of smart terminal devices, video streaming applications account for a substantial share of Internet traffic. Cisco predicts that by 2022, video traffic will account for 82% of all consumer traffic [1]. The traditional Content Delivery Network (CDN) has achieved the goals of reducing latency and improving access speed by deploying content on edge servers close to users [2]. However, with the diversification and dynamism of user demand, the deployment and management of traditional CDNs have become increasingly complex, leading to the

F. Gao et al. (Eds.): Chinacom 2023, LNICST 590, pp. 459–477, 2024.
https://doi.org/10.1007/978-3-031-67162-3_29

emergence of Cloud Content Delivery Networks (CCDN) [3]. CCDNs apply cloud computing technology to content distribution, providing a flexible and scalable solution. By deploying multiple cloud storage servers in the cloud, content providers can intelligently choose and adjust content distribution strategies based on coverage, budget, and QoS preferences. This enables CCDNs to more effectively meet user demand for high-quality content and adapt to the dynamic changes in network conditions and bandwidth [4].

In CCDN, when an edge CDN node does not cache the video requested by the user, it needs to send a video request to the origin server or other nodes. To improve the quality of user experience, the target node receiving the video request builds a low-latency video transmission path to send the video to the requesting node [5]. Existing video transmission strategies typically perform well under static or predictable network environments, but perform poorly in dealing with the dynamic changes of network congestion in CCDN. These strategies often lack real-time network status awareness, unable to adjust routing choices in time to adapt to congestion changes. Additionally, existing strategies have limitations in the utilization of network bandwidth resources and cannot achieve optimal allocation of bandwidth and computational resources [6]. Therefore, how to reduce end-to-end video transmission latency and improve network bandwidth utilization in the CCDN environment remains a challenging problem.

Reinforcement Learning (RL) is a machine learning method that learns the optimal strategy to achieve a goal in a given environment through the interaction of an agent with the environment. In reinforcement learning, an agent observes the state of the environment, takes actions, and receives corresponding rewards or punishments, thereby continuously updating its strategy [7]. The core of reinforcement learning is to learn an optimal policy that allows the agent to obtain the maximum cumulative reward in the long term. In cloud content delivery networks, reinforcement learning can be applied to adaptive routing selection problems. By viewing nodes in the CCDN as agents, network status as environment status, video transmission latency and bandwidth utilization as reward or punishment signals, reinforcement learning algorithms can learn network status in real-time to optimize end-to-end video transmission.

The Q-Adaptive Video Routing (Q-AVR) algorithm proposed in this paper is a reinforcement learning method. Q-learning is a value iteration algorithm that learns a function called Q-value to estimate the expected return of taking certain actions under a given state. In CCDN, this algorithm can learn network status in real time to optimize end-to-end video transmission. In the Q-AVR algorithm, edge CDN nodes exchange information by sending data packets to each other, then each node stores the information in the Q-table, and makes routing decisions based on Q-values. Therefore, this algorithm can adapt to dynamically changing networks, choose paths with low congestion costs for data transmission, effectively reduce network transmission latency and improve video transmission performance.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 introduces the system model and describes the problem to be solved. Section 4 mainly introduces the Q-AVR algorithm and related pseudocode. Then, in Sect. 5, we present simulation results to evaluate the algorithm we proposed. Finally, conclusions are drawn in Sect. 6.

2 Related Work

There is relatively limited research in the field of cloud content delivery networks, particularly in relation to video requests to the origin server or other CDN servers when videos are missing from the edge CDN servers. As a result, this section will introduce relevant work from two perspectives. On the one hand, we will focus on research about content delivery network, especially on the issue of content placement, studying how to reduce the cost of content distribution. The methodologies involved in these studies include strategies such as optimal replicas, multicast trees, and cooperative reinforcement learning. On the other hand, we will look at research aimed at optimizing video transmission. By planning routes for video transmission paths, we aim to reduce both the cost and latency of video transmission.

Regarding content placement in traditional CDNs, Kangasharju et al. formulate it as a combinatorial optimization problem and propose a heuristic algorithm for solving optimal replica placement [8]. Xu et al. proposed a joint optimization strategy to alleviate the request burden of the content delivery networks [9]. Gamehi et al. proposed the integration of CDN-P2P hybrid networks with CDN and P2P technologies to combine their respective advantages [10]. In this model, they create and maintain a multicast tree, transmitting content from the origin server to edge servers, clients, and peer nodes.

In the study of cloud content delivery networks, Sajithabanu et al. introduced a video content distribution model based on shared storage cloud CDN (SS-CCDN), which improves the quality of content distribution services and user experience through the optimization of effective placement and dynamic updates of video data [11]. Hu et al. proposed a dynamic method that reduces total traffic and delay costs with the aim of lowering operational costs by solving an optimization problem [12]. Wang et al. proposed a hybrid video replication framework assisted by edge cloud and nodes, where video is replicated among edge cloud servers and nodes in different geographic regions [13]. Zhang et al. proposed a cloud-optimized video distribution service deployment strategy and explored in depth the balance between operational costs and user experience [14]. Considering the dynamic characteristics of mobile users in cloud video distribution networks, Lu et al. designed a session-based cloud video distribution network framework that significantly reduces costs [15]. Shorfuzzaman et al. optimized the resource placement model and determined from a user and system perspective the location of the minimum quantity of portions that meet the quality requirements in some cases [16]. The placement algorithm proposed by the authors is based on the data access history of popular data files, and calculates the replication position by minimizing the overall replication cost and increasing the QoS satisfaction of traffic patterns. Gong et al. [17] proposed a multicast tree construction technique based on the Steiner tree problem, adopting a concise distribution model to build the multicast tree, achieving low cost and low energy consumption. Haghighi et al. [18] designed a QoS-based greedy heuristic algorithm to optimize the placement of replicas in CCDN.

However, due to the elasticity and flexibility of cloud content delivery networks, the aforementioned methods are not entirely suitable for dealing with content distribution issues in dynamic cloud content delivery networks. In order to deal with the dynamic changes in cloud CDN network congestion, many scholars have conducted in-depth

research and discussion. He et al. [19] built a new CCDN framework based on cooperative reinforcement learning, capturing the dynamic characteristics of cloud content delivery networks using the V-value feedback model, and using the Q-value to measure node transmission efficiency. Based on this framework, they proposed a transmission tree construction algorithm based on cooperative reinforcement learning to improve the efficiency of content distribution. Chen et al. [20] proposed a cost optimization model for coupled video delivery, taking into account performance indicators such as bandwidth, delay, and personalized requirements, and minimized delivery costs in terms of bandwidth consumption, delay performance, and personalized requirements. Liu et al. [6] considered dynamic network characteristics and designed a dynamic CCDN content placement model based on Q-learning, effectively reducing the overall congestion cost of content placement by constructing a Q-adaptive delivery tree to adapt to the dynamic changes in CCDN.

In the Cloud Content Delivery Network, the process of edge CDN servers requesting missing videos can cause certain transmission delay, thereby affecting the quality of the end-user experience. Therefore, in order to reduce the end-to-end delay of video transmission and improve the quality of the user experience, many scholars have done a lot of work on this. Cherkasova et al. [21] proposed a content distributed replication algorithm for large-scale network files. This algorithm allows cloud nodes to divide the large files to be transmitted into appropriately sized subfiles, and distribute these subfiles to edge CDN servers located in different geographical areas. Then, the corresponding multicast trees are established, and the multicast trees on different nodes exchange their corresponding subfiles through the attached cross-node links. In this way, this file replication strategy ensures that each node can eventually receive the complete file with lower latency. Liu et al. suggested that nodes missing videos run low-latency or low-cost video request algorithms according to the urgency of the video segment, but this algorithm does not consider the dynamic changes in network congestion in the cloud content delivery network [5]. Chandy et al. [22] designed a copy placement strategy based on video fragments. By logically pairing resources between end users and nodes in the system, the number of hops required for end-user requests was reduced, effectively reducing transmission delay. In the context of video stream transmission in data centers, Laoutaris et al. [23] designed and implemented the NetSwitcher system for non-real-time applications, such as backup and data migration. This system utilizes the geographical distribution of cloud data centers to have different peak bandwidths at different times and the cyclical changes in bandwidth usage, to transmit and store in data centers along paths with lower bandwidth utilization, thereby reducing the bandwidth cost of data transmission.

3 System Model and Problem Description

In this section, we primarily introduce the system model of CCDN and the problem description of video transmission. The important symbols used are shown in Table 1.

Table 1. Notations and definitions

Symbol	Description
S	the State space
A	the Action space
$R(s, a)$	the Reward function
$Q(i, a)$	the Q-value function
$F(j)$	the F-value function
λ	the learning rate
$aviband_{ij}$	the available bandwidth on link {i, j}
γ	the Discount factor
D	the set of target nodes
O	the source server
ε	the action selection probability

3.1 System Model

The architecture of the CCDN network is composed of end-users, source servers, and edge CDN servers. As shown in Fig. 1, both the source servers and the edge CDN servers are the same type of server. The source server is responsible for generating and delivering content, while the edge CDN server, located near users, receives and stores content from the source server. End users obtain the required content from the nearest server.

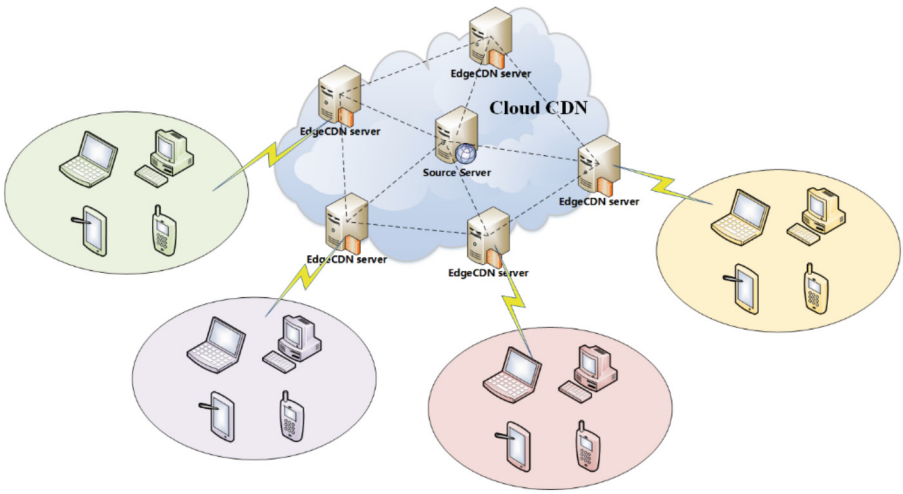

Fig. 1. CCDN network architecture

3.2 Problem Description

Servers are interconnected via high-speed links, forming a highly interconnected network. Within the cloud CDN, users obtain the desired video content by sending video requests. When a server receives a video request and doesn't store the requested video, it needs to send a request to another server that does have the video. In order to improve the quality of user experience, the server that stores the video needs to transmit the video to the requesting server with low latency.

In the cloud content delivery network video transmission model constructed in this paper, there are request nodes, target nodes, and intermediate nodes. The target node stores the video data that the request node lacks and transmits the content to the request node upon receiving a video request. Other nodes act as relay nodes involved in the transmission process. The model diagram of the video transmission path is shown in Fig. 2.

In the CCDN, network congestion varies with the number of packets handled by the cloud CDN servers. Traditional video transmission methods have difficulty in obtaining global congestion information. They can only provide relatively fixed routing paths based on local decisions. Therefore, considering the dynamic changes in network congestion, this paper uses an algorithm based on Q-learning to calculate the best transmission strategy for each node in the network. This dynamic learning method enables the algorithm to adapt to complex and changing network environments, improving video transmission efficiency. The target node builds the video transmission path according to the best strategy learned and transmits the video to the request node along this path. During the problem-solving process, the optimal transmission strategy is adjusted and updated in real-time to adapt to the dynamic nature of the network. By optimizing latency and bandwidth utilization, we can achieve higher resource utilization efficiency. This means

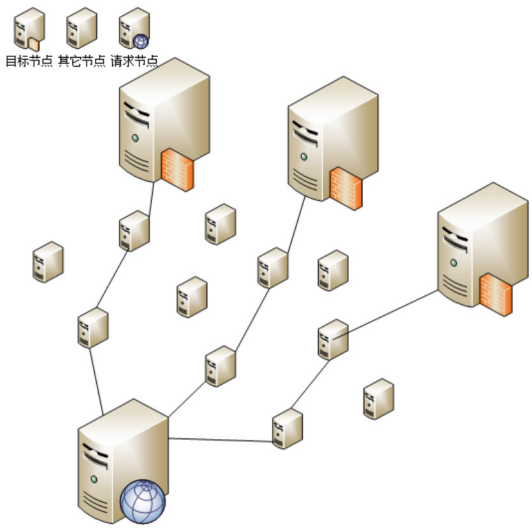

Fig. 2. Video transmission path model diagram

that while maintaining video transmission quality, fewer network resources can be used, thereby reducing overall costs.

4 Algorithm Design and Description

4.1 Related Definitions of Video Transmission Algorithm Based on Q-Learning

In this section, we introduce the proposed Q-learning-based Adaptive Video Routing algorithm (Q-AVR). First, we define the CCDN as a directed graph G = (V, E), where V represents the vertex set, indicating the nodes in the network, and E represents the edge set, indicating all possible connections in the network. Next, we introduce the related concepts in the Q-AVR algorithm.

Definition 1: State space. We use S to represent the network state space, which consists of all nodes in the network. Each node (e.g., $v \in V$) represents a state s, and state $s \in S$ transitions to the next state s' by executing action a. In the problem-solving process of this paper, each step of the decision-making process requires the current node to select an adjacent node and send data to it.

Definition 2: Action space. The action space includes all possible action choices taken to transition from one state to another state, with action choices representing which neighboring node the current node selects as the next hop. Suppose a node has k neighbors; then, the action space size for that node is k. In other words, the action space is the set of all selectable edges for the current node.

Definition 3: Target node. Since the request node will be missing many videos, and these missing videos may be present in multiple different nodes, we refer to these nodes as target nodes.

Definition 4: State transition and reward function. Based on the current node's state and the selected action (neighboring node), calculate the end-to-end video transmission latency and link bandwidth utilization. The reward function is the weighted sum of latency and available link bandwidth. State transitions are updated according to the current node's selected action (i.e., the next-hop neighbor node) and the state of that neighbor node.

Definition 5: Reward function. Each node in the network represents its understanding of the network condition through its own Q-table. A node's action is to find the node with the best expected transmission performance among its neighboring nodes, thereby efficiently transmitting video data to the request node. To ensure that the path selection at each step finds the node with the best expected transmission performance, the reward function in this paper consists of two parts, which take into account both the current available bandwidth and latency of the link. The specific reward function formula is as follows:

$$R(s, a) = (\alpha * delay_{ij} - \beta * availband_{ij}) \tag{1}$$

In the above formula, α and β are weight parameters used to balance the impact of network latency and link available bandwidth. $delay_{ij}$ Represents the sum of the queuing delay of the data packet in node i's queue and the transmission delay between nodes j and i. $availband_{ij}$ represents the available bandwidth of the link between nodes j and i. Please note that a subtraction is used here because we want to minimize latency while maximizing available bandwidth. Depending on the different types of video transmission requirements, the values of α and β can be adjusted flexibly to meet various performance demands. By optimizing latency and bandwidth utilization, we can achieve higher resource utilization efficiency. This means that, while maintaining video transmission quality, we can use fewer network resources and reduce overall costs.

In the reward function mentioned above, latency (in milliseconds) and available bandwidth (in bps) cannot be directly added because their units are different. To solve this problem, we need to normalize them, making them have the same order of magnitude. In this paper, we normalize latency and available bandwidth to a range of 0 to 1. We set the maximum and minimum latency in the simulation, and then calculate the normalized latency and available bandwidth. The normalized latency $norm\_delay_{ij}$ and normalized available bandwidth $norm\_aviband_{ij}$ can be calculated using the following formulas:

$$norm\_delay_{ij} = \frac{delay_{ij} - delay_{min}}{delay_{max} - delay_{min}} \tag{2}$$

In the above formula, $delay_{ij}$ represents the sum of the queuing delay of the data packet in node i's queue and the transmission delay between nodes j and i, $delay_{min}$ is the minimum link latency set in the simulation, and $delay_{max}$ is the maximum latency set in the simulation.

$$norm\_aviband_{ij} = \frac{aviband_{ij} - aviband_{ij}^{min}}{aviband_{ij}^{max} - aviband_{ij}^{min}} \tag{3}$$

In the above formula, $aviband_{ij}$ represents the available bandwidth on link {i, j} calculated by the node based on probe packets, $aviband_{ij}^{min}$ is the minimum link bandwidth, and $aviband_{ij}^{max}$ is the link capacity bandwidth.

Next, we can integrate the normalized delay and the normalized available bandwidth into the reward function:

$$R(s, a) = (\alpha * norm\_delay_{ij} - \beta * norm\_availband_{ij}) \tag{4}$$

To more accurately find the neighbor node with the best expected transmission performance, each node sends its expected delay cost generated when selecting neighbor nodes to the previous hop node after receiving the request packet from the previous hop. Since each node has multiple neighbor nodes, we employ the ε-greedy strategy to select actions. Specifically, node j selects an action randomly with probability ε and selects the action with the maximum Q value with probability $1 - \varepsilon$. Then, the value calculated according to the formula is returned to the previous hop node, as detailed in the following formula:

$$F(j) = (1 - \varepsilon) \cdot Q(j, a_{max}) + \varepsilon \times Q(j, a_{random}) \tag{5}$$

In the above formula, we set the probability of selecting the action with the maximum Q value to be 0.9, and the probability of selecting other actions to be 0.1.

Definition 6: Q-learning update rule. After each time a node selects a neighboring node to send a data packet, the Q value is updated based on the selected action, current state, new state, and reward. The update rule is as follows:

$$Q(i,a)_{new} = Q(i,a)_{old} + \lambda[R(\mathrm{i},a) + \gamma F(j) - Q(\mathrm{i},a)_{old}] \tag{6}$$

In the above formula, $Q(\mathrm{i},a)_{old}$ is the current state, a is the chosen action (i.e., the next hop node), j is the new state (i.e., the next-hop node), λ is the learning rate, with a range of 0 to 1, used to balance the old state and the new state, γ is the discount factor, with a range of 0 to 1; R(i, a) represents the reward obtained after node i takes action a; F(j) represents the feedback value sent by the node to the previous-hop node.

4.2 Construction of Video Transmission Path Algorithm Based on Q-Learning

Based on the CCDN network architecture and considering the dynamic characteristics of cloud content delivery networks, we propose a Q-learning-based Adaptive Video Routing (Q-AVR) algorithm. This algorithm aims to optimize the latency and bandwidth utilization during video transmission, improving overall network performance. First, the Q-table is initialized. Next, nodes exchange information through data packets and update the Q-values in the Q-table based on the feedback from the data packets, allowing the algorithm to adapt to network congestion changes. Then, through adaptive routing selection based on Q-learning, each node selects the next-hop node according to the information in its Q-table. Finally, upon receiving the data packets, each target node constructs a video transmission path in reverse based on the routing information in the packets. In this way, the target node transmits the video to the requesting node along this path, effectively reducing network transmission latency and improving video transmission performance.

4.2.1 Q Tables and Packets

In the Q-learning learning method, each node needs to store information related to its neighboring nodes. Therefore, we set up a table for each node, which is the Q-table. A node's Q-table consists of four parts: NextHop, Route, QValue, and Fvalue. NextHop represents the next-hop node in the Q-table; Route describes the path from the current node to the next-hop node; QValue indicates the Q-value obtained when the node selects the next-hop node; and Fvalue represents the feedback value received from the neighboring nodes.

To implement the Q-AVR algorithm, nodes need to exchange information using the following types of data packets:

Definition 7: Request Packet. The current node sends a request packet to the next hop node. The format is <node ID, Target node ID collection, path information, next hop ID, video ID, timestamp>. The node ID represents the position of the sending node, the Target node ID collection represents the node ID set of the target nodes, path information represents the sequence of nodes traversed by the data packet, next hop ID is the node selected by the current node based on the Q-table, video ID represents the identifier for the video requested by the node to the target node, and timestamp represents the time the node sends the data packet.

Definition 8: Acknowledgment Packet. After receiving the request packet, the next hop node replies with an acknowledgment packet, which contains link status information (e.g., delay). The format is <node ID, Request packet node ID, delay information, Fvalue, timestamp>. The node ID represents the position of the sending node, the Request packet node ID represents the position of the node sending the request packet, delay information represents the queuing time of the data packet in the node, and Fvalue represents the value obtained by the next hop node based on its own Q-table and Eq. 5.

Definition 9: Probe Packet. The current node sends a probe packet to the next hop node to estimate the available bandwidth on the link. The format is <node ID, next hop ID, Probe packet number, timestamp>. The node ID represents the position of the sending node, next hop ID is the node selected by the current node based on the Q-table, and Probe packet number represents the probe packet sequence number.

Definition 10: Probe Response Packet. When the next hop node receives the probe packet, it sends back these probe packets in the order and time intervals they were received. After receiving the returned probe packets, the node estimates the available bandwidth on the link by analyzing the time intervals of these probe packets. The format is <node ID, Probe Packet node ID, Probe packet number, timestamp>. The node ID represents the position of the sending node, and Probe Packet node ID represents the position of the node sending the probe packet.

Definition 11: Video Data Packet. The actual video data being transmitted, containing video content and other information related to video transmission, such as sequence numbers, timestamps, etc.

Through these packets, nodes can collect information about the link status, including delay, available bandwidth, and so on, and use this information to determine the optimal transmission path.

4.2.2 Q-AVR Algorithm

In constructing the transmission path for the Q-learning-based adaptive video routing (Q-AVR) algorithm, we first set the request node as the sender node and then send a request packet to the next hop node to find a path from the target node to the request node. Once all target nodes have established video transmission paths with the request node, video data transmission begins. The construction process of the Q-AVR algorithm can be divided into the following stages.

Stage One (Path Selection): Using the request node as the sender node, the node with the smallest Q value is selected according to the Q-table, and a request data packet and a probe packet are sent to it, calculating the available bandwidth and delay on the link. Upon receiving the request data packet, the neighboring nodes will add themselves to the path information. If a neighboring node is a target node, it constructs the video transmission path based on the path information in the received data packet. If it is not a target node, the neighboring node will return the probe packets in the order and time intervals they were received after receiving the probe packet. After receiving the request data packet, the neighboring node calculates the feedback value according to the Q-table and Formula (5) and returns the acknowledgment packet and probe response packet to the previous hop node. Next, the next hop node is selected according to the Q-table, and the request data packet and probe packet are sent continuously until all target nodes are reached.

Stage Two (Q Value Update): Nodes receive acknowledgment packets and probe response packets sent back by neighboring nodes. If the acknowledgment packet and probe response packet come from the target node, the reward value obtained from the target node will decrease since the next action of the current node is the target node. Therefore, we need to modify the reward function and calculate it according to the formula $R(s, a) = R(s, a) - C$, where C is a constant. Next, update the Q value according to Formula (6). If the data packet does not come from the target node, the node can estimate the available bandwidth on the link by analyzing the time intervals of the probe packets returned by neighboring nodes. Obtain the feedback value and queuing delay of the neighboring nodes according to the received acknowledgment packet and calculate the transmission time using the timestamp on the data packet. Then, calculate the reward function R(s, a) according to Formula (4) and update the corresponding Q value in the Q-table for that node according to Formula (6).

Stage Three (Data Transmission): When the requesting node finds a path to a certain target node, the target node establishes a video transmission path through reverse routing based on the path information in the data packet. The requesting node continuously explores the environment through this process, eventually finding paths to different target nodes. Then, the target nodes send data along these paths to the requesting node.

The pseudocode for the above process is shown in Algorithm 1.

Algorithm 1 Q-learning-based Adaptive Video Transmission Routing Algorithm

Stage 1: Path Selection

1. select node with the smallest Q value from the Q-table

2. For each neighbor node that receives the request data packet

3. Add itself to the path information

4. If neighbor node is the destination node

5. Build video transmission path based on path information

6. Else

7. Calculate feedback value based on Q-table and formula (4-5)

8. Return acknowledgement packet and probe response packet to the previous-hop node

9. Select the next-hop node and send data packets

Stage 2: Q-value Update

10. For each node that receives the acknowledgement packet and probe response packet:

11. If the data packet is from the destination node:

12. calculate the reward value using formula $R(s,a) = R(s,a) - C$

13. Update Q value according to formula (4-6)

14. Else

15. obtain $delay_{ij}$, transmission time, $availband_{ij}, Fvalue$

16. Calculate the reward function $R(s,a)$ according to formula (4-4)

17. Update Q value according to formula (4-6)

Stage 3: Data Transmission

18. When the request node finds the path to the destination node:

19. Destination node builds a reverse-routing video transmission path and sends video data

The Q-AVR algorithm selects the best path for video transmission based on Q-values while using probe packets and request packets to obtain real-time link information. During the node selection process, the Q-table will be continuously updated, allowing the algorithm to adapt to changes in network conditions and optimize video transmission latency and link bandwidth utilization.

5 Evaluation

We evaluate the performance of the proposed algorithm through simulation experiments. Performance metrics include end-to-end video transmission latency, PSNR (Peak Signal-to-Noise Ratio), and link bandwidth utilization. The Q-AVR method proposed in this paper is compared with the SLA-RPM [24] algorithm and the Widest-Shortest Path Algorithm (WSPA). The WSPA first focuses on the path with the highest available bandwidth when searching for the path from the request node to the target node, and then selects the one with the shortest delay among these paths. To some extent, this method balances two performance metrics, bandwidth and delay, but it's a static routing algorithm that cannot adapt to dynamic changes in network conditions. Simulation results show that the Q-AVR algorithm performs better compared to the SLA-RPM and WSPA algorithms.

5.1 Simulation Setup

OMNeT++ is a flexible, extensible, and modular C++ library for network simulations. In this paper, we implement simulation experiments through OMNeT++ to emulate the proposed Q-learning based video transmission algorithm. Here are the main modules and their functionalities: Node Model (responsible for node functions and Q-table management), Link Model (simulates delay and bandwidth between node connections), Video Transmission Application (generates and processes video data slices), Q-AVR algorithm implementation (executes the Q-learning process), and Statistics and Performance Evaluation Module (collects simulation statistical data and evaluates performance metrics). These modules are interconnected in OMNeT++ through a Network Description File (NED), and their respective functionalities are implemented using C++.

In the experiments, we deploy 50 nodes, where one is set as the request node, five as target nodes, and the rest as relay nodes. As can be seen from the Q-value update formula, a higher learning rate λ signifies weaker retention capacity of prior training results during the update process, while a larger discount factor γ implies greater attention to expected gains. Therefore, to better adapt to the dynamic environment, we set λ and γ to 0.9. Given that the reward function considers both delay and available bandwidth, in order to focus on lower end-to-end video transmission latency, we set the latency weight value to be relatively high. In the simulation experiment, the Q-value is initialized to 1 and the algorithm path is initialized to the shortest path from the target node to the request node. Initially, due to insufficient understanding of the environment and inaccurate information acquisition, it results in a video transmission path with high end-to-end latency cost. However, as the environment is explored more deeply, it builds a video transmission path with lower latency based on the accurate environment information obtained. The simulation experiment parameter settings are shown in Table 2.

Table 2. Experimental parameter setting

Parameter name	Parameter value	Parameter name	Parameter value
Node number	50个	Discount factor	0.9
Link delay	1–100 ms	Learning rate	0.9
Delay weight	0.7	Video fragment size	1M
Band weight	0.3	Total video data size	100G

5.2 Simulation Results

(1) End-to-end delay

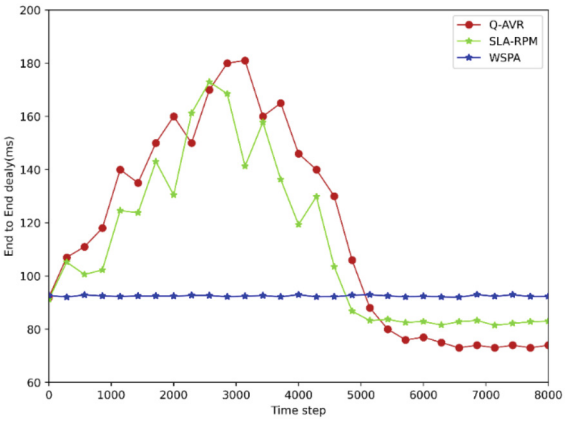

Fig. 3. Time-varying end-to-end delay

The end-to-end delay from the target node transmitting video to the request node is calculated, which includes queue delay and propagation delay. A lower end-to-end delay generally indicates better real-time performance and user experience, as shown in Fig. 3. At the beginning of the algorithm, due to the initial values of the Q-table and related parameters not being optimal, the end-to-end delay is higher. Over time, thanks to the exploration and exploitation strategies of the Q-learning algorithm, the end-to-end delay fluctuates for a period. This means that at certain moments, the algorithm attempts to explore new paths, causing a brief rise in delay; while at other moments, it exploits known better paths, thus reducing the delay. As the algorithm converges, with the Q-table and related parameters gradually approaching the optimal values, the end-to-end delay progressively stabilizes at a lower level. This signifies that the algorithm has found the optimal transmission path, resulting in a lower video transmission delay. With changes in network conditions, such as link load and available bandwidth, the curve will exhibit brief fluctuations. This is because the algorithm requires some time to adapt to the new network conditions. The end-to-end delay in SLA-RPM shows a trend of initially

increasing and then decreasing. This is because this method begins by exploring the environment to a certain extent, then calculates the optimal path based on the acquired environmental information. Although the SLA-RPM method converges the fastest, Q-AVR performs superiorly in terms of video transmission end-to-end delay. The end-to-end delay obtained using the WSPA shows a relatively stable curve. This is because this algorithm first chooses the path with the highest available bandwidth and then, from those paths, selects the one with the shortest delay. Thus, under relatively stable network conditions, this algorithm achieves a more stable end-to-end delay. However, as a static routing algorithm, it cannot adapt to the dynamic changes in network conditions. As can be seen from the above graph, with the increase in time, Q-AVR demonstrates a more significant advantage in reducing the end-to-end delay compared to the other two methods.

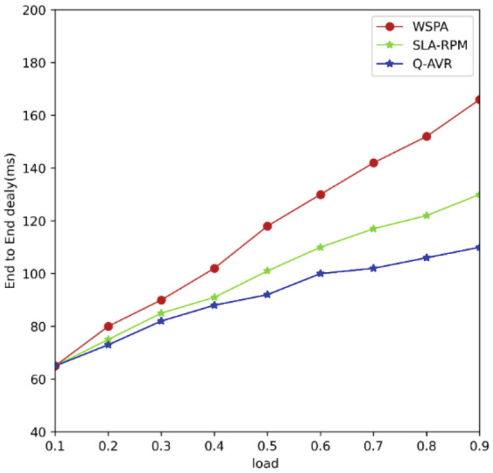

Fig. 4. Load-varying end-to-end delay

In order to better demonstrate the performance of the algorithm, we increase the load on the nodes and test the end-to-end video transmission delay of different algorithms, as shown in Fig. 4. With the increase in load, the end-to-end delays obtained by Q-AVR, SLA-RPM, and WSPA algorithms all show an upward trend. This is due to the fact that as the network load increases, the links and nodes within the network become more congested, leading to increased end-to-end latency. As the load gradually increases from 0.1 to 0.9, the Q-AVR algorithm performs better in reducing video transmission end-to-end latency compared to the other two methods. When the load increases to 0.9, the end-to-end video transmission delay of Q-AVR is 15% lower than that of SLA-RPM and 33% lower than that of WSPA. From Fig. 4, it can be seen that the end-to-end delay of the video transmission path obtained by the Q-AVR algorithm is the lowest under different loads.

(2) PSNR

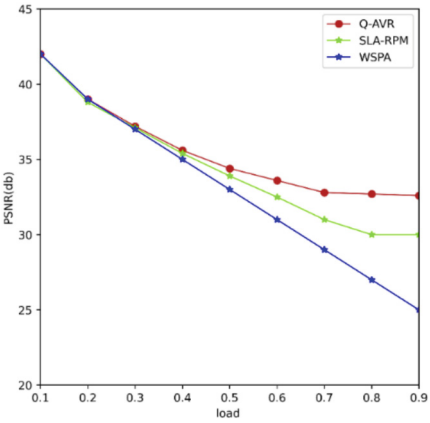

Fig. 5. Load-varying PSNR

We use PSNR (Peak Signal-to-Noise Ratio) to measure the video quality received by the nodes. Higher video quality implies a better user experience, as illustrated in Fig. 5. If the x-axis represents the load and the y-axis represents the PSNR, both the Q-AVR and SLA-RPM algorithms show a pattern where the PSNR first decreases as the load increases, and then gradually stabilizes. This is because when the network load is low, the Q-AVR and SLA-RPM algorithms can find better paths for the video, thereby achieving a higher PSNR. However, as the load increases, the available resources in the network become scarce, forcing Q-AVR and SLA-RPM algorithms to make trade-offs between latency and bandwidth, thus causing a drop in PSNR. Nevertheless, because the Q-AVR algorithm is an adaptive algorithm based on reinforcement learning that learns and optimizes strategies through continuous interaction with the environment, it can adjust its decisions based on changes in network conditions. Therefore, as the load further increases, the Q-AVR algorithm finds a more suitable path, which makes the PSNR gradually stabilize. The SLA-RPM algorithm also explores the environment to some extent and then calculates the optimal path based on the explored environment, but it's less effective than the Q-AVR algorithm. For the WSPA algorithm, at lower loads, it produces a higher PSNR since it only focuses on the path with the least weight. However, as the load increases and the resource distribution in the network becomes uneven, the WSPA algorithm cannot effectively utilize network resources, leading to a decrease in PSNR. When the load increases to 0.9, the Q-AVR algorithm can still achieve a video quality of 33 dB, demonstrating its superiority over the other two methods.

(3) Network bandwidth utilization

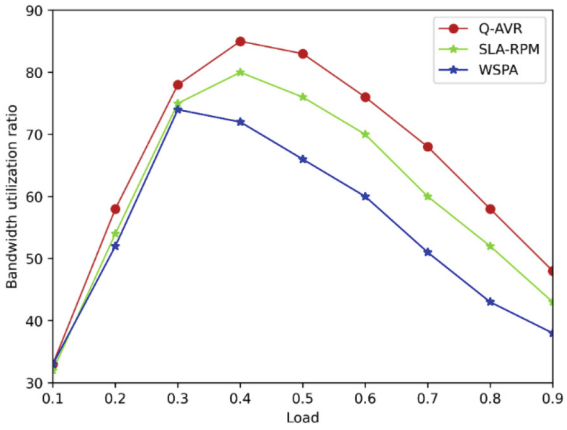

Fig. 6. Bandwidth utilization

As shown in Fig. 6, at lower loads, the bandwidth utilization of all three algorithms shows an increasing trend. This is due to the algorithms finding more optimal paths for data transmission. However, as the load increases, the Q-AVR algorithm can still maintain a high level of bandwidth utilization by dynamically adjusting paths to adapt to changes in network conditions. As can be seen from Fig. 6, compared to the SLA-RPM and WSPA algorithms, Q-AVR performs better in terms of bandwidth utilization.

6 Conclusions

In this paper, we consider the dynamic changes in network congestion in cloud content delivery networks. To reduce video transmission end-to-end latency and improve network bandwidth utilization, we propose a Q-learning-based Adaptive Video Routing (Q-AVR) algorithm for cloud content delivery networks. The algorithm balances the impact of latency and bandwidth utilization by introducing a reward function, in which the weight parameters α and β are used to adjust the relationship between latency and bandwidth. Through dynamic learning and adaptation to the network environment, the Q-AVR algorithm can find the optimal transmission path in situations such as load variations and network congestion, ensuring transmission quality and efficiency. Experimental results demonstrate that the Q-AVR algorithm exhibits higher adaptability in complex network environments, thereby achieving low latency and high bandwidth utilization in video transmission.

Acknowledgment. This research was funded by the National Key Research and Development Program of China (Grant No.2020YFF0305300), the Key Research and Development Program of Guangxi (Grant No. GuikeAB22080065), and the National Natural Science Foundation (Grant No. 61762029).

References

1. Cisco, U.: Cisco annual internet report (2018–2023) white paper, pp. 1–35 (2020). Accessed 10 Jan 2021
2. Maggs, B.M., Sitaraman, R.K.: Algorithmic nuggets in content delivery. ACM SIGCOMM Comput. Commun. Rev. **45**(3), 52–66 (2015)
3. Chen, F., et al.: Intra-cloud lightning: building CDNs in the cloud. In: 2012 Proceedings IEEE INFOCOM. IEEE (2012)
4. Salahuddin, M.A., et al.: A survey on content placement algorithms for cloud-based content delivery networks. IEEE Access **6**, 91–114 (2017)
5. Liu, P., et al.: Reducing video transmission cost of the cloud service provider with QoS-guaranteed. In: ICPCSEE 2022, Part I. CCIS, vol. 1628, pp. 387–402. Springer, Singapore (2022). https://doi.org/10.1007/978-981-19-5194-7_29
6. Liu, Y., et al.: Q-learning based content placement method for dynamic cloud content delivery networks. IEEE Access **7**, 66384–66394 (2019)
7. Yan, J., et al.: Q-learning-based vulnerability analysis of smart grid against sequential topology attacks. IEEE Trans. Inf. Forensics Secur. **12**(1), 200–210 (2016)
8. Kangasharju, J., Roberts, J., Ross, K.W.: Object replication strategies in content distribution networks. Comput. Commun. **25**(4), 376–383 (2002)
9. Xu, K., et al.: Joint replica server placement, content caching, and request load assignment in content delivery networks. IEEE Access **6**, 17968–17981 (2018)
10. Garmehi, M., Analoui, M.: A distributed mechanism for economic management of transmission infrastructure in hybrid CDN-P2P networks. Econ. Comput. Econ. Cybern. Stud. Res. **48**(3) (2014)
11. Sajithabanu, S., Balasundaram, S.R.: Direct push–pull or assisted push–pull? Toward optimal video content delivery using shared storage-based cloud CDN (SS-CCDN). J. Supercomput. **75**(4), 2193–2220 (2019)
12. Hu, H., et al.: Community based effective social video contents placement in cloud centric CDN network. In: 2014 IEEE International Conference on Multimedia and Expo (ICME). IEEE (2014)
13. Wang, Z., et al.: Propagation-based social-aware replication for social video contents. In: Proceedings of the 20th ACM International Conference on Multimedia (2012)
14. Zhang, Q., et al.: Harmony: dynamic heterogeneity-aware resource provisioning in the cloud. In: 2013 IEEE 33rd International Conference on Distributed Computing Systems. IEEE (2013)
15. Lu, D., et al.: Session-based cloud video delivery networks in mobile internet. J. Internet Technol. **18**(7), 1561–1571 (2017)
16. Shorfuzzaman, M., Graham, P., Eskicioglu, R.: Distributed placement of replicas in hierarchical data grids with user and system QoS constraints. In: 2011 International Conference on P2P, Parallel, Grid, Cloud and Internet Computing. IEEE (2011)
17. Gong, H., et al.: Distributed multicast tree construction in wireless sensor networks. IEEE Trans. Inf. Theory **63**(1), 280–296 (2016)
18. Haghighi, A.A., Heydari, S.S., Shahbazpanahi, S.: Dynamic QoS-aware resource assignment in cloud-based content-delivery networks. IEEE Access **6**, 2298–2309 (2017)
19. Borst, S., Gupta, V., Walid, A.: Distributed caching algorithms for content distribution networks. In: 2010 Proceedings IEEE INFOCOM. IEEE (2010)
20. Chen, J., et al.: Cost optimization for the coupled video delivery networks. IEEE Access **7**, 79136–79146 (2019)
21. Cherkasova, L.: Optimizing the reliable distribution of large files within CDNs. In: 10th IEEE Symposium on Computers and Communications (ISCC 2005). IEEE (2005)

22. Chandy, J.A.: A generalized replica placement strategy to optimize latency in a wide area distributed storage system. In: Proceedings of the 2008 International Workshop on Data-Aware Distributed Computing (2008)
23. Laoutaris, N., et al.: Inter-datacenter bulk transfers with NetStitcher. In: Proceedings of the ACM SIGCOMM 2011 Conference (2011)
24. Ma, Y.-W., et al.: A novel dynamic resource adjustment architecture for virtual tenant networks in SDN. J. Syst. Softw. **143**, 100–115 (2018)