Hindawi
Security and Communication Networks
Volume 2024, Article ID 9868731, 1 page
https://doi.org/10.1155/2024/9868731

Retraction

Retracted: Fault-Tolerant Secure Routing Based on Trust Evaluation Model in Data Center Networks

Security and Communication Networks

Received 8 January 2024; Accepted 8 January 2024; Published 9 January 2024

This article has been retracted by Hindawi, as publisher, following an investigation undertaken by the publisher [1]. This investigation has uncovered evidence of systematic manipulation of the publication and peer-review process. We cannot, therefore, vouch for the reliability or integrity of this article.

Please note that this notice is intended solely to alert readers that the peer-review process of this article has been compromised.

Wiley and Hindawi regret that the usual quality checks did not identify these issues before publication and have since put additional measures in place to safeguard research integrity.

We wish to credit our Research Integrity and Research Publishing teams and anonymous and named external researchers and research integrity experts for contributing to this investigation.

The corresponding author, as the representative of all authors, has been given the opportunity to register their agreement or disagreement to this retraction. We have kept a record of any response received.

References

[1] N. Liu, W. Fan, J. Fan, and H. Zheng, "Fault-Tolerant Secure Routing Based on Trust Evaluation Model in Data Center Networks," *Security and Communication Networks*, vol. 2022, Article ID 9339515, 13 pages, 2022.

Hindawi
Security and Communication Networks
Volume 2022, Article ID 9339515, 13 pages
https://doi.org/10.1155/2022/9339515

Research Article

Fault-Tolerant Secure Routing Based on Trust Evaluation Model in Data Center Networks

Ningning Liu,[1] Weibei Fan ,[2,3] Jianxi Fan,[4] and Hui Zheng[5]

[1]*Suzhou Vocational University, Suzhou, China*
[2]*School of Computer, Nanjing University of Posts and Telecommunications, Nanjing, China*
[3]*Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Jilin University, Changchun 130012, China*
[4]*School of Computer Science and Technology, Soochow University, Suzhou, China*
[5]*School of Computer, Swinburne University of Technology, Hawthorn, Australia*

Correspondence should be addressed to Weibei Fan; wbfan@njupt.edu.cn

Received 29 September 2021; Accepted 5 April 2022; Published 12 May 2022

Academic Editor: Xingsi Xue

With the rise of cloud computing technology, the whole information technology industry has undergone dramatic changes. Large-scale computing and the growing demand for computing power from large data have made it impossible for traditional data center network topology construction methods and network layer operation control methods with good stability to meet the requirements of technological development. The topology of a data center network is usually characterized as an undirected graph, and graph algorithms are used to implement communication patterns in complex network structures. However, the routing algorithms proposed earlier for graphs cannot be applied to all topological graphs. Based on the comprehensive interactive trust evaluation network security model, this study analyzes the advantages and disadvantages of relevant algorithms and proposes a fault-tolerant routing algorithm suitable for all networks. The algorithm can dynamically call different algorithms according to the number of current nodes and links. From the simulation results, it can be found that the algorithm not only improves the accuracy and fault tolerance but also effectively reduces the loss of time and memory.

1. Introduction

With the rapid development and progress of cloud computing and its related technologies, the development of the computer industry shows a high-speed growth trend and also promotes the informatization process of all walks of life. At present, in the field of computer security research, the main security problems faced by the data center network are network intrusion and system failures [1]. Although security and fault tolerance are two different concepts, they are closely related and interdependent. The primary problem in the research of security and fault-tolerant technology is to ensure the availability of network system [2].

Digitization and green are important issues facing the future for all countries in the world. A data center network (DCN) is a network that provides internal interconnection, data center, and external interconnection of the data center. It is an important infrastructure of modern network and cloud computing. As the infrastructure of data center, DCN is facing unprecedented challenges in realizing the performance required by users under the current background due to the limitations of its topology and transmission capacity [3]. Fault-tolerant secure routing means that when there are a certain number of failed nodes in the network system, an efficient transmission route can still be found using the concept of node security level [4, 5]. Therefore, only by designing a better fault-tolerant routing strategy we can effectively improve the performance of the whole network. Based on different network security models, the optimal path information is recorded in the system as much as possible, so that the system can quickly provide safer and more effective fault-tolerant routing in case of failures [6]. We design a universal fault-tolerant routing algorithm for data center networks and perform a detailed performance

simulation of the proposed algorithm [7]. To ensure reliable communication of data, the security performance of data center network must be effectively improved. Optimizing a fault-tolerant routing algorithm is an effective method at present.

Network security and fault-tolerant routing algorithm are an important and popular research direction in the field of computer network [8–11]. Many scholars have proposed an algorithm design or optimization for a specific network [12, 13]. Firstly, for the internal attacks on the network, researchers have proposed many trust-based methods to enhance the security of routing protocols. Cohen et al. [14] mainly use fuzzy logic to synthesize trust and construct trust security model, but this model often leads to inaccurate results due to the loss of information, and it ignores the key factor of indirect trust in the calculation process. Vamsi et al. [15] consider the impact of direct trust and indirect trust and use the beta model to determine the weight, but ignore the importance of historical trust value, so the results are not accurate. Secondly, Luo et al. proposed a secure routing protocol improved ACO-based security routing protocol (IASR) [16]. The algorithm takes the historical and current trust values into account. The calculated successful interaction rate improves the accuracy of the calculation results of direct trust value to a certain extent, but it cannot quickly identify the fake and violent attacks of malicious nodes. How to combine security and fault tolerance perfectly is a new problem. For the research of fault-tolerant routing, many scholars have also put forward corresponding solutions. In the beginning, an equivalent cost multipath routing (ECMP) [17] was widely used. However, with the emergence of software-defined network, the control surface and data surface of network equipment were separated, which realized the flexible statistical control of network traffic and provided a good platform for the proposal of subsequent routing algorithms.

Chu et al. [18] proposed a software-defined hybrid routing (SHR) mechanism for data center networks. SHR adopts a compromise method to classify data flows and adopt different routing algorithms for different types of traffic. Compared with the traditional ECMP algorithm, this mechanism reduces the data flow discarding rate and improves the network throughput, but does not consider the performance of link utilization, which brings some defects to the mechanism. The multipath routing on link real-time status and flow characteristic (MLF) algorithm proposed by Peng et al. [19] uses the link remaining bandwidth and hash operation to select the path for large streams and selects the path with the largest remaining bandwidth for small streams. This algorithm can make up for the shortcomings of other algorithms to a certain extent. However, if the routing is only based on the residual bandwidth of the link for the delay-sensitive small stream, the selected time delay is likely to be large, resulting in the unsatisfactory time delay of small stream transmission.

Lei et al. [20] proposed a multipath routing algorithm based on branch and bound in software-defined data center networks (MPB-AA), which gives priority to link delay and residual bandwidth according to the characteristics of large and small streams and uses branch-and-bound method to find paths; compared with MLF, it has shorter end-to-end delay and higher throughput. Finally, the LABERIO algorithm proposed by Long et al. [21] takes the bandwidth variance used by the whole network as the load balance index. When it is found that this index exceeds the threshold, it starts the rerouting mechanism to schedule the data flow on the link with the highest utilization to the available path with the largest bottleneck bandwidth. Although the LABERIO algorithm realizes the timely processing of congested links, when the network fluctuates greatly, the load balance index will be too large, which will continue to trigger the LABERIO algorithm, resulting in the waste of network resources. There is little research on fault-tolerant routing mechanism in the data center network, which mainly uses the multiple paths in a new specific topology to design the corresponding fault-tolerant mechanism. On the basis of load balancing, Kanagavelu et al. [22] collected link failure information using NOX components, and then, the routing module in the centralized controller made full use of the currently destroyed paths to find an alternative path for each affected data stream to solve the problem of fault tolerance. Fan et al. [23] proposed a fault-tolerant routing algorithm with load balancing for LeTQ networks. The proposed algorithm uses the node shrinkage method to evaluate the priority of nodes. The sending node adaptively adjusts the probability of forwarding packets to the neighbor node according to the priority of the neighbor node and the state of the network.

The main results of this study are as follows:

(1) We analyzed various previous fault-tolerant secure routing algorithms based on the trust evaluation model in DCNs and implemented the Floyd algorithm and double algorithm based on double_stack. Based on this strategy, this study writes a program to encapsulate the two and uses the ratio of edge ratio to node to automatically select an algorithm.

(2) We give the construction of the network topology and the display of the topological shape and read TXT through the object-oriented high-level programming language Python.

(3) We compare the performance indicators of the proposed algorithm under different network topologies, such as execution efficiency, memory footprint, and pathfinding accuracy, such as the proposed bus network, fat tree, DCell, FiConn, and BCube.

(4) We give a detailed analysis of the execution results of the algorithm. By finding the appropriate threshold, the proposed algorithm can automatically adapt to different network environments.

The rest of this study is organized as follows. Section 2 mainly introduces the basic knowledge of data center network security model and fault-tolerant routing algorithm of data center network in cloud environment. Section 3 introduces the trust evaluation security model based on DCN. Section 4 gives the design of fault-tolerant routing algorithm

in the ideal environment, including how to globally map the network topology to the undirected graph and a variety of graph routing algorithms and how to dynamically call each routing algorithm according to the existing situation of the graph to obtain the highest efficiency. The final section mainly contains the summary of the full text and the outlook for the future.

2. Preliminaries

In this section, some necessary definitions and concepts are given. The section presents the mapping of the software-defined network (SDN) and data center networks (DCNs).

A data center network can usually be represented by a simple graph $G = (V(G), E(G))$, abbreviated as G. Among them, the node set $V(G)$ is a nonempty finite set of nodes in the graph, which is also represented as the set of all servers in the data center network. The edge set $E(G)$ is the set of edges in the graph, which is also represented as the set of links between the connecting servers in the data center network. Switches are seen as transparent devices.

DCN [24, 25] refers to an infrastructure of data center network, which uses high-speed links to connect with switches and servers. Through unified planning and arrangement of resources, it can make full use of centralized large-scale resources to provide reliable and safe services for decentralized users [26].

A graph is usually represented by an ordered binary group $G = (V(G), E(G))$, where the elements in $V(G)$ are represented as graph nodes, and the elements in $E(G)$ are represented as graphs, while $E(G) \subseteq V(G) \times V(G)$. A graph G is called a directed graph when $V(G) \times V(G)$ is an ordered set of node pairs. When $V(G) \times V(G)$ is an unordered set of node pairs, the graph G is called an undirected graph. Let $W = (v_1, v_2, \ldots, v_n)$ be a nonempty finite alternating sequence of nodes in G, and for any $1 \le j < n$, v_j and v_{j+1} are adjacent, and then, we call W a pathway from v_1 to v_n, where v_1 and v_n are the start and end nodes of the pathway, respectively. The number of edges traversed in the path is called the length of W. If there are no duplicate nodes in W, then the path is called a path or path from u_1 to u_n. At the same time, we also use $(u_1, u_2, \ldots u_i, Q, u_j, \ldots u_n)$ to represent the path $P = (u_1, u_2, \ldots u_n)$, where Q represents the path $(u_{i+1}, u_{i+2}, \ldots, u_{j-1})$. Let $P - 1$ denote the path $(u_n, u_{n-1}, \ldots, u_2, u_1)$, the reverse order of nodes in the path P. Meanwhile, we use $\mathrm{Path}(P, u_i, u_j)$ to denote the path from u_i to u_j along $P = (u_1, u_2, \ldots u_n)(u_i, u_{i+1}, \ldots, u_{j-1}, u_j)$, where $1 \le i \le j \le n$. Simply, we denote a path with n nodes by P_n.

Let P and Q be two distinct paths between nodes u and v in G. If $V(P) \cap V(Q) = \{u, v\}$, then P and Q are called vertex disjoint paths, or disjoint paths for short or no traffic. In addition, if P does not contain faulty nodes, then we call P to be a fault-free path. The minimum value of the lengths of all paths between any two nodes u and v in G is called the distance between them, denoted as $\mathrm{dist}(G, u, v)$, abbreviated as $\mathrm{dist}(u, v)$. The diameter of G is $D(G) = \max\{\mathrm{dist}(G, u, v) | u, v \in V(G) \text{ and } u, v\}$. It can be seen that the diameter is the maximum value of the distance between any two distinct nodes in the graph.

Therefore, the diameter of the graph can be used to measure the worst case of network communication performance.

3. Comprehensive Interactive Trust Security Model Based on Selection and Comparison

To improve the security of DCN, we adopt an improved trust evaluation model and integrate the model into the dynamic fault-tolerant routing algorithm. The trust evaluation model infers the future behavior of network nodes through the statistics of the original trust value of network nodes. The forwarding node adopts an appropriate trust model to calculate the direct and indirect trust values, and the comprehensive interactive trust value can be obtained by weighting the two.

We adopt an improved trust evaluation model, integrate the model into a two-hop real-time reliable routing protocol, and study a trusted and secure routing protocol for data center network nodes. First of all, the direct trust value interaction update method can not only prevent the brute force attacks of malicious nodes but also identify the camouflage behaviors of malicious nodes. Secondly, the recommendation trust reliability calculation method based on recommended nodes excludes unreliable recommended nodes and prevents malicious nodes from slandering other nodes. Ultimately, network security is improved.

3.1. Direct Trust. Each node in DCN is assigned a component to monitor the behavior of other nodes. When the forwarding node i sends a packet to the next-hop neighbor node $i + 1$, the monitoring component starts to monitor the wireless channel within the node trust value periodic update time Δt. The purpose is to receive a passive acknowledgment message and monitor whether the next-hop node forwards the packet it sends. If the passive acknowledgment signal is not received within the time interval, the packet will be retransmitted until the retransmission times reach the set upper limit of transmission times (the upper limit of retransmission times is set to 5).

To ensure the freshness and time correlation of trust information, forwarding node i needs to constantly update the trust value with one-hop neighbor node $i + 1$. For this purpose, the trust value is updated by periodic update. The update cycle is Δt. When $t = t_0$, the trust mechanism of DCN starts to work. The number of successful and failed interactions at time $t_n = t_{n-1} + \Delta t\,(n = 1, 2, 3, \ldots)$ is updated as follows:

$$\begin{cases} a_{i,i+1}^{n} = a_{i,i+1}^{n-1} + (1 + \lambda) \times \varphi_{i,i+1} \\ b_{i,i+1}^{n} = b_{i,i+1}^{n-1} + (1 + \lambda) \times \delta_{i,i+1} \end{cases}, \tag{1}$$

where $a_{i,i+1}^{n}$ and $b_{i,i+1}^{n}$ represent the number of successful and failed interactions between node i and node $i + 1$ accumulated from t_0 to t_n, respectively. When time is t_0, $a_{i,i+1}^{0} = 0$ and $b_{i,i+1}^{0} = 0$. $\varphi_{i,i+1}$ and $\delta_{i,i+1}$ are the number of successful

and failed interactions between the two nodes in Δt. λ is the dynamic update weight factor [27].

The calculation formula of dynamic update weight factor λ is as follows:

$$\lambda = \begin{cases} 0, & \text{when } |z| < \mu \text{ or } n = 1 \\ z, & \text{when } |z| \geq \mu \end{cases}, \quad Z = \frac{a_{i,i+1}^{n-1}}{a_{i,i+1}^{n-1} + b_{i,i+1}^{n-1}} - \frac{\varphi_{i,i+1}}{\varphi_{i,i+1} + \delta_{i,i+1}}, \tag{2}$$

where z represents the difference between the historical successful interaction rate accumulated from t_0 to t_{n-1} and the recent successful interaction rate. $\mu \in [0, 0.5]$ is the joint measure to measure the closeness between the history and recent behavior of node $i + 1$ [28].

If $|z| < \mu$ or $n = 1$, it means that the behavior of node $i + 1$ is consistent and the reliability is high. λ takes 0. The number of successful exchanges in (1) is the number of recent successful exchanges directly accumulated by historical data. The update method of the number of unsuccessful exchanges is the same.

If $|z| > \mu$, which indicates that the behavior of node $i + 1$ is quite different, that is, the behavior is abnormal, it should be discussed in the following two cases.

Case 1. $(z < 0)$.

It indicates that the historical success rate of node $i + 1$ is lower than the recent success rate. When $\lambda < 0$, the number of successful and failed interactions used to measure direct trust rises slightly in (1). The resultant rate of successful interaction increases slightly relative to the rate of successful interaction calculated by the weightless factor λ, which can prevent malicious nodes from increasing the rate of successful interaction quickly by deceiving or disguising.

Case 2. $(z \geq 0)$.

It indicates that the historical successful interaction rate of node $i + 1$ is higher than the recent successful interaction rate, and the probability of node malignant behavior increases. When $\lambda \geq 0$, the number of successful and failed interactions used to measure direct trust in (1) increases dramatically, resulting in a greater decrease in the calculated successful interaction rate than that calculated with no weight factor λ, which allows malicious nodes to be quickly identified.

When $t = t_0$, the direct trust $\mathrm{DT}_{i,i+1}^n$ of node i to node $i + 1$ is defined as a function of the historical success interaction rate as shown in (4).

$$\mathrm{DT}_{i,i+1}^n = \begin{cases} \dfrac{a_{i,j+1}^n}{a_{i,j+1}^n + b_{i,j+1}^n} \dfrac{1}{\sqrt{b_{i,j+1}^n}} & \text{when } n \neq 0, \\ 0.5 & \text{when } n = 0, \end{cases} \tag{3}$$

where $1/\sqrt{b_{i,j+1}^n}$ is the penalty factor, and the increase in the number of noncooperation will lead to the decrease in $\mathrm{DT}_{i,i+1}^n$, which can reduce the attack risk of malicious nodes that have cheated a higher trust value.

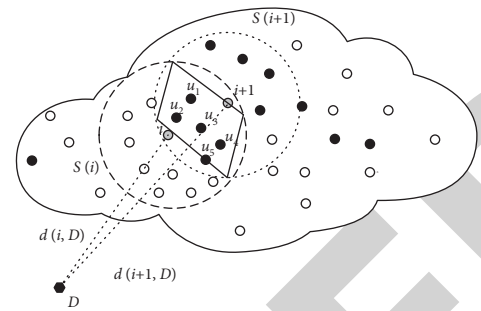

Figure 1: Schematic diagram of recommended node set.

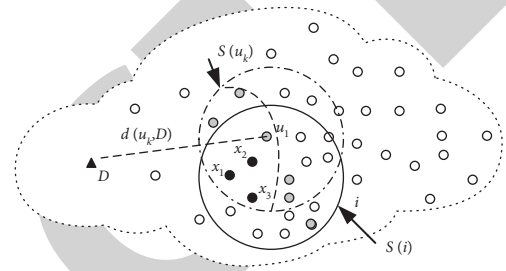

Figure 2: Schematic diagram of reliability evaluation node set.

3.2. Indirect Trust. Suppose $S(i)$ is the set of neighbor nodes for node i, $S(i + 1)$ is the set of neighbor nodes for node $i + 1$, and the set of common neighbor nodes [27] for nodes i and $i + 1$ is as follows:

$$C_i = \{u_k | u_k \in [S(i) \cap S(i + 1)]\} \; k = 1, 2, 3, \ldots. \tag{4}$$

Because of the randomness of black hole attacks initiated by malicious nodes, it is not sufficient to select a direct trust value directly. Node i needs to request a recommended trust value from the recommended node in C_i. The recommended node u_k will return the direct trust value of node $i + 1$ as the recommended information to node i. The recommended set of nodes is represented as follows.

$$R_1 = \{u_k | [d(u_k, D) - d(i + 1, D) > 0] \cap [d(i, D) - d(u_k, D) > 0] \; u_k \in C_1\}, \tag{5}$$

where $d(i, i + 1)$ represents the distance between two nodes and $[d(u_k, D) - d(d + 1, D) > 0] \cap [d(i, D) - d(u_k, D) > 0]$ represents the node between the radius arcs formed by node i and node $i + 1$ in the direction of target node D. As shown in Figure 1, black solid nodes u_1, u_2, and u_3 are located in the recommended node area. They represent the recommended nodes.

To ensure the reliability of the recommended node u_k, it is also necessary to calculate the trust deviation between node i and node u_k, to obtain the reliability of the recommended node u_k. The trust deviation between node i and node u_k depends on the common neighbor nodes of the two nodes.

Then, the common neighbor node set of node i and node u_k is expressed as follows:

$$C_2 = \left\{x_p | x_p \in \left[S(u_k) \cap S(i)\right]\right\} \quad p = 1, 2, 3, \ldots \tag{6}$$

The reliability evaluation node set of node u_k is as follows:

$$R_2 = \left\{x_p | d(u_k, D) - d\left(x_p, D\right) > 0, \quad x_p \in C_2\right\}, \tag{7}$$

where $d(u_k, D) - d(x, D) > 0$ represents the node in the radius arc formed by node u_k in the direction of target node D. The reliability evaluation node set of node i and node u_1 is shown in Figure 2. Black solid nodes x_1, x_2, and x_3 are the reliability evaluation node set in the reliability evaluation area of u_1.

The reliability formula of the recommended node u_k at time is defined as follows:

$$\mathrm{Re}_{i,u_k} = \begin{cases} 1 - \dfrac{\ln \eta}{\ln\left(De_{i,u_k}\right)}, & \text{when } \mathrm{De}_{i,u_k} < \eta, \\ 0, & \text{when } \mathrm{De}_{i,u_k} \geq \eta, \end{cases} \tag{8}$$

where η represents the deviation tolerance of node trust, and the value range is [0, 1]. De_{i,u_k} represents the trust deviation of node i to u_k, and its value can be expressed as follows:

$$\mathrm{De}_{i,u_k} = \frac{\sqrt{\sum_{x_p \in R_2}\left(\mathrm{DT}^n_{i,x_p} - \mathrm{DT}^n_{u_k,x_p}\right)^2}}{\left|R_2\right|}. \tag{9}$$

The numerator represents the square root of the difference between the direct trust values of node i and the recommended node u_k to all reliability evaluation nodes. If $\mathrm{De}_{i,u_k} > \eta$, it means that there is a large deviation between the two nodes in the direct trust of the reliability evaluation node, so the reliability Re_{i,u_k} of u_k is 0. If $\mathrm{De}_{i,u_k} < \eta$, it means that the deviation between the two nodes in the direct trust of the reliability evaluation node is small, so the recommendation of u_k has a certain reliability. However, with the increase in trust deviation De_{i,u_k}, the value of reliability Re_{i,u_k} will decrease rapidly.

We assume l recommendation nodes u_k, and then, l recommendation trust values $\mathrm{DT}^n_{u_1,i+1}, \mathrm{DT}^n_{u_2,i+1}, \mathrm{DT}^n_{u_3,i+1}, \ldots, \mathrm{DT}^n_{u_l,i+1}$ can be obtained. The weight of indirect trust value based on reliability is calculated:

$$\varepsilon_k = \frac{\mathrm{Re}_{i,u_k}}{\sum_{k=1}^{l} \mathrm{Re}_{i,u_k}}. \tag{10}$$

When the final $t = t_n$, the indirect trust value calculation formula of node i to point $i + 1$ is as follows:

$$Rt^n_{i,i+1} = \sum_{k=1}^{l}\left(\varepsilon_k \times \mathrm{DT}^n_{u_k,i+1}\right). \tag{11}$$

3.3. Integrated Interactive Trust Computing Model. When evaluating the trust relationship between nodes, we should consider not only the direct trust value but also the indirect trust value. We mainly take the weighted sum of direct trust value and indirect trust value as the comprehensive trust value. It is defined that the comprehensive interactive trust value of node i to node $i + 1$ is the weighted sum of direct trust value and indirect trust value, as shown in formula:

$$\mathrm{AT}^n_{i,i+1} = \xi\, \mathrm{DT}^n_{i,i+1} + (1 - \xi)\mathrm{RT}^n_{i,i+1}, \tag{12}$$

where $0 < \xi < 1$ represents the weighting factor. To avoid the intervention of human factors, the adaptive method is selected to calculate ξ. The calculation formula is as follows:

$$\xi = \frac{G\left(\mathrm{TN}_{i,i+1}\right)}{G\left(\mathrm{TN}_{i,i+1}\right) + G(l)}, \tag{13}$$

where $\mathrm{TN}_{i,i+1} = a^n_{i,i+1} + b^n_{i,i+1}$ represents the total number of successful and failed interactions within Δt time. l represents the number of trusted recommended nodes. $G(y) = y/1 + y$ is defined as a function of variable y. With the increase in variable y, $G(y)$ decreases slowly. The size of the weighting factor ξ depends on the total number of interactions when calculating the direct trust value and the number of recommendations when calculating the indirect trust value. If the former is greater than the latter, the former has a significant weight and the latter has a significant weight.

4. Dynamic Fault-Tolerant Routing Algorithm

In this section, we design an efficient dynamic fault-tolerant routing algorithm for the data center network. The shortest path algorithm of graph also includes the Bellman–Ford algorithm [29, 30] and its improved algorithm (SPFA), and Dijkstra algorithm [31, 32]. Table 1 compares the differences in various algorithms and lists the reasons for choosing DFS and Floyd algorithms.

For dense graphs, the Dijkstra algorithm cannot handle negative weight edges. Although negative weight edges are outside the scope of this study, they can represent the incentives of links in practice. Using Dijkstra is not conducive to subsequent expansion and program generality. For routing between any two nodes, n times Dijkstra is less efficient than Floyd. For sparse graphs, the time complexity and space complexity of Bellman–Ford, SPFA, and shortest path DFS are basically the same, but Bellman–Ford and SPFA can only judge whether there is a negative weight cycle.

The core pseudocode of the dynamic fault-tolerant routing algorithm designed in this study is as follows:

```
(1) Begin
(2) Count the number of now_nodes and now_edges;
(3) α = 1.4;
(4) if now_edges/now_nodes > α
(5) Call Floyd algorithm;
(6) Else
(7) Call double_stack_DFS;
(8) End
```

Table 1: Comparison of shortest path algorithms.

	Floyd	Dijkstra	Bellman–Ford	SPFA	Double-stack_DFS
Spatial complexity	$O(N^2)$	$O(E)$	$O(E)$	$O(E)$	$O(N)$
Time complexity	$O(N^3)$	$O(N+E)\log N$	$O(NE)$	$O(NE)$	$O(N!)$
Application	Dense graph Nodes are closely related	Dense graph Nodes are closely related	Sparse graph Edges are closely related	Sparse graph Edges are closely related	Sparse graph Edges are closely related
Usage	The optimal path of any two nodes can be obtained by executing once	Once executed, the optimal path from the first node to any node can be obtained	Once executed, the optimal path from the first node to any node can be obtained	Once executed, the optimal path from the first node to any node can be obtained	Once executed, all paths are specified between two nodes
Spatial complexity	$O(N^2)$	$O(E)$	$O(E)$	$O(E)$	$O(N)$

When there are negative weight cycles, the shortest path DFS can run directly, and the DFS can record all feasible paths. This way, if the link fails frequently, and the failed nodes are random, DFS has the shortest path priority and can find the second path in $O(N)$ time. Therefore, the Floyd and DFS algorithms are chosen here, which are dynamically invoked by the ratio of now_nodes and now_edges. It should be noted that the implementation of the following two programs can calculate the case of negative weights, but we mainly focus on the optimal fault-tolerant routing with the number of hops as the path value.

The value of α is 1.4. When the edge-node ratio is greater than the threshold α, the topology graph is identified as a dense graph and the Floyd algorithm is called. When the ratio is less than the threshold α, the graph is identified as a sparse graph and the double-stack_DFS is called. $\alpha = 1.4$ is the best value selected after a variety of topology simulation, and its performance will be described in the experiment in the next chapter.

The Floyd algorithm [33] can be used without negative weight edge loop, which is consistent with the data center network in this study. The algorithm can not only calculate the shortest path between any two nodes through a weighted matrix but also record the shortest path between two nodes by introducing a successor node matrix. The algorithm flow chart is shown in Figure 3, and its specific implementation ideas are as follows:

(1) Read in the adjacency matrix. If there is a connection between two nodes, set it to 1, and if there is no connection, set it to infinity inf.

(2) For any two distinct nodes u and v, check whether there is a third node w, so that the path value passing through w is shorter.

The specific method is as follows:

(1) Define the adjacency matrix distance according to the above design scheme. If there is a reachable path from nodes u to v, let distance[u][v] = 1; otherwise, set distance[u][v] = inf.

(2) Define another matrix route with the same size, record the information of the inserted point, and initialize route[u][v] = v.

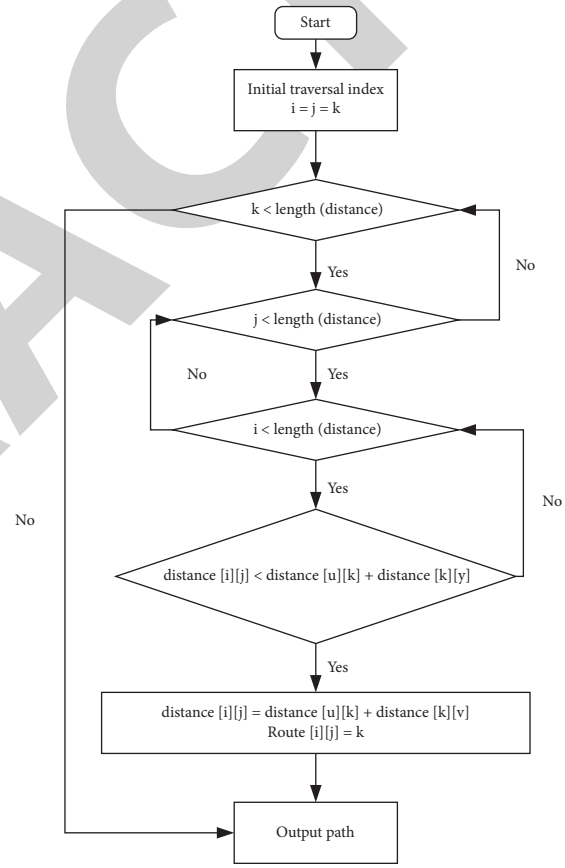

Figure 3: Floyd algorithm flow chart.

(3) Insert each node into the diagram in turn, and compare the path value after inserting the new node with the original path value, that is, distance [u][v] = min (distance[u][v], distance[u][k]+distance[k][v]). If distance [u][v] becomes smaller, let route[u][v] = k.

(4) After traversing all nodes, generate the shortest path from any source node i to destination node j through the route matrix and output.

The Floyd algorithm belongs to dynamic programming, with time complexity of $O(n^3)$ and space complexity of

$O(n^2)$. The Floyd algorithm performs best in dense graphs. The efficiency of the algorithm is higher than that of the Dijkstra algorithm or SPFA algorithm. When the topology has not changed, the shortest path between any two nodes can be obtained only by calculation once. The code implementation is simple, compact, and robust. However, its performance on sparse graph is not very ideal, and the algorithm itself has no memory function except the optimal path. When there is an error in the optimal path, the adjacency matrix needs to be modified and calculated again.

To supplement the performance of Floyd's fault-tolerant routing algorithm on sparse graph, this study introduces the second algorithm, the depth-first algorithm based on double stack, which dynamically calls the two algorithms by comparing the ratio of edge to node in the existing topology.

To avoid implementing depth-first search using recursion, we use two stacks to implement node expansion and path recording, where the main_stack stores a node for recording the path, and the edge_stack stores a list of adjacent nodes for the current element. Taking topology diagram 4 as an example, the best path is calculated from node 3 to node 6. The specific ideas are as follows:

(1) Set two stacks, main_stack and side_stack. Always keep the stack height consistent.

(2) Put the source node into the main_stack and the list of adjacent nodes of the top element of the main_stack into the side_stack.

(3) Select a node in the top element of the side_stack and move it into the main_stack, and add the list of adjacent nodes of the top element of the new main_stack at the corresponding height of the side_stack.

(4) When the top of the side_stack is empty, check whether the top element of the main_stack of the main_stack is the destination node. If not, an element will pop up on both the main stack and the side_stack.

(5) Repeat steps (3) and (4) until the top of the main_stack is the target node. Record the main_stack sequence to get an available path.

(6) Sort all paths by path length to obtain the optimal path.

The depth-first algorithm based on double stack needs the topology to adopt the form of an adjacency table to facilitate the query of adjacent nodes. In this case, the time complexity of the algorithm is O(N+E), and the space complexity is O(N). The algorithm is very suitable for sparse graph. Its advantage is that it has memory ability for all paths, can quickly find a standby scheme in the case of a node or link error, and can make a rough judgment on the fault tolerance of a topology. Its disadvantage is that the worst time complexity of depth first is. When there is no macro-understanding of the whole topology, the blind use of the algorithm may not meet the time limit and be inefficient.

Table 1 shows the different application scenarios of the shortest path algorithm. Since we are targeting a generalized data center network topology, it is clear that a single algorithm cannot adapt to all situations. This requires dynamically invoking various algorithms based on edge and node conditions.

5. Experimental Results

5.1. Experimental Environment and Content. We use a plain text file to define the network topology of data center, use Python 3.9 to implement the dynamic fault-tolerant routing algorithm, and complete the functions of topology storage, topology display, simulating network fault, detecting connectivity performance, dynamic fault-tolerant routing, and so on. The experimental environment of this study is 2.0 GHz 4-Core 10th-Generation Intel Core i5 Processor with 16 GB 3733 MHz LPDDR4x memory. The test environment parameters are shown in Table 2.

Meanwhile, to verify the correctness of the algorithm and quantitatively analyze its related performance, we use plain text files to define multiple topologies, including bus topology, 4-dimensional fat tree topology, and DCell topology. These topologies are manually input according to the description and definition of previous papers, and the third-party library NetworkX [34] is used to display the topology diagram to ensure that the topology diagram is consistent with the blueprint.

Finally, to better analyze the dynamic routing fault-tolerant algorithm, this study sets up several groups of comparative experiments:

(1) Taking the Floyd algorithm as the benchmark algorithm, the accuracy of dynamic routing fault-tolerant algorithm is tested.

(2) Comparing the time complexity of Floyd algorithm, depth-first algorithm, and different α value dynamic routing algorithms in the ratio of topology with different edge node ratios.

(3) Comparing the spatial complexity of Floyd algorithm, depth-first algorithm, and different value dynamic routing algorithms in topological graphs with different edge node ratios.

(4) Comparing the fault tolerance performance of each topology using the dynamic fault-tolerant routing algorithm.

For the experimental parameter selection of the algorithm, after reading any fat tree or DCell topology, we simulate the routing fault by randomly closing the node or link, reduce the link-node ratio by equal difference, and count the time complexity and space complexity of each algorithm through the test code, where α is set to 1.3, 1.4, and 1.5.

5.2. Algorithm Validity Test. For the effectiveness of the algorithm, we mainly test two aspects: the accuracy and fault tolerance of the algorithm. The accuracy of the algorithm reflects whether the algorithm is correct or not, that is, whether the algorithm can find the shortest routing path when the topology is fixed, and the path does not include any faulty nodes or links. The fault tolerance of the algorithm reflects the reliability of the algorithm, that is, whether the

Table 2: Test environment parameter table.

Host environment	
Operating system	MacOS Catalina 10.15.7
Processor	2.0 GHz 4-Core 10th-Ggeneration Intel Core i5 Processor
Memory	16 GB 3733 MHz LPDDR4x

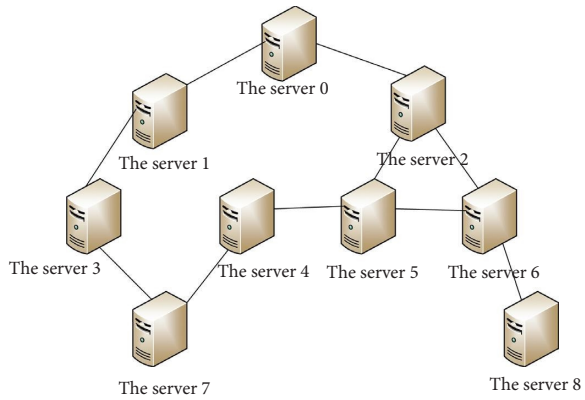

Figure 4: Sample topology.

algorithm can still work normally and find a feasible path in the case of as many faults as possible (Figure 4).

The Floyd algorithm is used as the benchmark algorithm, a topology map is defined by text file, and the dynamic routing algorithm and benchmark algorithm are used for testing. A random function is defined to randomly send multiple source nodes and target nodes into the two algorithms, and then, the obtained paths are compared. The accuracy of the algorithm is 100% after multiple tests at any closed node or link. The screenshot of the correctness test command line is shown in Figure 5.

In the actual production environment, nodes or links in the network topology will fail. The original static routing algorithm will set many paths in advance and quickly switch to the next preset path in case of error in the current path. The fault tolerance of this algorithm is limited. When the damaged node or link exceeds a certain value, the routing may make an error. A dynamic fault-tolerant routing algorithm adopts the way of dynamic path acquisition, and its fault-tolerant performance actually depends on the fault-tolerant ability of the network topology.

In the most ideal case, the whole topology is in the form of full connection. At this time, the fault tolerance performance of the whole network is the strongest. Except for the source node and the destination node itself and the link before them, all other node or link failures will not affect the data transmission. Therefore, we define the fault tolerance of dynamic fault-tolerant routing algorithm as ftv (fault_tolerant_value), and its calculation formula is shown in (14):

$$\text{ftv} = \frac{1/n \sum_0^n n\_\text{break}_i}{(\text{node} - 2)}, \tag{14}$$

where n_break_i is the number of nodes deleted when there is no link between the two nodes due to random deletion after

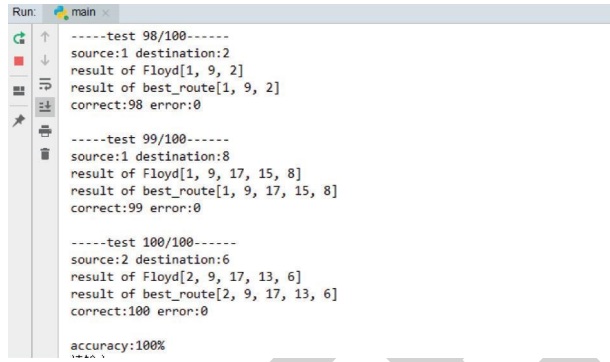

Figure 5: Effectiveness test of dynamic fault-tolerant routing algorithm.

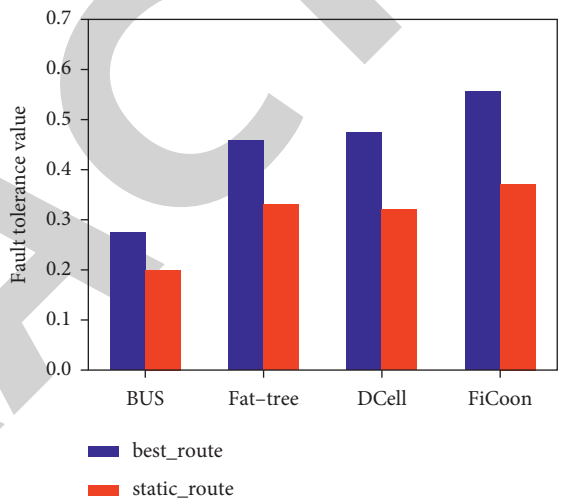

Figure 6: Comparison of time consumption of benchmark algorithm.

Table 3: Fault tolerance of dynamic fault-tolerant routing in various topologies.

	BUS	Fat tree	DCell	FiConn
best_route	0.2765	0.4598	0.4778	0.5569
static_route	0.2025	0.3316	0.3219	0.3716

any two nodes are selected in Figure 6. $1/n \sum_0^n n\_\text{break}_i$ is repeated n times, and the average value is taken to eliminate contingency. Its value is between 0 and 1, and 1 represents the most ideal full connection. The fault tolerance of the algorithm under various topologies is tested below. The test results are shown in Table 3 and Figure 6.

5.3. Algorithm Performance Analysis. In the data center network topology tested in this study, the time consumption of each algorithm mainly depends on two parts: one is the scale of the data center network topology (mainly depends on the number of nodes and links in the topology) and the other is the ratio of link nodes (i.e., the graph is dense graph or sparse graph). We mainly discuss the impact of link-node

Table 4: Time consumption of each algorithm under fat tree (unit: ms).

	1.15	1.20	1.25	1.30	1.35	1.40	1.45	1.50	1.55	1.60
Floyd	1.764	1.822	1.922	1.775	1.781	1.946	1.962	1.863	1.843	1.870
double_stack	0.839	1.099	1.221	1.577	1.837	2.759	5.757	7.869	9.771	12.062
$\alpha = 1.3$	0.929	1.211	1.331	1.817	1.871	2.107	2.110	2.099	1.992	2.094
$\alpha = 1.4$	0.904	1.137	1.327	1.673	2.037	2.110	2.166	2.167	1.937	2.126
$\alpha = 1.5$	0.908	1.112	1.311	1.612	2.011	2.907	5.979	2.003	1.937	2.023

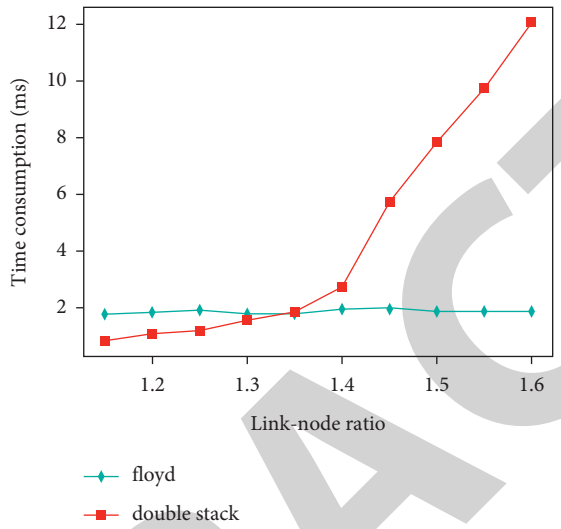

Figure 7: Comparison of time consumption of benchmark algorithm.

ratio on the performance of the algorithm under the same topology size.

Firstly, the fat tree topology with $k = 4$ and 20 nodes is used to test. The fat tree topology has a large number of redundant links, strong fault resistance, and high link-node ratio. Two benchmark algorithms and dynamic fault-tolerant routing algorithms with different values are tested to compare the time overhead (average end-to-end delay) of fat tree topology routing under the same topology scale and link-node ratio. The test data are shown in Table 4.

Figure 7 shows two benchmark algorithms by simulating link failures to achieve different link-node ratios when the number of fat tree nodes remains unchanged (the problem scale does not change). When fat tree is just initialized, the link-node ratio can reach 1.6, and then, it is reduced to 1.15 according to the gradient equal difference of 0.05. t is obvious that the intersection of the two lines is approximately 1.35. Next, the time consumption when α is 1.3, 1.4, and 1.5 will be compared, respectively.

Figure 8 shows that there will be additional time consumption when the value is 1.5. When the value is 1.3 and 1.4, the dynamic fault-tolerant routing algorithm can perfectly call the function with less time consumption in the two functions. Although it needs to read and judge the current node and link number before dynamic call, it will consume some additional time. It makes the time consumption of dynamic fault-tolerant routing function slightly higher, but it can ensure the efficiency of routing algorithm. It is better than the two benchmark algorithms.

The second test uses a 25-node DCell topology as an example. DCell adopts a recursive method to ensure the reliability of the network, and its link nodes are relatively low. Under the same topology scale and link-node ratio, the time consumption of two benchmark algorithms and dynamic fault-tolerant routing algorithms with different values is tested. The test data are shown in Table 5.

In the DCell topology, the link-node ratio can only reach 1.2 when there is no link or node failure. In this case, the three dynamic fault-tolerant routing algorithms with α value will complete the optimal routing based on double-stack DFS. This also exists in the later bus topology. It can be seen that when the link nodes of the topology are relatively low, the dynamic fault-tolerant routing can also have high efficiency.

This section concludes that in terms of time, the dynamic fault-tolerant routing with $\alpha = 1.3$ and $\alpha = 1.4$ can well select the more efficient algorithm of the two benchmark algorithms to complete routing generation, and the overall efficiency is better than that of the two benchmark algorithms and when α is other values.

The test of memory consumption in this study is still carried out under the condition of fixed number of nodes (inconvenient problem scale). In this study, $k = 4$ fat tree topology with 20 nodes is used to test the memory consumption of two benchmark algorithms and dynamic fault-tolerant routing algorithms with different values under the same topology scale and link-node ratio. The test data are shown in Table 6.

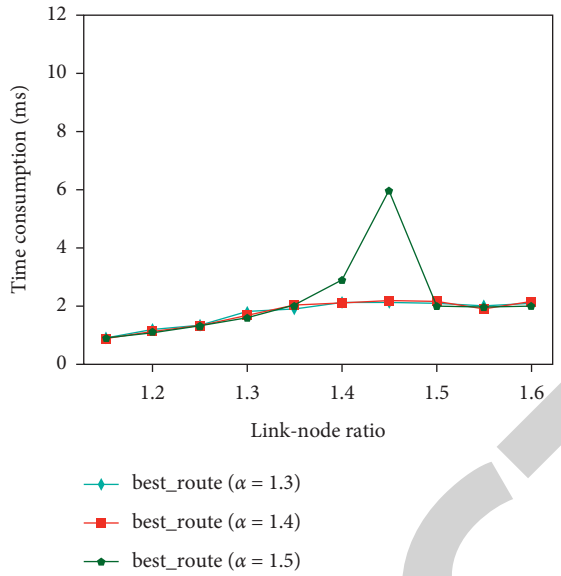

Figure 8: Comparison of time consumption between different α value routing algorithms.

Table 5: Time consumption of each algorithm under DCell (unit: ms).

	1.00	1.04	1.08	1.12	1.16	1.20
Floyd	3.902	3.835	3.889	4.414	4.097	4.307
double_stack	0.380	0.400	0.623	0.649	0.934	1.194
best_route (α = 1.3)	0.609	0.620	0.751	0.879	1.183	1.397
best_route (α = 1.4)	0.601	0.619	0.749	0.889	1.177	1.392
best_route (α = 1.5)	0.609	0.620	0.750	0.881	1.219	1.401

Table 6: Memory occupied by algorithm under fat tree structure (unit: MB).

	1.15	1.20	1.25	1.30	1.35	1.40	1.45	1.50	1.55	1.60
Floyd	0.0492	0.0501	0.0518	0.0502	0.0504	0.0518	0.0504	0.0505	0.0502	0.0513
Double_stack	0.0025	0.0024	0.0026	0.0026	0.0027	0.0026	0.0028	0.0027	0.0028	0.0028
α = 1.3	0.0026	0.0025	0.0026	0.0504	0.0510	0.0513	0.0502	0.0511	0.0512	0.0514
α = 1.4	0.0026	0.0025	0.0026	0.0026	0.0028	0.0518	0.0504	0.0510	0.0503	0.0510
α = 1.5	0.0026	0.0026	0.0026	0.0026	0.0027	0.0025	0.0026	0.0503	0.0503	0.0511

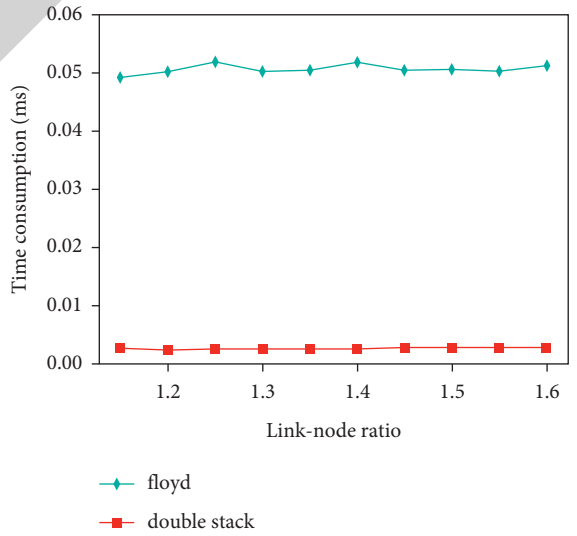

Figure 9: Comparison of memory consumption of different benchmark algorithms.

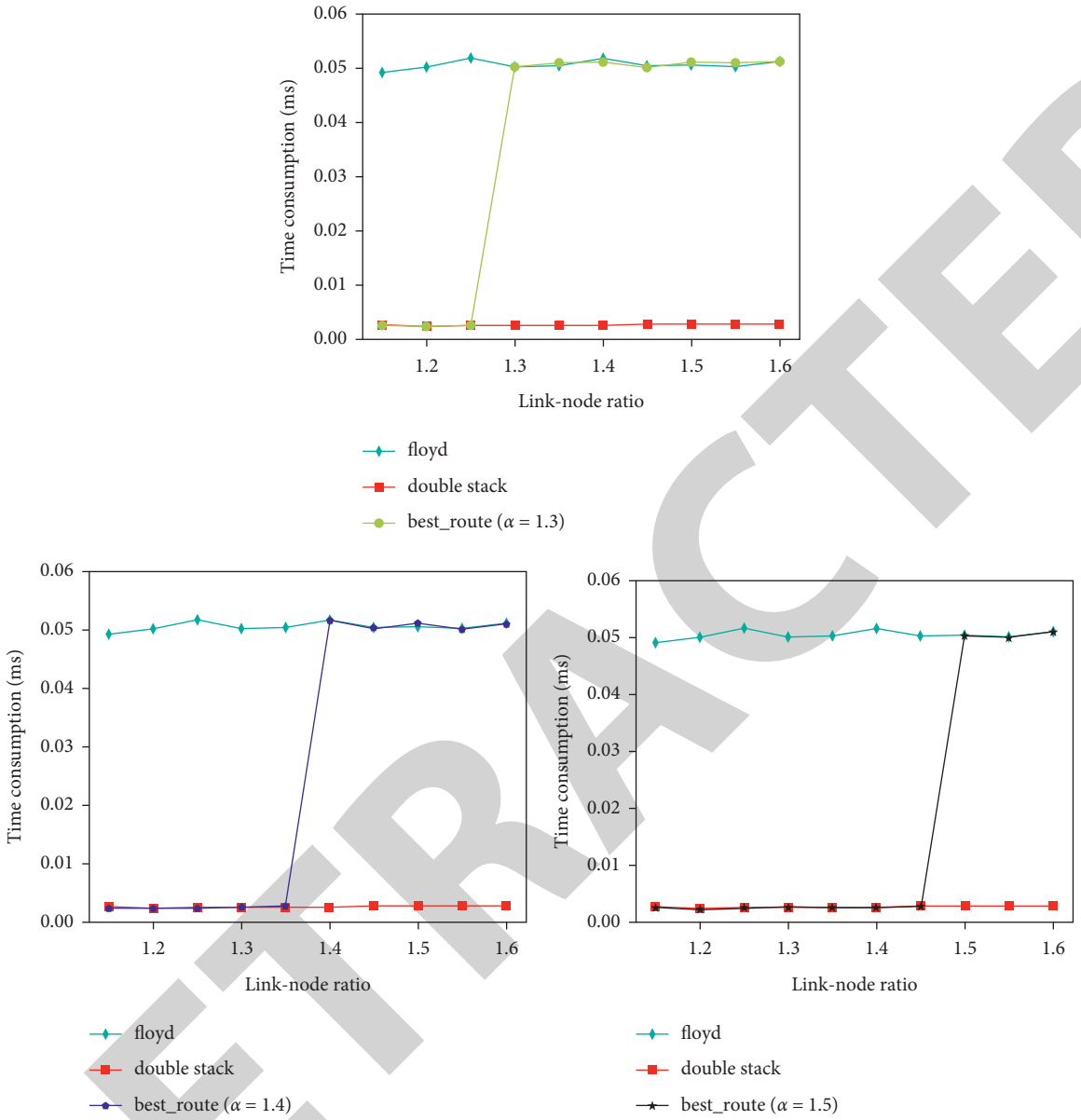

Figure 10: Comparison of time consumption between different α value routing algorithms and benchmark algorithms.

Figure 9 shows that the memory consumption of the two benchmark algorithms is compared. It can be seen that the memory consumption of the two benchmark algorithms is independent of the link-node ratio, which can be obtained from the algorithm analysis in the previous section. The spatial complexity of Floyd is fixed as $O(N^2)$, while the spatial complexity of DFS based on double stack is fixed as $O(N)$. The spatial complexity of both is only related to the size of network topology (problem scale), and under normal circumstances, the spatial complexity of Floyd is always greater than that of DFS based on double stack.

As shown in Figures 10 and 11, the smaller the value of α, the more Floyd will be called by the dynamic fault-tolerant routing algorithm, which will increase the average memory consumption of the dynamic fault-tolerant routing algorithm. The larger the value of α, the more dynamic fault-tolerant algorithms will call double-stack DFS to reduce the average memory consumption. Therefore, under the condition of ensuring the time efficiency of the algorithm, we need to increase the value of α as much as possible. At the same time, combined with the content of the previous section, when $\alpha = 1.3$ and $\alpha = 1.4$ can better show time efficiency, this study selects $\alpha = 1.4$ as the final threshold, which will bring lower average memory consumption. In terms of performance, the dynamic fault-tolerant routing time consumption of $\alpha = 1.4$ sensitively selects the algorithm with shorter time consumption and reduces the average memory consumption as much as possible under the condition of ensuring the time performance, which perfectly solves the routing problem of the data center network.

When the source node and destination node have no fault, it can achieve 100% accuracy, and the fault-tolerant

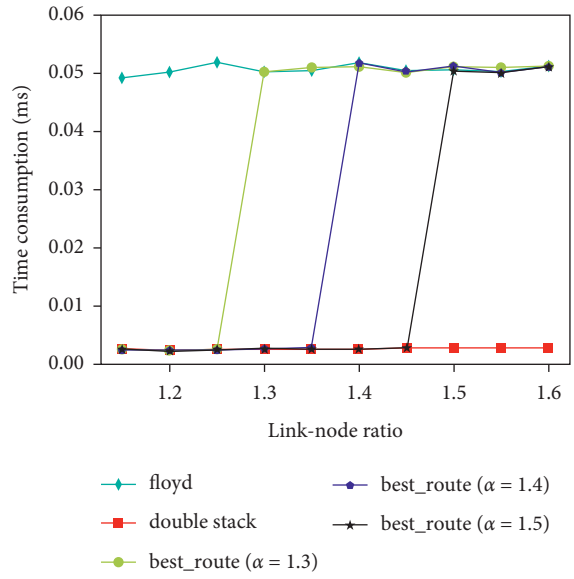

Figure 11: Comparison of memory consumption between different α value routing algorithms.

performance is significantly higher than the static routing algorithm. In terms of performance, the dynamic fault-tolerant routing time consumption of $\alpha = 1.4$ sensitively selects the algorithm with shorter time consumption and reduces the average memory consumption as much as possible under the condition of ensuring the time performance, which perfectly solves the routing problem of the data center network.

6. Conclusion

In this study, we propose a fault-tolerant routing algorithm suitable for data centers in cloud environments. Dynamic routing algorithms apply to macro-data center network topologies, not specific types of topologies. The algorithm combines the characteristics and advantages of software-defined networks, uses the number of links and the ratio of nodes, roughly divides the network topology into dense graphs and sparse graphs, and invokes dual-stack-based improved Floyd and DFS to obtain optimal routing paths.

Although the research of this study has completed dynamic fault-tolerant routing, there are still some problems that have not been perfectly solved. Due to the limitation of conditions and the author's limited ability, the following problems are not further discussed and studied in the study, which need to be improved and solved in the future work:

(1) For dense graphs, there is currently no effective single-source shortest path algorithm that can solve negatively weighted edges. Using the Floyd algorithm to solve the shortest path between two nodes in a dense graph will cause a certain performance waste.

(2) The dynamic factor of the dynamic fault-tolerant routing algorithm is only a macro-selection of the global edge and the number of nodes in the graph (i.e., whether the entire graph is dense or sparse). For a fixed source node and destination node, the dynamic factor cannot be modified. Some routing performance may be degraded [35–37].

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the Natural Science Foundation of China under grant (Nos. 62172291 and 62102196), the Natural Science Foundation of Jiangsu Province (No. BK20200753), the Jiangsu Postdoctoral Science Foundation Funded Project (No. 2021K096A), the Future Network Scientific Research Fund Project (No. FNSRFP-2021-YB-60), the Natural Science Fund for Colleges and Universities in Jiangsu Province (No. 21KJB520026), the Fundamental Research Funds for the Central Universities JL (No. 93K172021K03), and the Jiangsu Provincial Double-Innovation Doctor Program (No. RSCBS2020-01-04).

References

[1] C. Togay, A. Kasif, and C. Catal, "A firewall policy anomaly detection framework for reliable network security," *IEEE Transactions on Reliability*, vol. 99, no. 01, pp. 1–9, 2021.

[2] X. Xue and C. Jiang, "Matching sensor ontologies with multi-context similarity measure and parallel compact differential evolution algorithm," *IEEE Sensors Journal*, vol. 21, no. 21, pp. 24570–24578, 2021.

[3] W. Fan, J. He, Z. Han, P. Li, and R. Wang, "Intelligent resource scheduling based on locality principle in data center networks," *IEEE Communications Magazine*, vol. 58, no. 10, pp. 94–100, 2020.

[4] L. Guo, "Reliability analysis of twisted cubes," *Theoretical Computer Science*, vol. 707, pp. 96–101, 2018.

[5] L. Lin, S.-Y. Hsieh, R. Chen, L. Xu, and C.-W. Lee, "The relationship between g -restricted connectivity and g-Good-Neighbor fault diagnosability of general regular networks," *IEEE Transactions on Reliability*, vol. 67, no. 1, pp. 285–296, 2018.

[6] X. Xue, X. Wu, and C. Jiang, "Integrating sensor ontologies with global and local alignment extractions," *Wireless Communications and Mobile Computing*, Article ID 6625184, 10 pages, 2021.

[7] N. Liu, W. Fan, and J. Fan, "Performance evaluation of fault tolerant routing algorithm in data center networks," *Communications in Computer and Information Science*, pp. 17–33, 2021.

[8] L. Guo, G. Su, W. Lin, and J. Chen, "fault tolerance of locally twisted cubes," *Applied Mathematics and Computation*, vol. 334, pp. 401–406, 2018.

[9] W. Fan, P. Li, Z. Han et al., "Dynamic virtual network embedding of mobile cloud system based on global resources in internet of vehicles," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 8, pp. 8161–8174, 2021.

[10] B. Rauf, H. Abbas, and A. M. Sheri, "Enterprise integration patterns in SDN: a reliable, fault-tolerant communication framework," *IEEE Internet of Things Journal*, vol. 8, no. 8, pp. 6359–6371, 2020.

[11] R. Fernandes, C. Marcon, R. Cataldo, and J. Sepulveda, "Using smart routing for secure and dependable NoC-based MPSoCs," *IEEE/ACM Transactions on Networking*, vol. 28, no. 3, pp. 1158–1171, 2020.

[12] X. Xue and J. Zhang, "Matching large-scale biomedical ontologies with central concept based partitioning algorithm and adaptive compact evolutionary algorithm," *Applied Soft Computing*, vol. 106, no. 01, pp. 1–11, 2021.

[13] X. Xue and J. Chen, "Matching biomedical ontologies through compact differential evolution algorithm with compact adaption schemes on control parameters," *Neurocomputing*, vol. 458, pp. 526–534, 2021.

[14] D. Cohen, M. Kelly, and X. Huang, "Trustability based on beta distribution detecting abnormal behavior nodes in WSN," in *Proceedings of the 2013 19th Asia-Pacific Conference on IEEE Communications(APCC)*, pp. 333–338, IEEE, Denpasar, Indonesia, 29 August 2013.

[15] P. R. Vamsi and P. K. Batra, "An integrated trust model for secure geographic routing in wireless sensor networks," *Engineering & Systems*, pp. 1–6, 2014.

[16] Z. Luo, R. Wan, and X. Si, "An improved ACO-based security routing protocol for wireless sensor networks," *International Conference on Computer Sciences & Applications*, vol. 42, no. 2, pp. 90–93, 2013.

[17] F. Rhamdani, N. A. Suwastika, and M. A. Nugroho, "Equal-cost multipath routing in data center network based on software defined network," in *Proceedings of the 2018 6th International Conference on Information and Communication Technology (ICoICT)*, vol. 1–8, IEEE, Bandung, Indonesia, 3 May 2018.

[18] Y. Cai and C. Wang, "Software defined data center network hybrid routing mechanism," *Journal of Communication*, vol. 37, no. 04, pp. 44–52, 2016.

[19] D. Peng and X. Lai, "Multi path routing algorithm for fat-tree data center network based on SDN," *Computer Engineering*, vol. 44, no. 4, pp. 41–45, 2018.

[20] T. Lei and Z. Lin, "Software defined data center network multipath routing algorithm based on branch and bound method," *Minicomputer system*, vol. 39, no. 08, pp. 1713–1718, 2018.

[21] N. Ya, X. Wang, and S. Zhang, "Multipath fault-tolerance routing mechanism in data center network," in *Proceedings of the 2018 17th International Symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES)*, IEEE, Wuxi, China, 19 October 2018.

[22] C. Jiang, W. L. Liang, and M. Xu, "MTR: fault Tolerant routing in clos data center network with miswiring links," in *Proceedings of the 2014 IEEE 20th International Workshop on Local & Metropolitan Area Networks (LANMAN)*, 21 May 2015.

[23] W. Fan, F. Xiao, J. Fan, Z. Han, J. L. Sun, and W. Ruchuan, "fault-tolerant routing with load balancing in LeTQ networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 1109, no. 10, p. 1, 2021.

[24] J. Ying, "Research on cloud computing oriented data center network architecture design," *Network security technology and Application*, vol. 15, no. 05, pp. 81-82, 2021.

[25] Z. Zhou and F. Li, "Modeling and quantitative calculation of server energy consumption in cloud data center," *Journal of Hunan University*, vol. 48, no. 04, pp. 36–44, 2021.

[26] K. Sharma and R. N. Yadav, "An adaptive, fault tolerant, flow-level routing scheme for data center networks," *Computer Networks*, vol. 175, Article ID 107235, 2020.

[27] Y. Ling, A. Ye, and L. Xu, "Secure localization algorithm of sensor network nodes based on reputation mechanism," *Computer application*, vol. 32, no. 1, pp. 70–73, 2012.

[28] S. Ahmed, S. Al-Rubeaai, and K. Tepe, "Novel trust framework for vehicular networks," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 10, pp. 9498–9511, 2017.

[29] N. Banerjee, S. Chakraborty, and V. Raman, "Improved space efficient linear time algorithms for BFS," *DFS and Applications*, Springer, Cham, 2016.

[30] W. Zhao and Z. Gong, "Comparative analysis of several classical shortest path algorithms," *Journal of Chifeng University (Natural Science Edition)*, vol. 34, no. 12, pp. 47–49, 2018.

[31] Y. Li, "Improvement of Dijkstra algorithm for dealing with the shortest path of negative weight graph and determining negative ring," *China new communications*, vol. 1, no. 07, pp. 166-167, 2019.

[32] J. Gong, Z. Niu, and Y. Zhang, "Multi objective path planning of campus meal delivery robot based on local dimension reduction Dijkstra algorithm," *Journal of Shandong University of Technology*, vol. 35, no. 04, pp. 75–80, 2021.

[33] K. Arai, "Routing protocol based on floyd-warshall algorithm allowing maximization of throughput," *International Journal of Advanced Computer Science and Applications*, vol. 11, no. 6, pp. 436–441, 2020.

[34] B. Zhong, Y. Hu, and J. Yang, "Research based on the Python Networkx toolbox Analysis of the trade network structure from an information flow perspective," *Journal of Physics: Conference Series*, vol. 1646, no. 1, pp. 1–6, 2020.

[35] Y. Li, C. S. Chen, Y.-Q. Song, Z. Wang, and Y. Sun, "Enhancing real-time delivery in wireless sensor networks with two-hop information," *IEEE Transactions on Industrial Informatics*, vol. 5, no. 2, pp. 113–122, 2009.

[36] W. Fan, J. He, M. Guo, P. Li, Z. Han, and R. Wang, "Privacy preserving classification on local differential privacy in data centers," *Journal of Parallel and Distributed Computing*, vol. 135, no. 01, pp. 70–82, 2020.

[37] W. Fan, F. Xiao, X. Chen, L. Cui, and S. Yu, "Efficient virtual network embedding of cloud-based data center networks into optical networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 11, pp. 2793–2808, 2021.