The distance vector routing algorithm based on TCP connection

Qijing Zeng
1930702
Word count: 1397

1. Introduction

Distance Vector Routing is an efficient Routing mechanism, which is being used more and more in the Internet nowadays[1]. However, the efficiency of DV routing algorithms implemented in different ways is different. In order to make this routing algorithm more convenient to use, this paper proposes a simple and efficient method to implement the DV routing algorithm. This method uses TCP to connect adjacent nodes. TCP is bidirectional, so only one connection is needed between each pair of nodes. And it uses the Bellman-Ford algorithm to calculate the path between nodes. Bellman-Ford algorithm is the shortest path algorithm that allows each node to perform operations at the same time [2].

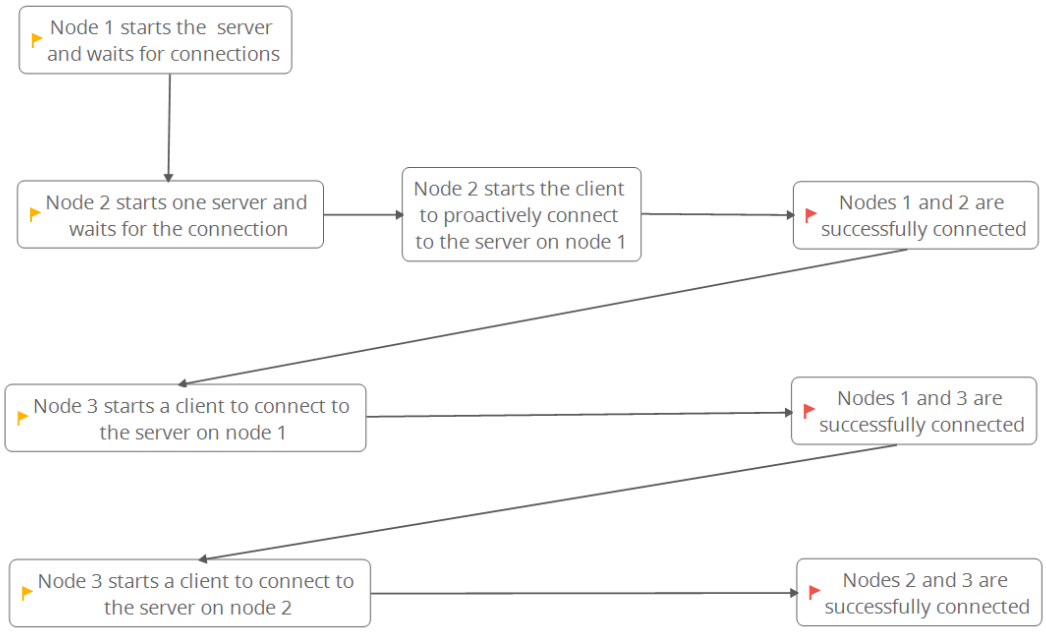
The following will introduce the DV routing algorithm proposed in this paper by describing the methodology, specific implementation steps, and defined python functions, etc. Finally, a suitable plan will be used to test the correctness and efficiency of the algorithm.

2. Methodology

The following will introduce the methodology in accordance with the process required to implement this DV routing algorithm. Three nodes are used as examples.

2.1 Read the configuration file about node information, distribute the information to the three nodes in the order of analysis, run three sub-processes, and initialize their own B-F state to the routing table.

2.2 Start a thread in each child process to establish a TCP connection with its neighbor node. Due to the TCP connection being bidirectional, there is only one TCP connection between each pair of nodes. For instance, the process of establishing connections between the three nodes is shown as follows:



2.3 After establishing a connection, each node sends its initialized routing table to its neighbors. After the node receives the routing information from the neighbor, it uses the Bellman-Ford formula to determine the shortest path to other nodes and updates its routing table. Send its updated routing information to neighbors. Iterate until the algorithm converges. After convergence, each node will output its final DV table to the corresponding json file

3. Implementation

3.1 Steps

a) Divide the functions that need to be implemented into local functions, and define each local function in the code.

b) Integrate local functions into the main code according to the overall flow of ideas.

c) Test code, if there is an error, modify the corresponding code until the desired test result appears

3.2 Python modules used

json, copy, struct, threading, time, collections, dataclasses, socket, typing, multiprocessing

3.3 Functions

Functions are defined to implement some of the corresponding functions. The following table describes the class names, function names, and detailed illustrates. Such as SocketHandle, which has multiple functions, and ItemProcess, which has no function.

Class name	**Function name**	**Specific functions implemented**
	create_socket()	create a socket
	create_server()	create a server
	create_client()	create a client
SocketHandle	__init__()	store all information about the node and use lock to prevent collisions with routing table modifications during concurrency
	show_message()	print message
	send_message()	send the routing table dictionary
	_recv()	receive information block by block
	recv_message()	unpack it with struct.unpack(), then convert the received information from a string to a dictionary return via json.load()
	update()	after receiving the neighbor node information, bellman-Ford algorithm is used to determine the shortest path, update the routing table, and push the latest routing table to all connected neighbor nodes
	save_table()	print out the local DV table and save it to a json file
	get_send_table()	form the table to be sent to the neighbor. replace all "next_hop" in the table with this node.
ServerHandle	accept()	after the connection between the server and the client is successful, the server maintains the connection with the client through threads
	run_client()	perform interaction with the client
ClientHandle	accept()	operations performed by the client after the client connects to the server. Receives the routing table sent by the server, checks whether its own table needs to be updated, and sends the updated routing table to the connected neighbor nodes
ItemProcess		store the routing table of a node
	main()	parse the configuration file, distribute it to three nodes, and run three sub-processes in the order of analysis
	parse_conf()	parse the configuration file

3.4 Problem and solution

Problem: when the child process of the node establishes a TCP connection, if the server receives multiple client connections, it may be blocked.

Solution: asynchronous processing is achieved through multiple threads, allowing multiple clients to connect to the same server at the same time, avoiding blocking.

4. Testing and result

4.1 Test plan

a) Create an “input. txt” file in the current directory and save the information about the weighted graph as follows:

   Node 1

   2 4

   3 2

   Node 2

   3 5

b) Run main.py

4.2 Test results

The test result is divided into two parts. The first part is the process of being printed out in the execution result display area. The content is as follows:

a) after reading that there are three nodes in the configuration file, run three sub-processes, and run threads in the sub-processes, create a server or client, and successfully establish a TCP connection with other nodes. The printed outcome:

   Node<1>: create server: ('localhost', 20000)

   Node<2>: create server: ('localhost', 20001)

   Node<2>: start client connect:('localhost', 20000)

   Node<1>: ('127.0.0.1', 50170) connect !

   Node<3>: start client connect:('localhost', 20000)

Node<1>: ('127.0.0.1', 50171) connect !

Node<3>: start client connect:('localhost', 20001)

Node<2>: ('127.0.0.1', 50172) connect !

b) according to the information in the configuration file, each node initializes its own B-F state to the DV table:

Node<1> init tables:{'2': {'distance': 4, 'next_hop': '2'}, '3': {'distance': 2, 'next_hop': '3'}}
Node<2> init tables:{'3': {'distance': 5, 'next_hop': '3'}}
Node<3> init tables:{}

c) after the connection is successful, each node sends its initial DV routing table information to its neighbor nodes. The method in this article sets the 'next_hop' in the routing table sent to the neighbor to its own node, such as:

Node<1>: send message: {'2': {'distance': 4, 'next_hop': '1'}, '3': {'distance': 2, 'next_hop': '1'}}

d) after the node receives the information from the neighbor, it should use the B-F algorithm to determine the shortest path, and update its routing table correctly. The updated routing table is printed out in the format beginning with the string 'save', and then the routing table is sent to neighboring nodes until convergence. For example, the convergence process between nodes 1 and 2 is as follows,

Node<2>: client accept:{'2': {'distance': 4, 'next_hop': '1'}, '3': {'distance': 2, 'next_hop': '1'}}

Node<2>: save:{'3': {'distance': 5, 'next_hop': '3'}, '1': {'distance': 4, 'next_hop': '1'}}

Node<2>: send message: {'3': {'distance': 5, 'next_hop': '2'}, '1': {'distance': 4, 'next_hop': '2'}}

Node<1>: recv:{'3': {'distance': 5, 'next_hop': '2'}, '1': {'distance': 4, 'next_hop': '2'}}

Node<1>: save:{'2': {'distance': 4, 'next_hop': '2'}, '3': {'distance': 2, 'next_hop': '3'}}

The second part of this test result is after running the code a few seconds (different hardware environments may cause different waiting times), refresh pycharm, three json files corresponding to the nodes will be generated in the current directory, and the local DV table of the node after convergence will be saved in the file. The results are shown in the following table:

File name	File content
1.json	{"2": {"distance": 4, "next_hop": "2"}, "3": {"distance": 2, "next_hop": "3"}}
2.json	{"3": {"distance": 5, "next_hop": "3"}, "1": {"distance": 4, "next_hop": "1"}}
3.json	{"1": {"distance": 2, "next_hop": "1"}, "2": {"distance": 5, "next_hop": "2"}}

4.3 Analysis of test results

According to the node information in the configuration file, manually calculate the DV table initialized by each node, the DV table sent during the convergence process, the updated DV table, and the local DV table of each node after convergence. Compare with the test results and verify The DV routing algorithm implemented in this article is correct from the process to the final output. The correctness and feasibility of this method are verified.

5. Conclusion

With the increasing frequency of DV routing algorithms, the complexity of implementing the algorithm and the efficiency of the algorithm have been paid more and more attention. The DV routing algorithm proposed in this paper first reads the text file storing node information, establishes the process of each node, and then establishes a TCP connection between the nodes to communicate with neighbor nodes, and gradually realizes the convergence process. The test result of the DV routing algorithm proves its correctness and efficiency. In the case of fewer nodes, this method can be promoted as a simple and efficient DV routing algorithm. But when it is applied to multiple nodes, the correctness and efficiency remain to be investigated.

6. Reference

[1] B. Wang. *et al.*, "A secure routing model based on distance vector routing algorithm, " *Sci. China Inf. Sci.,* vol. 57, pp.1–13, Jan. 2014. [Online]. Available: https://doi-org.ez.xjtlu.edu.cn/10.1007/s11432-012-4659-7

[2] D. S. Maylawati, *et al.*, "The Purpose of Bellman-Ford Algorithm to Summarize the

Multiple Scientific Indonesian Journal Articles," 2020 6th International Conference on Wireless and Telematics (ICWT), pp. 1-5, 2020. [Online]. Available: https://ieeexplore-ieee-org.ez.xjtlu.edu.cn/document/9243645