CAN201 Introduction to Networking

Coursework 2

Routing simulation

Contribution to Overall Marks	**25%**
Submission Deadline	**Monday 15th Jan. 2021, 23:59**
Type	**Individual coursework**
Learning Outcome	**[A][B][C][D]**

How the work should be submitted?

- **SOFT COPY ONLY!**
- You must submit your work through Learning Mall.

Specification

Routing algorithms are very important to determine good paths from sending hosts to receiving host through a network of routers. In this coursework, you will simulate the routing process using Python Socket network programming. The goal of this project is to Bellman-Ford distance vector algorithm

Requirements

- You should design only one app for a node, which is used to simulate the routing behavior of a router. For each app, it will start with the initial distance information of neighbors and IP addresses + port numbers of their neighbors. Each app should run without errors and load its own information.

- Each app should implement UDP communication to exchange information with their neighbors. You should create a protocol to exchange distance vector information, as required by Bellman-Ford.

- In order to simulate the group behaviors of routers, unfixed number of apps will be started. The starting order of nodes is not fixed.

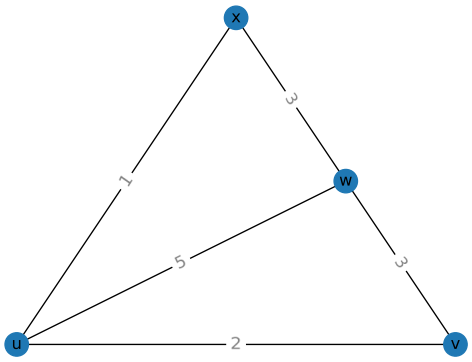
- Nodes should asynchronously exchange distance vector information as needed until the algorithm converges. After convergence, each node should output its distance vector table to a json file (one file per node with a given format).

Data Format (All files are located in CWD, ALL!!!)

Distance information of each node's neighbors is stored in a json file named with the *node name* + "_distance". (They are given, your apps should load them)

For Graph 1 in the right side, there are four json files to represent these four nodes:

u_distance.json	v_distance.json
{ "v": 2, "w": 5, "x": 1 }	{ "u": 2, "w": 3 }
w_distance.json	**x_distance.json**
{ "v": 3, "x": 3 }	{ "u": 1, "w": 3 }



Graph 1: An example

IP addresses + port numbers of each node's neighbors and this node is stored in a json file named with the *node name* + "_ip.json". (They are given, your apps should load it)

For Graph 1:

u_ip.json	v_ip.json
{ "u": ["127.0.0.1", 10000], "v": ["127.0.0.1", 10001], "w": ["127.0.0.1", 10002], "x": ["127.0.0.1", 10003] }	{ "v": ["127.0.0.1", 10001], "u": ["127.0.0.1", 10000], "w": ["127.0.0.1", 10002] }
w_ip.json	**x_ip.json**
{ "w": ["127.0.0.1", 10002], "v": ["127.0.0.1", 10001], "x": ["127.0.0.1", 10003] }	{ "x": ["127.0.0.1", 10003], "u": ["127.0.0.1", 10000], "w": ["127.0.0.1", 10002] }

The output distance vector table of each node should be stored in a json file named with node name + "_output.json". (Your apps should output them)

For Graph 1:

u_output.json

```
{
    "v": {
        "distance": 2,
        "next_hop": "v"
    },
    "w": {
        "distance": 4,
        "next_hop": "x"
    },
    "x": {
        "distance": 1,
        "next_hop": "x"
    }
}
```

v_output.json

```
{
    "u": {
        "distance": 2,
        "next_hop": "u"
    },
    "w": {
        "distance": 3,
        "next_hop": "w"
    },
    "x": {
        "distance": 3,
        "next_hop": "u"
    }
}
```

w_output.json

```
{
    "v": {
        "distance": 3,
        "next_hop": "v"
    },
    "x": {
        "distance": 3,
        "next_hop": "x"
    },
    "u": {
        "distance": 4,
        "next_hop": "x"
    }
}
```

x_output.json

```
{
    "u": {
        "distance": 1,
        "next_hop": "u"
    },
    "w": {
        "distance": 3,
        "next_hop": "w"
    },
    "v": {
        "distance": 3,
        "next_hop": "u"
    }
}
```

What should be submitted:

Codes:

- Python 3;
- Your application can be implemented using multiple Python scripts;
- But there is only one application.

A development report:

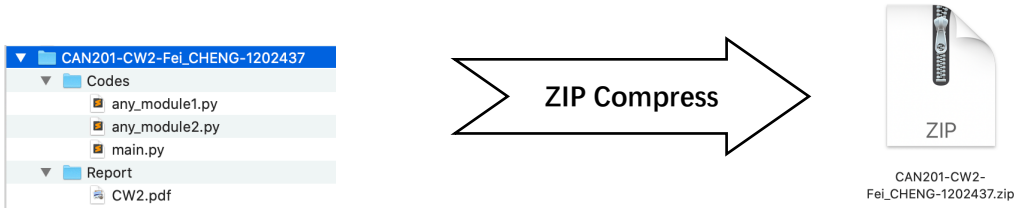
- A cover page with your full name and student ID;
- 3~ 8 pages, single column, 1.5x line space, 2.54cm margins, Serif font[1], font size:12pt;

[1] Eg. Times New Roman, Modern No. 20 or Cambria.

- PDF format, LaTeX is recommended;
- Including:
  - Abstract
  - Introduction: project requirement (do not copy from this document), background, literature review (try to find research papers, development reports or testing report of similar apps), and introduce what you did in this coursework;
  - Methodology: proposed protocols (using FSM or mind map), proposed functions and ideas;
  - Implementation: steps of implementation, program flow charts, programming skills (OOP, Parallel…) you used, what difficulties you met and how to solve them;
  - Testing and results: testing environment, testing plan and testing results (screenshot, tables or curves for showing performance)
  - Conclusion: what you did and why you did it. Future plan and so on.
  - Reference [IEEE format]

Meanwhile, you have to follow the compulsory requirement (no tolerance[2]):

- The file structure of your submission:



- Only ZIP file is allowed to submit.
- The starting Python script file should be named as "main.py";
- Run command: `python3 main.py --node <node_name>`[3]

Marking Criteria

Report (40%)

Marking Criteria	Item	Mark
Structure (5%)	Structure	5%
Contents (30%)	Abstract	2%
	Introduction	4%
	Methodology	6%
	Implementation	7%
	Testing and results	8%
	Conclusion and reference	3%
Format and language (5%)	Report style and format	3%
	Language	2%

Marking scheme:

1. Structure (5%)
   - Well organized structure: 5%

[2] It means that if you do not follow the compulsory requirement, your work will be marked as zero.

[3] Eg. python3 main.py -- node x. Your app should open x_distance.json and x_ip.json and finally output x_output.json.

- Reasonable structure: 3% ~ 4%
- Disordered structure: 0% ~ 2%

2. Contents (30%)
   2.1. Abstract (2%)
      - Appropriate abstract (2%)
      - No abstract (0%)
   2.2. Introduction (4%)
      - Excellent (4%)
      - Lack of necessary parts (1%-3%)
      - No introduction (0%)
   2.3. Methodology (6%)
      - Excellent methodology: sufficient and accurate figures and text description (6%)
      - Reasonable methodology: clear figures and text description (3%-5%)
      - Incomplete methodology (1%-2%)
      - No methodology (0%)
   2.4. Implementation (7%)
      - Excellent implementation: sufficient steps of the implementation with proper figures or charts (7%)
      - Reasonable methodology: clear steps of the implementation with figures or charts (4%-6%)
      - Incomplete implementation (1%-3%)
      - No implementation (0%)
   2.5. Testing and results (8%)
      - Excellent: sufficient testing plan for different cases, sufficient results to show the performance with proper analysis (7%-8%)
      - Acceptable: clear testing plan, clear results to show the performance with analysis (3%-6%)
      - Lack of testing plan, results or analysis (1%-2%)
      - No testing and results (0%)
   2.6. Conclusion and reference (3%)
      - Excellent conclusion and reference with the correct format (2%-3%)
      - Acceptable conclusion and reference (1%)
      - No conclusion or incorrect reference (0%)
3. Format and language (5%)
   3.1. Report style and format (3%)
      - Beautiful and clear typography: 3%
      - Acceptable typography: 2%
      - Bad typography: 0% ~ 1%
   3.2. Language (2%)
      - Accurate and concise language: 2%
      - Unclear and confusing language: 0% ~ 1%

- ***Code (60%)***

Code running environment:

- The same virtual machine as CW1;
- You have to test your app using the provided virtual machine without any modification;
- The app that cannot be executed properly in this virtual machine will be marked as zero.

Code testing steps:

1. Your app (you have only one app with a single python file or multiple python files) will be run on only one virtual machine VM;
2. Your app will be started in different terminals (run together, but maybe not start together):

```
python3 main.py --node <node_1>
python3 main.py --node <node_2>
python3 main.py --node <node_3>
python3 main.py --node <node_4>
python3 main.py --node <node_5>
……
python3 main.py --node <node_n>
```

```
An example:
Terminal1> python3 main.py --node u
Terminal2> python3 main.py --node v
Terminal3> python3 main.py --node w
Terminal4> python3 main.py --node x
```

3. Your apps can run without any error (RUN)
4. Each of your apps should be running until getting the final result and write to the corresponding json file (eg. u_output.json …) (RES = [RES1, RES2…RESn[4]]).

Marking Criteria	**Item for testing**	**Mark**
Phase 1 (20%)	RUN	20%
If you cannot pass Phase 1, your app will not go to the testing of the next phase.		
Phase 2 (30%)	30% / n * len(RES)	30%
Coding style (10%)	Comments / readability	10%

[4] No more than 10 in this test.