



Xi'an Jiaotong-Liverpool University
西交利物浦大学

Course Work Submission

Name	Shen	Wang
ID Number	1824260	
Programme	Information and Computing Science	
Module Title	Introduction to Networking	
Module Code	CAN201	
Assignment Title	Coursework 2: Routing simulation	
Submission Deadline	Monday 15th Jan. 2021, 23:59	
Lecturer Responsible	Fei Cheng	

I certify that:

- I have read and understood the University's definitions of COLLUSION and PLAGIARISM (available in the Student Handbook of Xi'an Jiaotong-Liverpool University).

With reference to these definitions, I certify that:

- I have not colluded with any other student in the preparation and production of this work;
- this document has been written solely by me and in my own words except where I have clearly indicated and acknowledged that I have quoted or used figures from published or unpublished sources (including the web);
- where appropriate, I have provided an honest statement of the contributions made to my work by other people including technical and other support staff.

I understand that unauthorized collusion and the incorporation of material from other works without acknowledgement (plagiarism) are serious disciplinary offences.

Signature: Shen Wang

Date: 2021/1/15

For Academic Office use:	Date Received	Days Late	Penalty

An introduction to CAN201 Coursework 2: Routing simulation

Abstract

Routing is the process of selecting a path across multiple networks. The routing algorithm determine the routing of packets in a node. It will output a routing table in each node of a network. This report is about the Python application of CAN201 coursework 2: routing simulation. The report will introduce the methodology and implementation to simulate routing process. Furthermore, some test results will be provided to help evaluate the simulation more visually.

Contents

1. Introduction	1
2. Methodology	1
3. Implementation	1
4. Testing and results	1
5. Conclusion	1
6. Reference	1

1. Introduction

The aim of this coursework is to create a Python application for a node that simulates the routing process. The initial distance information, IP address and port numbers of neighbour nodes will be provided. The application should load its own information and exchange information with neighbours through UDP transmission. Moreover, Bellman Ford algorithm is requested to be used to calculate distance vectors.

Codes should be in Python3 and run in the provided Linux virtual machine. The application will be run on the same virtual machine with different terminals. Each terminal can be seen as a node. Unfixed number of nodes will be started to simulate the group behaviours of routers.

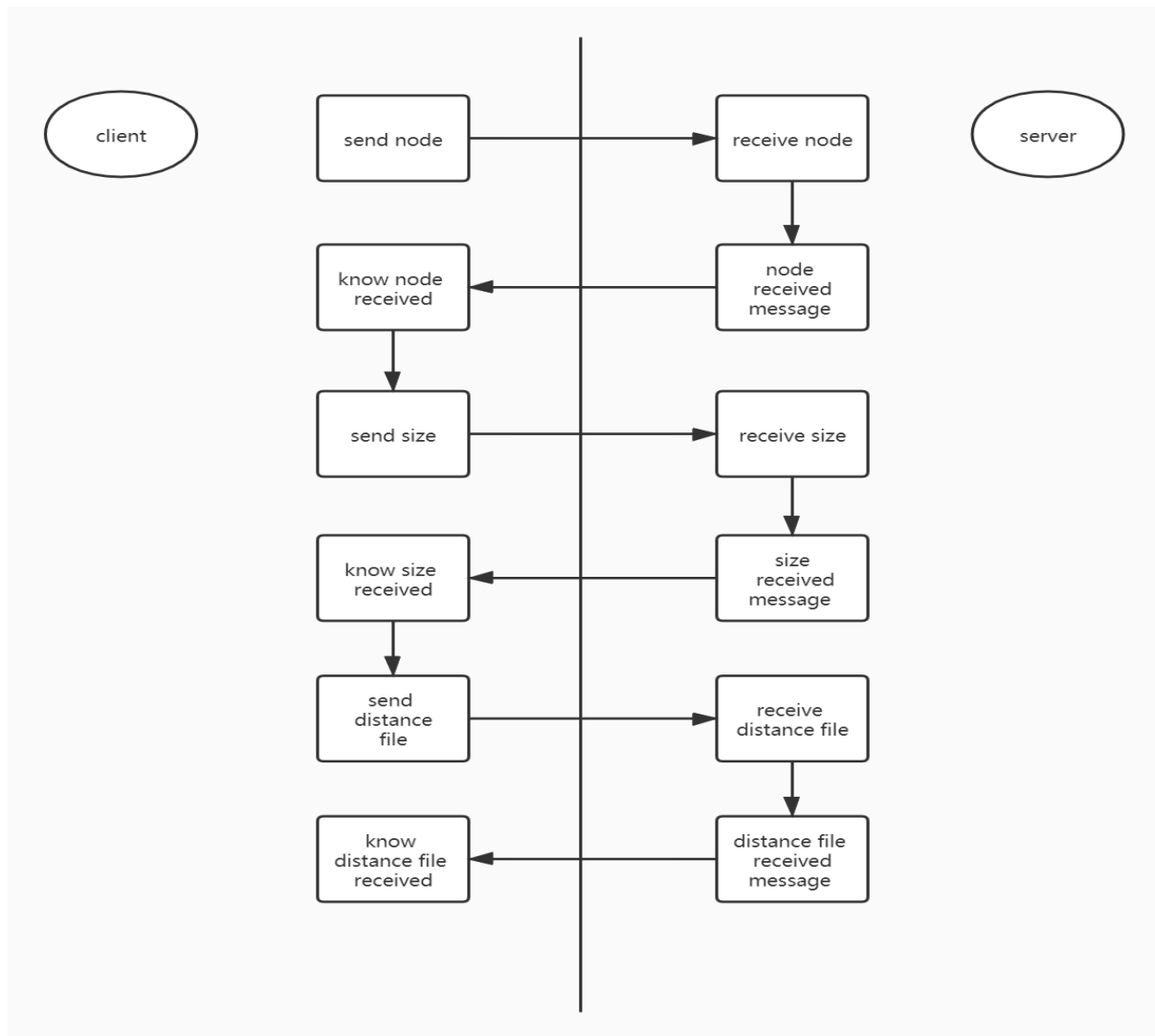
The Bellman Ford algorithm is an algorithm that computes shortest paths from a single source vertex to all of the other vertices in a weighted digraph [1]. Compare with Dijkstra's algorithm, Bellman Ford algorithm is slower. However, it is more versatile as it is capable of handling graphs with negative edge weights.

For this coursework, I create a Python application using multithreading and a UDP based transport protocol. This application can send and receive distance information to and from its neighbour nodes. Then the node will calculate the path in Bellman Ford algorithm and output routing table in .json format.

2. Methodology

I use the Client / Server architecture and socket based on TCP transport protocol to realize the communication between neighbour nodes. In fact, both client and server are the same node. Both the client and the server create temporary .json files to store and manipulate information. The client loads its own IP and distance information and send it to neighbour nodes. The server uses a list to store connected clients' information and check it to decide how to deal with the received.

Here is a simple diagram to help understand my protocol:



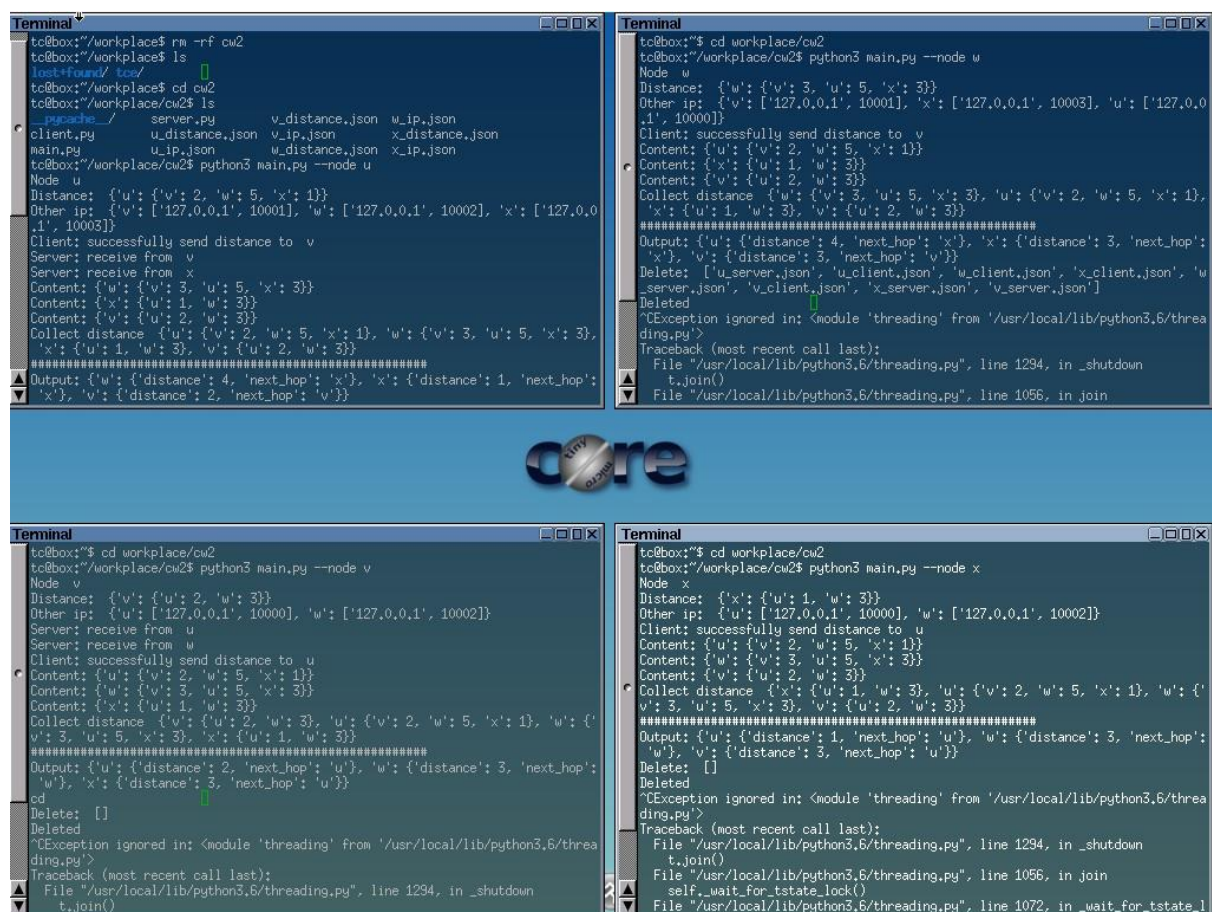
3. Implementation

Firstly, I use argparse to read the content of command line. Then the node parameters are passed to the client. A sub-thread will be start to run server to listen to other clients. The application will read, write and transmit .json files to communicate distance information with neighbour nodes. After all started nodes get others information, they will use Bellman Ford algorithm to calculate the ideal path. Finally output this and delete

the temporary files.

The difficulties I met was the data type [2] and flow control [3]. I solved them by reading and studying references.

4. Testing and results



```
Terminal
tc@box:~/workplace$ rm -rf cw2
tc@box:~/workplace$ ls
lost+found/  test/
tc@box:~/workplace$ cd cw2
tc@box:~/workplace/cw2$ ls
pycache/  server.py  v_distance.json  w_ip.json
client.py  u_distance.json  v_ip.json  x_distance.json
main.py    u_ip.json  w_distance.json  x_ip.json
tc@box:~/workplace/cw2$ python3 main.py --node u
Node u
Distance: {'u': {'v': 2, 'w': 5, 'x': 1}}
Other ip: {'v': ['127.0.0.1', 10001], 'w': ['127.0.0.1', 10002], 'x': ['127.0.0.1', 10003]}
Client: successfully send distance to v
Server: receive from v
Server: receive from x
Content: {'u': {'v': 3, 'w': 5, 'x': 3}}
Content: {'x': {'u': 1, 'w': 3}}
Content: {'v': {'u': 2, 'w': 3}}
Collect distance: {'u': {'v': 2, 'w': 5, 'x': 1}, 'w': {'v': 3, 'u': 5, 'x': 3}, 'x': {'u': 1, 'w': 3}, 'v': {'u': 2, 'w': 3}}
*****
Output: {'u': {'distance': 4, 'next_hop': 'x'}, 'x': {'distance': 1, 'next_hop': 'v'}, 'v': {'distance': 2, 'next_hop': 'u'}}

Terminal
tc@box:~/workplace/cw2$ python3 main.py --node w
Node w
Distance: {'w': {'v': 3, 'u': 5, 'x': 3}}
Other ip: {'v': ['127.0.0.1', 10001], 'x': ['127.0.0.1', 10003], 'u': ['127.0.0.1', 10000]}
Client: successfully send distance to v
Content: {'u': {'v': 2, 'w': 5, 'x': 1}}
Content: {'x': {'u': 1, 'w': 3}}
Content: {'v': {'u': 2, 'w': 3}}
Collect distance: {'u': {'v': 3, 'u': 5, 'x': 3}, 'u': {'v': 2, 'w': 5, 'x': 1}, 'x': {'u': 1, 'w': 3}, 'v': {'u': 2, 'w': 3}}
*****
Output: {'u': {'distance': 4, 'next_hop': 'x'}, 'x': {'distance': 3, 'next_hop': 'v'}, 'v': {'distance': 3, 'next_hop': 'u'}}
Delete: ['u_server.json', 'u_client.json', 'w_client.json', 'x_client.json', 'w_server.json', 'v_client.json', 'x_server.json', 'v_server.json']
Deleted
^CException ignored in: <module 'threading' from '/usr/local/lib/python3.6/threading.py'>
Traceback (most recent call last):
  File "/usr/local/lib/python3.6/threading.py", line 1294, in _shutdown
    t.join()
  File "/usr/local/lib/python3.6/threading.py", line 1096, in join

Terminal
tc@box:~/workplace/cw2$ python3 main.py --node v
Node v
Distance: {'v': {'u': 2, 'w': 3}}
Other ip: {'u': ['127.0.0.1', 10000], 'w': ['127.0.0.1', 10002]}
Server: receive from u
Server: receive from w
Client: successfully send distance to u
Content: {'u': {'v': 2, 'w': 5, 'x': 1}}
Content: {'w': {'v': 3, 'u': 5, 'x': 3}}
Content: {'x': {'u': 1, 'w': 3}}
Collect distance: {'v': {'u': 2, 'w': 3}, 'u': {'v': 2, 'w': 5, 'x': 1}, 'w': {'v': 3, 'u': 5, 'x': 3}, 'x': {'u': 1, 'w': 3}}
*****
Output: {'u': {'distance': 2, 'next_hop': 'u'}, 'w': {'distance': 3, 'next_hop': 'u'}, 'x': {'distance': 3, 'next_hop': 'u'}}
cd
Delete: []
Deleted
^CException ignored in: <module 'threading' from '/usr/local/lib/python3.6/threading.py'>
Traceback (most recent call last):
  File "/usr/local/lib/python3.6/threading.py", line 1294, in _shutdown
    t.join()

Terminal
tc@box:~/workplace/cw2$ python3 main.py --node x
Node x
Distance: {'x': {'u': 1, 'w': 3}}
Other ip: {'u': ['127.0.0.1', 10000], 'w': ['127.0.0.1', 10002]}
Client: successfully send distance to u
Content: {'u': {'v': 2, 'w': 5, 'x': 1}}
Content: {'v': {'u': 3, 'w': 5, 'x': 3}}
Content: {'w': {'u': 2, 'w': 3}}
Collect distance: {'x': {'u': 1, 'w': 3}, 'u': {'v': 2, 'w': 5, 'x': 1}, 'w': {'v': 3, 'u': 5, 'x': 3}, 'v': {'u': 2, 'w': 3}}
*****
Output: {'u': {'distance': 1, 'next_hop': 'u'}, 'w': {'distance': 3, 'next_hop': 'u'}, 'v': {'distance': 3, 'next_hop': 'u'}}
Delete: []
Deleted
^CException ignored in: <module 'threading' from '/usr/local/lib/python3.6/threading.py'>
Traceback (most recent call last):
  File "/usr/local/lib/python3.6/threading.py", line 1294, in _shutdown
    t.join()
  File "/usr/local/lib/python3.6/threading.py", line 1096, in join
    self._wait_for_tstate_lock()
  File "/usr/local/lib/python3.6/threading.py", line 1072, in _wait_for_tstate_l
```

This is the test results of the given test data, which I printed them out.

The image displays four terminal windows arranged in a 2x2 grid, showing the execution of a network simulation program. The program appears to be a Bellman-Ford algorithm implementation for finding shortest paths in a network.

Top Left Terminal: Shows the initial state of the network. The output displays the distance and next_hop values for nodes u, v, and x. The program is running in a loop, and the user has pressed Ctrl+C to interrupt it. The traceback shows the program was interrupted in the `wait_for_tstate_lock()` function.

Top Right Terminal: Shows the state of the network after a series of updates. The output displays the distance and next_hop values for nodes u, v, and x. The program is running in a loop, and the user has pressed Ctrl+C to interrupt it. The traceback shows the program was interrupted in the `wait_for_tstate_lock()` function.

Bottom Left Terminal: Shows the state of the network after a series of updates. The output displays the distance and next_hop values for nodes u, v, and x. The program is running in a loop, and the user has pressed Ctrl+C to interrupt it. The traceback shows the program was interrupted in the `wait_for_tstate_lock()` function.

Bottom Right Terminal: Shows the state of the network after a series of updates. The output displays the distance and next_hop values for nodes u, v, and x. The program is running in a loop, and the user has pressed Ctrl+C to interrupt it. The traceback shows the program was interrupted in the `wait_for_tstate_lock()` function.

I opened the output file. The test results are consistent with the given test results.

5. Conclusion

In this coursework, I used UDP transmission and multiple threads to increase the transmission speed. In addition, Bellman Ford algorithm is used to calculate the shortest path of current node. I have a deeper understanding of data types, file operations, data transmission and routing algorithm.

References:

- [1] "Bellman Ford algorithm." Wikipedia.org.
http://en.wikipedia.org/wiki/BellmanFord_algorithm
- [2] "5. Data Structures." docs.python.org.
<https://docs.python.org/3/tutorial/datastructures.html#more-on-lists>
- [3] "4. More Control Flow Tools." docs.python.org.
<https://docs.python.org/3/tutorial/controlflow.html>