Performance Evaluation of Fault Tolerant Routing Algorithm in Data Center Networks

Ningning Liu[1], Weibei Fan[2](✉), and Jianxi Fan[3]

[1] Suzhou Vocational University, Suzhou 215104, China
[2] Nanjing University of Posts and Telecommunications, Nanjing 210003, China
wbfan@njupt.edu.cn
[3] Soochow University, Suzhou 215031, China

Abstract. Nowdays, the vigorous development of cloud computing technology has brought great changes to the development of the whole information industry. The traditional data center network topology construction method and the operation mechanism of the network layer control plane are solidified, which have been difficult to meet the increasing demand for high performance and high cost performance under the new situation. Researchers map the topology of the data center network to an undirected graph, and use graph algorithms to implement fault-tolerant routing in complex networks. However, the shortest path algorithm of some early graphs cannot be applied to all topological graphs. In this paper, we analyze the advantages and disadvantages of related algorithms, and propose a fault-tolerant routing algorithm for all networks, which can dynamically call different algorithms according to the current number of nodes and links. The experimental results show that it can effectively improve the accuracy and fault tolerance of the algorithm and reduce the consumption of time and memory.

Keywords: Data center network · Graph algorithm · Fault tolerant routing · Performance evaluation

1 Introduction

Cloud computing and its related technologies have made great development and progress, which have also brought great changes to the computer industry. Data center network refers to an infrastructure of data center network, which is connected with switches and servers by using high-speed links. The traditional data center network topology construction method and the operating mechanism of the network layer control plane are solidified. It has been difficult to meet the increasing demand for high performance and high cost performance under. 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 by using the concept of node security level. Therefore, it is more and more important to design a better fault-tolerant routing strategy to record the information of the optimal path in the system as much as possible, and realize more effective fault-tolerant routing in the case of failure in the system, so as to improve the performance of the whole system. The optimization of fault-tolerant routing algorithm can effectively improve the security performance of data center network and ensure the reliable communication of data.

1.1 Related Work

Fault-tolerant routing algorithm is a significant and popular research direction in the field of computer networks. Many scholars have proposed an algorithm design or optimization for a specific network. At first, the routing algorithm mostly adopts the Equivalent Cost Multipath Routing (ECMP) [1], but with the emergence of Software Defined Network (SDN), the control surface and data surface of network equipment are separated, so as to realize the flexible statistical control of network traffic, which also provides a good platform for the proposal of some subsequent routing algorithms. Cai et al. proposed Software defined Hybrid Routing (SHR) [2], determined the threshold according to the statistical results, divided the data flow into large flow and small flow, adopted adaptive routing algorithm for large flow, and adopted traffic independent routing algorithm for small flow, meeting the transmission requirements of different large and small flows. Subsequently, Peng et al. proposed multi-path routing on link real-time status and flow characteristics (MLF) [3]. The algorithm proposed the idea of transforming topology into weighted directed graph, and adopted Dijkstra algorithm for the mapped graph of topology graph. The algorithm has higher link utilization and network throughput in Fat-tree environment. Then, Lei et al. proposed a multi path routing algorithm based on branch and bound in software defined data center networks (mpb-aa) [4], 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, Nan et al. proposed fault tolerance effect and cost function based multipath routing mechanism (feac) [5], designed a feasible path set generation algorithm using heuristic idea, and then used efficiency function and cost function to find the optimal solution. After that, some routing algorithms not only consider the failure of the node itself, but also consider the failure of the link or link connection error. Chang et al. proposed the miswiring tolerant routing protocol (MTR) in the cloud environment data center [6], which uses the openflow controller to complete the physical information collection, map the servers and devices in the physical network to the blueprint, detect wiring errors and facilitate the calculation of solutions, and then modify the configuration routing table through the controller to complete the routing correction.

1.2 Our Contributions

The main results of this paper are as follows:

1. According to the implementation of algorithms in the current data center network (DCN) and software defined network (SDN) environment, this paper analyzes various previous fault-tolerant routing algorithms, and implements Floyd algorithm and double algorithm based on double_stack. Based on this strategy, this paper

writes a program to encapsulate the two, and uses the ratio of edge ratio to node to automatically select an algorithm.

2. This paper reads TXT through the object-oriented high-level programming language Python to complete the construction of network topology and the display of topology shape.
3. We built several models such as bus network, Fat-tree, DCell and Bcube, and tested their algorithm speed, memory occupation, path finding accuracy and other performance indicators under different network topologies.
4. The results of the algorithm are analyzed and studied to find the appropriate threshold so that the algorithm can automatically adapt to different network conditions. The advantages and disadvantages of the algorithm are described, and the next work direction and goal are pointed out.

1.3 Organization of the Paper

The rest of this paper is organized as follows:

The second section mainly introduces the basic knowledge of fault-tolerant routing algorithm of data center network in cloud environment; the third section introduces 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; in the fourth section, the algorithm is tested experimentally; the last section mainly contains the summary of the full text and the outlook for the future.

2 Preliminaries

Cloud environment [7] refers to the Internet or big data environment that can provide computing power, storage capacity or virtual machine services to users or various application systems on demand from the dynamically virtualized resource pool.

DCN [8–10] 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.

2.1 Software Defined Network Mapping to Undirected Graph

With the development of software defined network SDN, we can map the data center network topology to undirected graph. The previously introduced algorithms such as MLF, mpb-aa and MTR have been mapped to graph, and some graph algorithms are used to complete fault-tolerant routing algorithms. This paper only discusses the situation in the ideal network, considers the shortest path finding algorithm based on the number of routing hops without bandwidth delay requirements, and realizes the efficient fault-tolerant path of dynamically calling Freud algorithm and double stack method in the case of node or link failure.

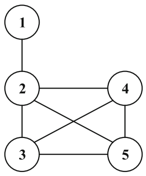
2.2 Data Center Network Definition Storage Mode

The data center topology is constructed here, and each link and node in the topology are recorded in text form. The format of each line can be defined as a ternary formula $L_i = \{node_s, node_d, value_{sd}\}$, where L_i represents a line in the plain text file of the stored data center network topology, $node_s$, $node_d$ here only represents the serial numbers of the two nodes connected by the link. Here, because they are mapped into an undirected graph, the sequence of the two nodes can be reversed. $value_{sd}$ represents the weight of the path between nodes s and d. Because this paper only discusses the ideal case, it defaults to 1, which can be extended according to the actual situation.

2.3 Undirected Graph Storage Mode

In all kinds of graph algorithms, undirected graphs usually have two storage methods: adjacency matrix and adjacency table. In this paper, the functions read_mtx() and get_map() are read from the data center network definition file and transformed into two storage forms respectively, so as to facilitate the call and search of fault-tolerant routing algorithm. The adjacency matrix stores the edge relationship between nodes in the form of matrix, and its corresponding relationship is shown in Fig. 1. Because of the characteristics of the relationship between nodes, the adjacency matrix is more suitable for storing dense graphs. This paper only considers the ideal data center network routing, and takes the number of routing hops as the optimization goal. Therefore, in Python code, the adjacency matrix is dynamically created by using a two-dimensional array. If there is a link between i and j, remember mtx[i][j] = 1 (i < j), otherwise set inf. The specific implementation steps of read_mtx() are as follows:

(1) Read the first line of the topology definition file to obtain the maximum node sequence number.
(2) Dynamically initialize a two-dimensional array mtx, all positions inf.
(3) Traverse the triplet $\{node_s, node_d, value_{sd}\}$ of each line in the text file and set mtx[s][d] = 1.
(4) Set mtx[s][d] = 1 on the other half of the matrix.



$$mtx = \begin{bmatrix} 0 & 1 & inf & inf & inf \\ 1 & 0 & 1 & 1 & 1 \\ inf & 1 & 0 & 1 & 1 \\ inf & 1 & 1 & 0 & 1 \\ inf & 1 & 1 & 1 & 0 \end{bmatrix}$$

Fig. 1. Temporary matrix storage.

The adjacency table can also store undirected graphs. Unlike the adjacency matrix, the adjacency table records the adjacent nodes of each node in the form of linked list array, and its corresponding relationship is shown in Fig. 2. The adjacency table does not have zero or positive infinite space occupation and will not consume too much space resources. The adjacency table is especially suitable for storing sparse graphs. In this paper, the dictionary (dict) in Python is used to realize the linked list array. The key of the dictionary represents the corresponding node in the diagram, and the value of the dictionary is a (list), which is used to store the serial numbers of other adjacent nodes. The specific implementation form of get_map() is as follows:

(1) Read the first line of the topology definition file to obtain the maximum node sequence number.
(2) Create a dictionary with the maximum node sequence number_map, all index values are set to an empty list.
(3) Traverse each triplet $\{node_s, node_d, value_{sd}\}$ in the text file, add d to the value of key s, and add s to the value of key d.

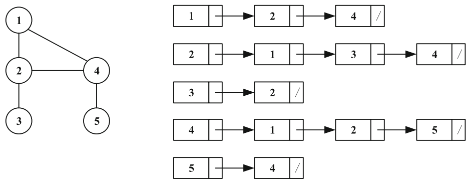


Fig. 2. Temporary connection table storage.

Finally, this paper realizes the visualization of data center topology definition, and realizes the topology diagram display through Networkx, a third-party library of Python.

2.4 Floyd Algorithm

Floyd algorithm [11] was proposed in 1962 and can be used without negative weight edge loop, which is consistent with the data center network in this paper. 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 Fig. 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 every two 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 node u to v, set 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.

(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, the shortest path from any source node i to destination node j is generated through route matrix and output.

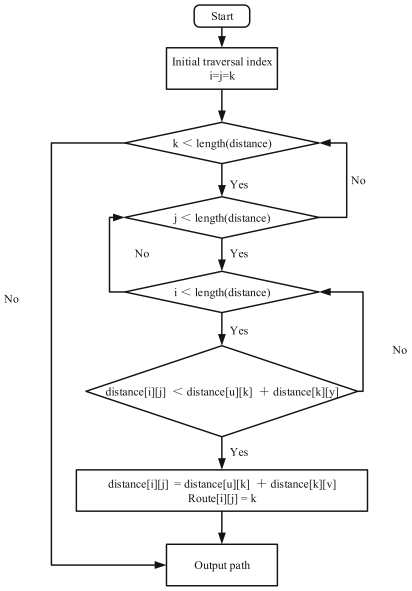

Start
Initial traversal index i=j=k
k < length(distance)
Yes
No
j < length(distance)
No
Yes
i < length(distance)
Yes
No
No
distance[i][j] < distance[u][k] + distance[k][v]
Yes
distance[i][j] = distance[u][k] + distance[k][v]
Route[i][j] = k
Output path

Fig. 3. Floyd algorithm flow chart.

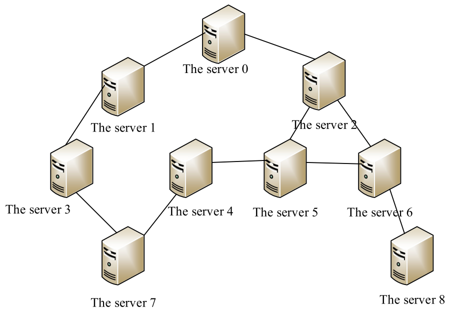


Fig. 4. Sample topology.

The idea of Floyd algorithm belongs to dynamic programming, with time complexity of O(n^3) and space complexity of O(n^2). Floyd algorithm performs best in dense graphs. The efficiency of the algorithm is higher than that of 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 very 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.

In order to supplement the performance of Floyd's fault-tolerant routing algorithm on sparse graph, this paper 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.

2.5 Depth First Algorithm Based on Double Stack

In order to avoid using recursion to realize depth first search, this paper uses two stacks to realize node expansion and path recording, in which main_stack stores a single node, which is used to record the path and side_stack is used to store a list of adjacent nodes of the current element. Take topology Fig. 4 as an example to calculate the optimal path from node 3 to node 6. The specific idea is 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. As shown in Fig. 5.

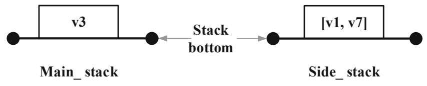


Fig. 5. Double stack based depth first algorithm step 1.

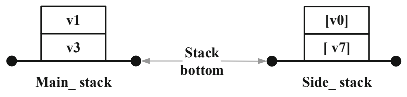


Fig. 6. Double stack based depth first algorithm step 2.

(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, as shown in Fig. 6.
(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 both the main stack and the side_stack, as shown in Fig. 7.

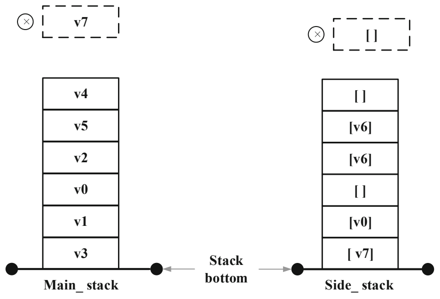


Fig. 7. Double stack based depth first algorithm step 3.

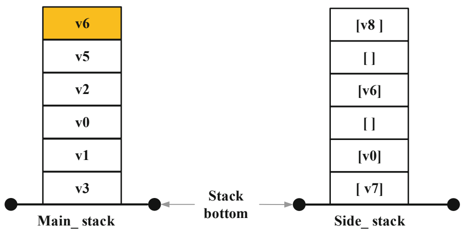


Fig. 8. Double stack based depth first algorithm step 4.

(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, as shown in Fig. 8.
(6) All paths are sorted by path length to obtain the optimal path.

The depth first algorithm based on double stack needs the topology to adopt the form of adjacency table to facilitate the query of adjacent nodes. In this case, the time complexity $O(N + E)$ and space complexity $O(N)$ of the algorithm. 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 O(n!). 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.

3 Design of Dynamic Fault Tolerant Routing Algorithm

3.1 Graph Algorithm Selection and Comparison

In addition to the above two graph path algorithms, the graph shortest path algorithm also includes Bellman-Ford algorithm [12, 13] and its improved algorithm (SPFA), Dijkstra algorithm [14, 15], etc. Table 1 compares the differences of various algorithms and writes out the reasons for choosing DFS and Floyd algorithms. The table assumes that the number of nodes of the graph is N and the number of edges is E.

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 Vertices are closely related	Dense graph Vertices 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	Execute once to specify all paths between two nodes

From the above Table 1, we can find that the application of different algorithms is not consistent. Since this paper is not for a specific topology, but for the data center network in the macro sense, one algorithm obviously can not adapt to all situations, and a variety of algorithms need to be called dynamically according to the conditions of edges and nodes.

In the case of dense graphs, Dijkstra algorithm can not deal with negative weight edges. Although negative weight edges are not within the scope of this paper, they can represent the excitation of a link in practice. Using Dijkstra is not conducive to subsequent expansion and program universality. And for the routing between any two nodes, the efficiency of n times Dijkstra is lower than Floyd. For sparse graphs, Bellman-Ford, SPFA and shortest path DFS have basically the same time complexity and spatial complexity, but Bellman-Ford and SPFA can only judge whether there is a negative weight loop. The shortest path DFS can run directly when there is a negative weight loop, and DFS can record all feasible paths. In this way, if the link often fails and the failed nodes are random, DFS has the shortest path priority, and can find the second path in $O(N)$ time. Therefore, Floyd and DFS algorithms are selected here, which are dynamically called 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 with negative weight, but this paper focuses on the optimal error tolerant routing with hops as the path value.

3.2 Algorithm Design

The core pseudo code of the dynamic fault-tolerant routing algorithm designed in this paper is as follows.

```
Begin
Count the number of now_nodes and now_edges
$\alpha = 1.4$
  if now_edges/now_nodes > $\alpha$
    Call Floyd algorithm
  Else
    Call double_stack_DFS
End
```

The value of α is 1.4. When the edge ratio 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.

4 Experimental Results

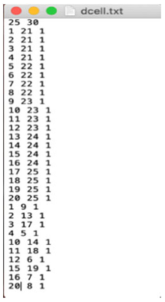
4.1 Experimental Environment and Content

This paper uses plain text file to define the network topology of data center, uses Python 3.9 to implement the dynamic fault-tolerant routing algorithm, and completes 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 paper 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.

Table 2. Test environment parameter table.

Host environment	
Operating system	MacOS Catalina 10.15.7
Processor	2.0 GHz 4-core 10th generation Intel Core i5 processor
Memory	16 GB 3733 MHz LPDDR4X

At the same time, in order to verify the correctness of the algorithm and quantitatively analyze its related performance, this paper uses plain text files to define multiple topologies, including bus topology, k = 4 Fat-tree topology, DCell topology and so on. These topologies are manually input according to the description and definition of previous papers, and the third-party library Networkx [16] is used to display the topology diagram to ensure that the topology diagram is consistent with the blueprint. Its general form is shown in Fig. 9 and 10.



```
25 30
1 21 1
2 21 1
3 21 1
4 21 1
5 22 1
6 22 1
7 22 1
8 22 1
9 23 1
10 23 1
11 23 1
12 23 1
13 24 1
14 24 1
15 24 1
16 24 1
17 25 1
18 25 1
19 25 1
20 25 1
1 9 1
2 13 1
3 17 1
4 5 1
10 14 1
11 18 1
12 6 1
15 19 1
16 7 1
20 8 1
```

Fig. 9. Definition form of DCell network topology.

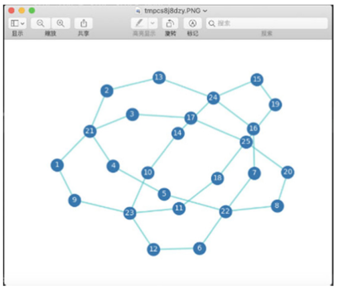


Fig. 10. DCell network topology visualization.

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

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

(2) Comparison of 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 ratio.
(3) Comparison of spatial complexity of Floyd algorithm, depth first algorithm and different value dynamic routing algorithms in topological graphs with different edge node ratios.
(4) Comparison of fault tolerance performance of each topology using dynamic fault-tolerant routing algorithm.

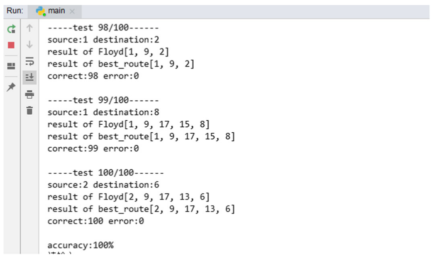
For the experimental parameter selection of the algorithm, after reading any Fat-tree or DCell topology, this paper simulates the routing fault by randomly closing the node or link, reduces the link node ratio by equal difference, and counts the time complexity and space complexity of each algorithm through the test code. Where α is set to 1.3, 1.4 and 1.5.

4.2 Algorithm Validity Test

For the effectiveness of the algorithm, this paper mainly tests 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 algorithm can still work normally and find a feasible path in the case of as many faults as possible.

4.2.1 Accuracy of Algorithm

Here, the Flyod 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 Fig. 11.



```
-----test 98/100------
source:1 destination:2
result of Floyd[1, 9, 2]
result of best_route[1, 9, 2]
correct:98 error:0

-----test 99/100------
source:1 destination:8
result of Floyd[1, 9, 17, 15, 8]
result of best_route[1, 9, 17, 15, 8]
correct:99 error:0

-----test 100/100------
source:2 destination:6
result of Floyd[2, 9, 17, 13, 6]
result of best_route[2, 9, 17, 13, 6]
correct:100 error:0

accuracy:100%
```

Fig. 11. Effectiveness test of dynamic fault tolerant routing algorithm.

4.2.2 Fault Tolerance of Algorithm

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. 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 itself.

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 Eq. 1.

$$ftv = \frac{\frac{1}{n}\sum_0^n n\_break_i}{(node - 2)} \tag{1}$$

Where $\mathrm{n\_break_i}$ is the number of nodes deleted when there is no link between the two nodes due to random deletion after any two nodes are selected in the figure. $\frac{1}{\mathrm{n}}\sum_0^{\mathrm{n}} \mathrm{n\_break_i}$ repeat n times and take the average value 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.

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

4.3 Algorithm Performance Test

4.3.1 Algorithm Time Consumption

In the data center network topology tested in this paper, 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). This paper mainly discusses the impact of link node ratio on the performance of the algorithm under the same topology size.

Firstly, the k = 4 Fat-tree topology with 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 consumption of Fat-tree topology routing under the same topology scale and link node ratio. The test data are shown in Table 4.

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
best_route ($\alpha =$ 1.3)	0.929	1.211	1.331	1.817	1.871	2.107	2.110	2.099	1.992	2.094
best_route ($\alpha =$ 1.4)	0.904	1.137	1.327	1.673	2.037	2.110	2.166	2.167	1.937	2.126
best_route ($\alpha =$ 1.5)	0.908	1.112	1.311	1.612	2.011	2.907	5.979	2.003	1.937	2.023

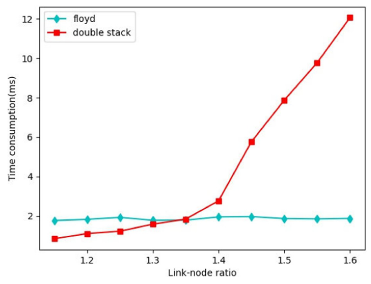


Fig. 12. Comparison of time consumption of benchmark algorithm.

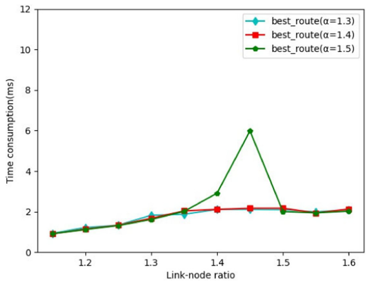


Fig. 13. Comparison of time consumption between different α value routing algorithms.

As shown in Fig. 12, the two benchmark algorithms achieve different link node ratios by simulating link failures 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.

As shown in Figs. 13, when the value is 1.5, there will be an additional period of time consumption. 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 in this paper takes the DCell topology of 25 nodes as an example. DCell uses recursive method to ensure the reliability of the network, and its link nodes are relatively low. It tests the time 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 5.

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 ($\alpha = 1.3$)	0.609	0.620	0.751	0.879	1.183	1.397
best_route ($\alpha = 1.4$)	0.601	0.619	0.749	0.889	1.177	1.392
best_route ($\alpha = 1.5$)	0.609	0.620	0.750	0.881	1.219	1.401

In 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, dynamic fault-tolerant routing with α = 1.3 and α = 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.

4.3.2 Space Consumption of Algorithm

The test of memory consumption in this paper is still carried out under the condition of fixed number of nodes (inconvenient problem scale). In this paper, the 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.

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
best_route ($\alpha = 1.3$)	0.0026	0.0025	0.0026	0.0504	0.0510	0.0513	0.0502	0.0511	0.0512	0.0514
best_route ($\alpha = 1.4$)	0.0026	0.0025	0.0026	0.0026	0.0028	0.0518	0.0504	0.0510	0.0503	0.0510
best_route ($\alpha = 1.5$)	0.0026	0.0026	0.0026	0.0026	0.0027	0.0025	0.0026	0.0503	0.0503	0.0511

As shown in Fig. 14, the memory consumption of the two benchmark algorithms is compared. It can be seen that the memory consumption of the two benchmark algorithms

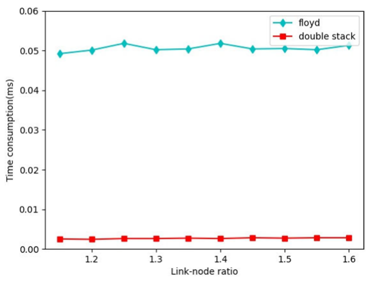


Fig. 14. Comparison of memory consumption of different benchmark algorithms.

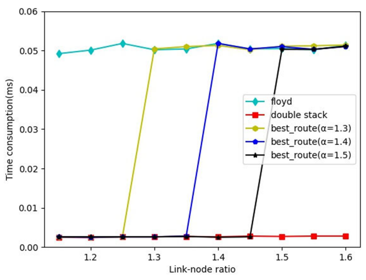


Fig. 15. Comparison of memory consumption between different α value routing algorithms.

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 Fig. 15, 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 paper selects $\alpha = 1.4$ as the final threshold, which will bring lower average memory consumption.

5 Conclusion

In this paper, we propose a fault-tolerant routing algorithm based on data center in cloud environment. The dynamic routing algorithm is suitable for macro data center network topology rather than a specific type of topology. Combined with the characteristics and advantages of software defined network, the algorithm roughly distinguishes the topology into dense graph or sparse graph by using the number of links and node ratio, and call the improved Floyd and DFS based on double stack to obtain the optimal routing path. When the source node and destination node have no fault, it can achieve 100% accuracy, and the fault-tolerant 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.

Although the research of this paper 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 paper, which need to be improved and solved in the follow-up:

(1) There is no efficient single source shortest path algorithm for dense graphs and can solve negative weighted edges. There is a certain performance waste when Floyd is used to solve the shortest path between two nodes in dense graphs.
(2) The dynamic factor of dynamic fault-tolerant routing algorithm is only the macro selection of the global edge and node number of the graph (that is, whether the whole graph is dense or sparse), and can not be modified for fixed source nodes and destination nodes, which may lead to the degradation of some routing performance.

References

1. Rhamdani, F., Suwastika, N., Nugroho, M.: Equal-cost multipath routing in data center network based on software defined network. In: International Conference on Information and Communication Technology (ICoICT), pp. 246–249 (2018)
2. Cai, Y., Wang, C.: Software defined data center network hybrid routing mechanism. J. Commun. **37**(04), 44–52 (2016)
3. Peng, D., Lai, X.: Multi path routing algorithm for fat-tree data center network based on SDN. Comput. Eng. **44**(4), 41–45 (2018)
4. Lei, T., Lin, Z.: Software defined data center network multipath routing algorithm based on branch and bound method. Minicomput. Syst. **39**(08), 1713–1718 (2018)
5. Ya, N., Wang, X., Zhang, S.: Multipath fault-tolerance routing mechanism in data center network. In: International Symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES), pp. 222–226. IEEE (2018)
6. Jiang, C., Wei, L., Xu, M.: MTR: fault tolerant routing in clos data center network with miswiring links. In: International Workshop on Local & Metropolitan Area Networks (LANMAN). IEEE (2015)
7. Ying, J.: Research on cloud computing oriented data center network architecture design. Netw. Secur. Technol. Appl. **2021**(05), 81–82 (2021)
8. Fan, W., He, J., Han, Z.: Intelligent resource scheduling based on locality principle in data center networks. IEEE Commun. Mag. **58**, 94–100 (2021)
9. Fan, W., He, J., Guo, M.: Privacy preserving classification on local differential privacy in data centers. J. Parallel Distrib. Comput. **135**, 70–82 (2020)
10. Fan, W., Xiao, F., Chen, X.: Efficient virtual network embedding of cloud-based data center networks into optical networks. Parallel Distrib. Syst. **32**, 2793–2808 (2021)
11. Arai, K.: Routing protocol based on floyd-warshall algorithm allowing maximization of throughput. Int. J. Adv. Comput. Sci. Appl. **11**(6), 436–441 (2020)
12. Banerjee, N., Chakraborty, S., Raman, V.: Improved space efficient linear time algorithms for BFS, DFS and applications. In: International Computing and Combinatorics Conference, vol. 97, pp. 119–130. Springer, Cham (2016)
13. Zhao, W., Gong, Z.: Comparative analysis of several classical shortest path algorithms. J. Chifeng Univ. (Nat. Sci. Edn.) **34**(12), 47–49 (2018)

14. Li, Y.: Improvement of Dijkstra algorithm for dealing with the shortest path of negative weight graph and determining negative ring. China New Commun. **1**(07), 166–167 (2019)
15. Gong, J., Niu, Z., Zhang, Y.: Multi Objective path planning of campus meal delivery robot based on local dimension reduction Dijkstra algorithm. J. Shandong Univ. Technol. (Nat. Sci. Edn.) **35**(04), 75–80 (2021)
16. Zhong, B., Hu, Y., Yang, J.: Research based on the Python networkx toolbox analysis of the trade network structure from an information flow perspective. J. Phys: Conf. Ser. **1646**(1), 1–6 (2020)