

COSC364 Assignment 1

Report

Canying Liu(47108377)

JIA HUA XIE CAO(87728076)

-A percentage contribution of each partner: 50% to 50%

-Which aspects of your overall program do you consider particularly well done?

One of the well-done parts we consider is the Packet class. As the routers are sending packets throughout the whole test, we thought using strings to represent packets would work but tedious. This is because we have to read the packets manually every single time we receive packets. However, we adopt to use a class, this makes more sense and saves us a lot of work. For example, we can initialize a packet by simply calling Packet() function and we can generate different types of packets by calling create_triggered_update_pkt and create_periodic_update_pkt functions in class Packet once the packet types are decided. Instead of creating it manually, we can just let the class do it for us.

The other aspect that we have done well is we have many functions perform syntax and consistency checks. In the Packet class, we have is_correct_rte_format to make sure the route entry is 20 bytes and values in the fixed fields are correct. The is_correct_pkt_format function checks whether the packet router receives is valid. Also, the functions is_valid_id, is_valid_port, and is_valid_cost ensure that we can create routers correctly. These checking functions are simple but they are very important roles in our code, as they can certainly reduce the bugs in the code.

-Which aspects of your overall program could be improved?

One thing I think we can improve is to use an Entry_table class to represent the entry table of routers. Now we are using a dictionary to store all the entries. This is not convenient because each time we look for entry from it, we have to put keys to search. And it is also for the coding standard. Entries are stored as an Entry class into a Entry_table class rather than a dictionary.

-How have you ensured atomicity of event processing?

After setting up all the routers, each of them will send a periodic update packet to its peers. And they record the time as sent_time, then they wait for packets to come from their peers. Whenever a router receives a packet from its peer, it will create entries from the packet and update its table by calling update_table function. In the update_table function, it will update the table and return whether it has an update if it has a better route to the destination or the destination is unreachable anymore. If it has updates, then it will send a triggered update packet to neighbor routers. And other routers will do the same things. After time periodic_update, they will send a periodic update packet again. Then every router has its complete table. If we close a router, it doesn't change anything at the beginning. But its peer router doesn't receive any periodic update from it, they start counting the GARBAGE_COLLECTION_PERIOD. Once it reaches 0, they will delete information about the closed router and send the triggered update packet to neighbor routers.

-weaknesses of the RIP routing protocol?

From this project, we know that the weakness of the RIP is very obvious and serious. As the maximum hop count of RIP is only 15, so it can not be used in a large network. Also, routers in RIP only communicate with its neighbors, and the entry doesn't have any path, therefore if

the destination is not that router's neighbor, it is very easy to have a routing loop even though we implement split-horizon with poisoned reverse to prevent it from happening. Finally, in RIP, routers send packets periodically, it will use lots of network sources and increase the network traffic.

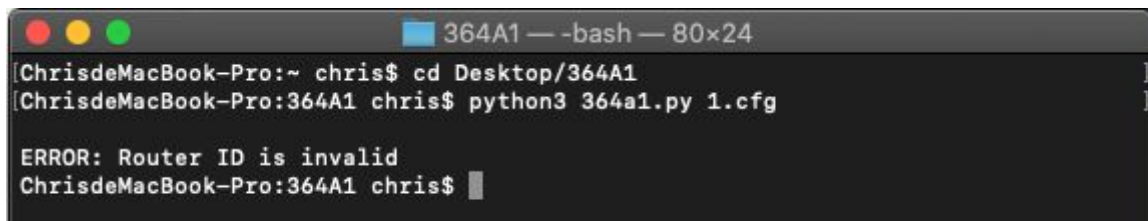
Configuration file tests

Case 1: testing on router-id

Set the router-id in the configuration file to 0, 64001, "string", and None respectively.

Expected: Print an error message and quit the program.

Actual:

A terminal window titled '364A1 — -bash — 80x24' shows the following commands and output:

```
ChrisdeMacBook-Pro:~ chris$ cd Desktop/364A1  
ChrisdeMacBook-Pro:364A1 chris$ python3 364a1.py 1.cfg  
  
ERROR: Router ID is invalid  
ChrisdeMacBook-Pro:364A1 chris$
```

Set the router-id to a value between 1 and 64000 inclusive.

Expected: The router-id is successfully read and the program moves on.

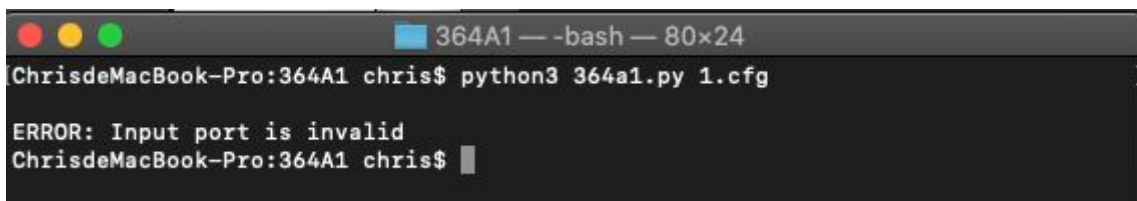
Actual: as expected

Case 2: testing on input ports

Set one of the input ports to 1023, 64001, "string", and None respectively.

Expected: Print an error message and quit the program.

Actual:

A terminal window titled '364A1 — -bash — 80x24' shows the following commands and output:

```
ChrisdeMacBook-Pro:364A1 chris$ python3 364a1.py 1.cfg  
  
ERROR: Input port is invalid  
ChrisdeMacBook-Pro:364A1 chris$
```

Set the input ports to values between 1024 and 64000 inclusive.

Expected: Input ports are successfully read and the program moves on.

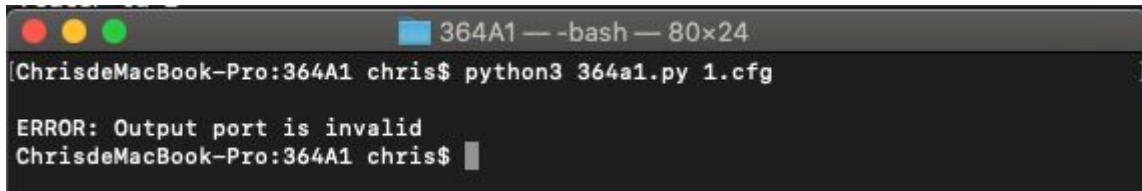
Actual: as expected

Case 3: testing on output ports

Set one of the output ports to 1023, 64001, "string", and None respectively.

Expected: Print an error message and quit the program.

Actual:

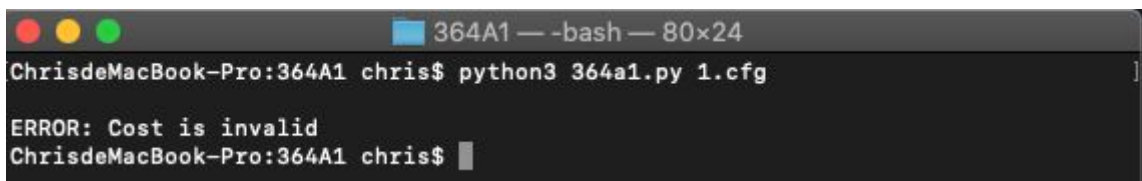


```
364A1 — -bash — 80x24
[ChrisdeMacBook-Pro:364A1 chris$ python3 364a1.py 1.cfg ]
ERROR: Output port is invalid
ChrisdeMacBook-Pro:364A1 chris$
```

Set matric value to 0, 17, "string", and None respectively.

Expected: Print an error message and quit the program.

Actual:

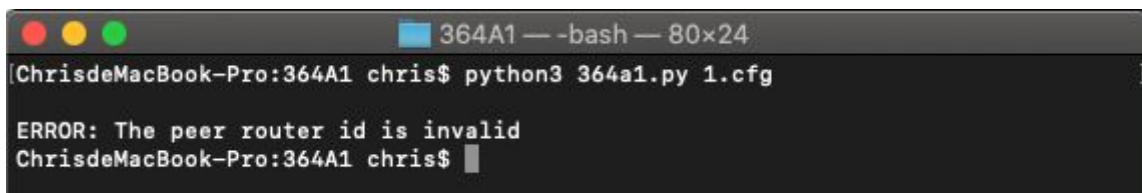


```
364A1 — -bash — 80x24
ChrisdeMacBook-Pro:364A1 chris$ python3 364a1.py 1.cfg ]
ERROR: Cost is invalid
ChrisdeMacBook-Pro:364A1 chris$
```

Set the router-id in outputs to 0, 64001, "string", and None respectively.

Expected: Print an error message and quit the program.

Actual:

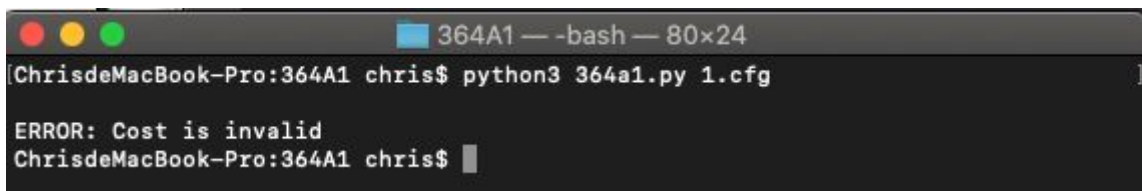


```
364A1 — -bash — 80x24
[ChrisdeMacBook-Pro:364A1 chris$ python3 364a1.py 1.cfg ]
ERROR: The peer router id is invalid
ChrisdeMacBook-Pro:364A1 chris$
```

Set outputs with missing output port/matric value/peer router-id

Expected: Print a message indicating an error in port/matric/router-id parameter correspondingly and quit the program.

Actual: (an example of missing matric value)



```
364A1 — -bash — 80x24
[ChrisdeMacBook-Pro:364A1 chris$ python3 364a1.py 1.cfg ]
ERROR: Cost is invalid
ChrisdeMacBook-Pro:364A1 chris$
```

Set outputs in the correct format(port-cost-peer) with no missing parameters, where $1024 \leq \text{port} \leq 64000$ and $1 \leq \text{cost} \leq 16$ and $1 \leq \text{peer} \leq 64000$. All parameters are of type int.

Expected: Outputs are successfully read and the program moves on.

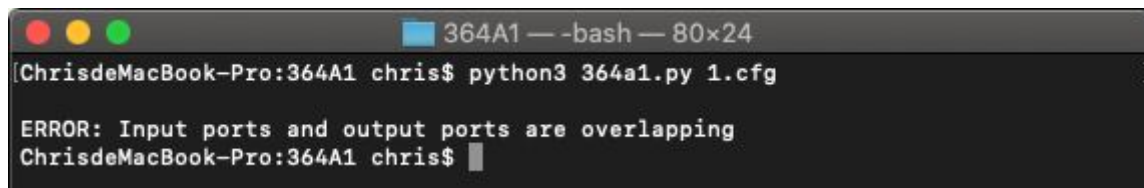
Actual: as expected

Case 4: testing on input/output ports overlap

Set the input port and the output port to the same value.

Expected: Print an error message and quit the program

Actual:



```
ChrisdeMacBook-Pro:364A1 chris$ python3 364a1.py 1.cfg
ERROR: Input ports and output ports are overlapping
ChrisdeMacBook-Pro:364A1 chris$
```

Set port numbers of input ports and output ports to unique values with no duplicates.

Expected: Both input ports and output ports are successfully read and the program moves on.

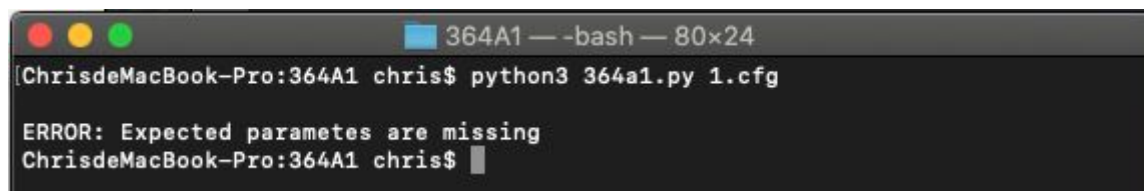
Actual: as expected

Case 5: testing on mandatory parameters

Parse a configuration file which has no router-id/input ports/outputs parameter respectively.

Expected: The program stops with an error message.

Actual:

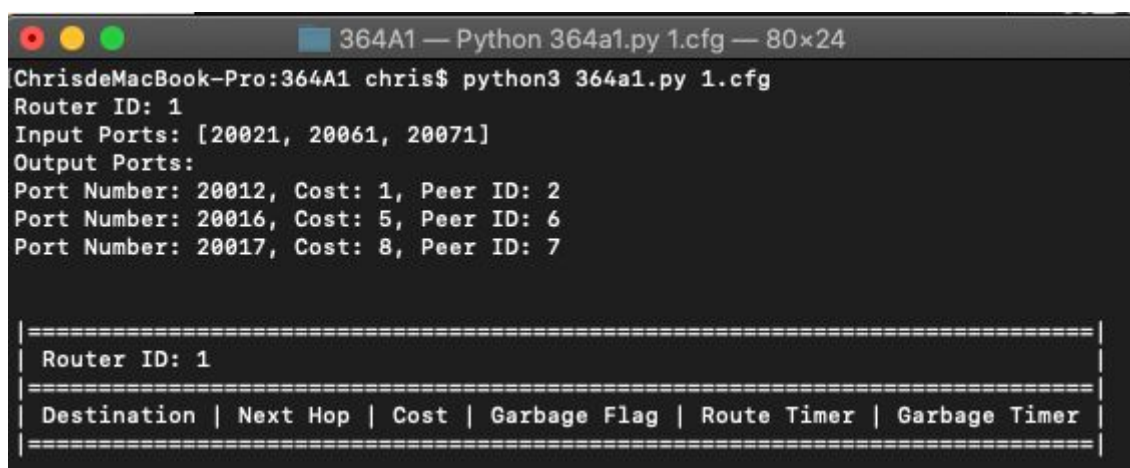


```
ChrisdeMacBook-Pro:364A1 chris$ python3 364a1.py 1.cfg
ERROR: Expected parametes are missing
ChrisdeMacBook-Pro:364A1 chris$
```

Parse a configuration file which includes mandatory parameters, empty lines, and comments. All values of parameters are in their allowed range.

Expected: The configuration file successfully parsed, and a router is established based on the given information.

Actual:



```
ChrisdeMacBook-Pro:364A1 chris$ python3 364a1.py 1.cfg
Router ID: 1
Input Ports: [20021, 20061, 20071]
Output Ports:
Port Number: 20012, Cost: 1, Peer ID: 2
Port Number: 20016, Cost: 5, Peer ID: 6
Port Number: 20017, Cost: 8, Peer ID: 7

=====
| Router ID: 1 |
=====
| Destination | Next Hop | Cost | Garbage Flag | Route Timer | Garbage Timer |
=====
```

Conclusion:

Test cases above ensure that the data used for initialising a router is correctly parsed from the given configuration file, and the setup_router function works as expected. If the

configuration file does not include all mandatory parameters or values of parameters are not in allowed range, the program will stop with an error message.

RIP packet tests

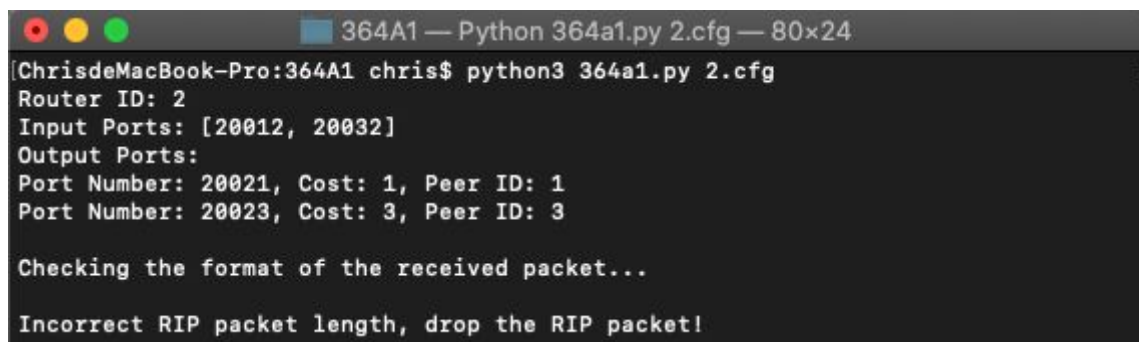
A rip packet is comprised of a 4 bytes header(command(1), version(1), originated router-id(2)) and multiple 20 bytes route entries(AFI(2), route flag(2), destination(4), zeros(8), metric(4)). Therefore, the length of the rip packet is $4 + 20 * n$ bytes where n is the number of route entries. There are several fields with a fixed value. The command, version, and AFI are always 2. The route flag is set to either 0 or 1. Values in the destination field and the metric field are expected to be within the valid scope.

Case 1: testing on packet length

Start two adjacent routers and set the self.header in class Packet to 3 bytes in router 1.

Expected: After printing an error message, router 2 drops the packet.

Actual:

A terminal window titled "364A1 — Python 364a1.py 2.cfg — 80x24" showing the output of a Python script. The script prints configuration for Router ID: 2, including input and output ports. It then checks the format of a received packet and reports an error: "Incorrect RIP packet length, drop the RIP packet!".

```
[ChrisdeMacBook-Pro:364A1 chris$ python3 364a1.py 2.cfg]
Router ID: 2
Input Ports: [20012, 20032]
Output Ports:
Port Number: 20021, Cost: 1, Peer ID: 1
Port Number: 20023, Cost: 3, Peer ID: 3

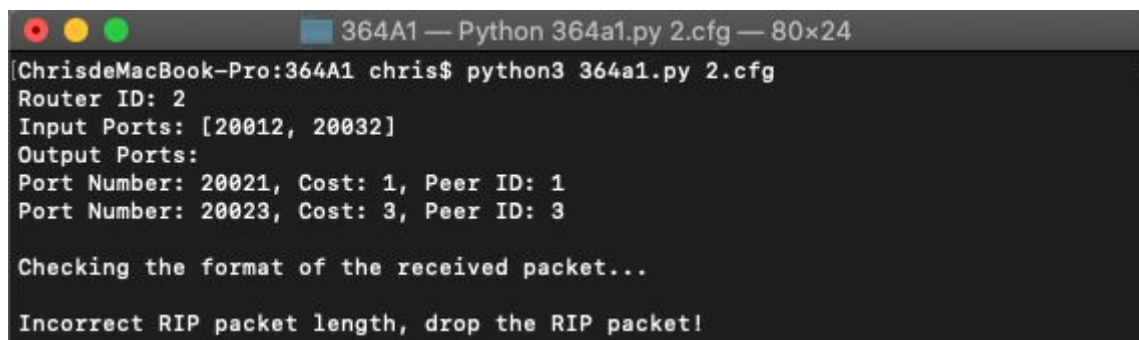
Checking the format of the received packet...

Incorrect RIP packet length, drop the RIP packet!
```

Start two adjacent routers and set the self.rtes in class Packet to 19 bytes in router 1.

Expected: After printing an error message, router 2 drops the packet.

Actual:

A terminal window titled "364A1 — Python 364a1.py 2.cfg — 80x24" showing the output of a Python script. The script prints configuration for Router ID: 2, including input and output ports. It then checks the format of a received packet and reports an error: "Incorrect RIP packet length, drop the RIP packet!".

```
[ChrisdeMacBook-Pro:364A1 chris$ python3 364a1.py 2.cfg]
Router ID: 2
Input Ports: [20012, 20032]
Output Ports:
Port Number: 20021, Cost: 1, Peer ID: 1
Port Number: 20023, Cost: 3, Peer ID: 3

Checking the format of the received packet...

Incorrect RIP packet length, drop the RIP packet!
```

Case 2: testing on fixed fields

Start two adjacent routers and set the COMMAND to 1 in router 1.

Expected: After printing an error message, router 2 drops the packet.

Actual:

```
Checking the format of the received packet...  
Incorrect value in COMMAND field, drop the RIP packet!
```

Start two adjacent routers and set the VERSION to 1 in router 1.
Expected: After printing an error message, router 2 drops the packet.
Actual:

```
Checking the format of the received packet...  
Incorrect value in VERSION field, drop the RIP packet!
```

Start two adjacent routers and set the AFI in a router entry to 1 in router 1.
Expected: After printing an error message, router 2 drops the packet.
Actual:

```
Checking the format of the received packet...  
Incorrect value in AFI field, drop the RIP packet!
```

Start two adjacent routers and set the ROUTE FLAG in a router entry to 3 in router 1.
Expected: After printing an error message, router 2 drops the packet.
Actual:

```
Checking the format of the received packet...  
Incorrect value in ROUTE FLAG field, drop the RIP packet!
```

Start two adjacent routers and set a value in MUST-BE-ZEROS field to non-zero number
Expected: After printing an error message, router 2 drops the packet.
Actual:

```
Checking the format of the received packet...  
Incorrect value in MUST BE ZERO field, drop the RIP packet!
```

Start two adjacent routers and set the MATRIC in a router entry to 17 in router 1.
Expected: After printing an error message, router 2 drops the packet.
Actual:

```
Checking the format of the received packet...  
Incorrect value in MATRIC field, drop the RIP packet!
```

Conclusion:

Test cases above provide details about how basic syntax and consistency checks are performed on an incoming rip packet after it is received and before it is read. If there is an

error found in any fields of the packet, the whole packet is dropped with an error message. Further details could be found in `is_correct_pkt_format()` method in class `Packet`.

Periodic update tests(including split-horizon with poisoned reverse tests)

Case 1: testing on generating periodic updates

related source code: `send_msg` function in class `Router` and `create_periodic_update` function in class `Packet`

Start a router.

Expected: Periodic update packets are created separately for each neighbor.

Actual:

```
Creating a periodic update for neighbor 2...
Send update message to neighbor 2...

Creating a periodic update for neighbor 6...
Send update message to neighbor 6...

Creating a periodic update for neighbor 7...
Send update message to neighbor 7...
```

Start a router.

Expected: Periodic updates are sent periodically according to the `UPDATE_PERIOD`.

Actual:

```
364A1 — Python 364a1.py 1.cfg — 80x27

Creating a periodic update for neighbor 2...
Send update message to neighbor 2...

Creating a periodic update for neighbor 6...
Send update message to neighbor 6...

Creating a periodic update for neighbor 7...
Send update message to neighbor 7...

=====
| Router ID: 1 |
=====
| Destination | Next Hop | Cost | Garbage Flag | Route Timer | Garbage Timer |
=====
Creating a periodic update for neighbor 2...
Send update message to neighbor 2...

Creating a periodic update for neighbor 6...
Send update message to neighbor 6...

Creating a periodic update for neighbor 7...
Send update message to neighbor 7...
```

Start three routers. Router 2 is an intermediate router that connects router 1 and router 3. There is no direct link between router 1 and router 3.

Expected: Router 1 creates periodic updates including an entry to itself with a cost of 0 and all entries in its routing table. In particular, a packet generated for router 2 has an entry to

Actual:

```
=====
Router ID: 1
=====
Destination | Next Hop | Cost | Garbage Flag | Route Timer | Garbage Timer
=====
Reading packets from Router 2...

=====
Router ID: 1
=====
Destination | Next Hop | Cost | Garbage Flag | Route Timer | Garbage Timer
=====
2           | 2         | 1    | 0             | 29          | 20
```

```

=====
Router ID: 2
=====
Destination | Next Hop | Cost | Garbage Flag | Route Timer | Garbage Timer
=====

```

Reading packets from Router 1...

```

=====
Router ID: 2
=====
Destination | Next Hop | Cost | Garbage Flag | Route Timer | Garbage Timer
=====

```

Start router 1, router 2, and finally router 3. Each of them has a direct link to others of cost 1. Expected: If the packet received from router 1 is read first, the routing table of router 3 should have an entry to destination 2 with a cost of 2 and a next hop of 1. After reading the packet from router 2, the route to destination 2 should have a cost of 1 and a next hop of 2. Actual:

```

Reading packets from Router 1...
=====
Router ID: 3
=====
Destination | Next Hop | Cost | Garbage Flag | Route Timer | Garbage Timer
=====
1 | 1 | 1 | 0 | 29 | 20
=====

```

```

=====
Router ID: 3
=====
Destination | Next Hop | Cost | Garbage Flag | Route Timer | Garbage Timer
=====
1 | 1 | 1 | 0 | 29 | 20
2 | 1 | 2 | 0 | 29 | 20
=====

```

Reading packets from Router 2...

```

=====
Router ID: 3
=====
Destination | Next Hop | Cost | Garbage Flag | Route Timer | Garbage Timer
=====
1 | 1 | 1 | 0 | 26 | 20
2 | 2 | 1 | 0 | 29 | 20
=====

```

Conclusion:

Test cases above show that periodic update packets are generated separately for each neighbors and the split-horizon with poisoned reverse is implemented to the packets generating function to avoid loops. Entries in routing tables are updated in some cases shown above and the read_msg function works as expected.

Triggered update tests(including timer events tests)

related source code: send_msg function and check_entries function in class Router and create_triggered_update function in class Packet

Case 1: detect link failure and generate triggered updates

Start three routers. Router 2 is an intermediate router that connects router 1 and router 3. There is no direct link between router 1 and router 3. After 20 seconds, turn off router 1.

Expected: Router 2 makes changes on its routing table entry of destination 1 and generates triggered updates. When the route timer of the entry reaches 0, router 2 sets the entry cost to 16, changes the garbage flag of the entry to 1, and starts a garbage timer. After modifying its routing table, router 2 generates triggered updates including only the entry of destination 1. The route entry in the triggered update packets has a route flag of 1 and a cost of 16. The created packets are then sent to its neighbors. Router 2 will start garbaging the entry of destination 1 if the garbage timer reaches 0.

Actual:

```

=====
Router ID: 2
=====
Destination | Next Hop | Cost | Garbage Flag | Route Timer | Garbage Timer
=====
1 | 1 | 1 | 0 | 5 | 20
3 | 3 | 3 | 0 | 26 | 20
=====

Router ID: 2
=====
Destination | Next Hop | Cost | Garbage Flag | Route Timer | Garbage Timer
=====
1 | 1 | 1 | 0 | 0 | 20
3 | 3 | 3 | 0 | 26 | 20
=====

Router ID: 2
=====
Destination | Next Hop | Cost | Garbage Flag | Route Timer | Garbage Timer
=====
1 | 1 | 16 | 1 | 0 | 19
3 | 3 | 3 | 0 | 26 | 20
=====
Generating triggered packet for router 1...
Sending the packet to router 1
Generating triggered packet for router 3...
Sending the packet to router 3

```

```

=====
Router ID: 2
=====
Destination | Next Hop | Cost | Garbage Flag | Route Timer | Garbage Timer
=====
1 | 1 | 16 | 1 | 0 | 1
3 | 3 | 3 | 0 | 26 | 20
=====

Router ID: 2
=====
Destination | Next Hop | Cost | Garbage Flag | Route Timer | Garbage Timer
=====
1 | 1 | 16 | 1 | 0 | 0
3 | 3 | 3 | 0 | 26 | 20
=====
Deleting Entry of destination 1 from the routing table...

Router ID: 2
=====
Destination | Next Hop | Cost | Garbage Flag | Route Timer | Garbage Timer
=====
3 | 3 | 3 | 0 | 26 | 20
=====

```

Case 2: receive triggered update from a neighbor, update own routing table, and generate triggered updates for other neighbors

Start three routers. Router 2 is an intermediate router that connects router 1 and router 3. There is no direct link between router 1 and router 3. After 20 seconds, turn off router 1.

Expected: After receiving the triggered update packet from router 2, router 3 changes the cost of its routing table entry of destination 1 to 16, and sets a garbage timer directly without waiting for route timeout. Then, it generates triggered updates for its neighbors except router 2. Router 3 will garbage the entry of destination 1 if the garbage timer reaches 0.

Actual:

```

=====
Router ID: 3
=====
Destination | Next Hop | Cost | Garbage Flag | Route Timer | Garbage Timer
=====
1 | 2 | 4 | 0 | 26 | 20
2 | 2 | 3 | 0 | 26 | 20
=====

Generating triggered packet for router 4...
Sending the packet to router 4

Router ID: 3
=====
Destination | Next Hop | Cost | Garbage Flag | Route Timer | Garbage Timer
=====
1 | 2 | 16 | 1 | 26 | 16
2 | 2 | 3 | 0 | 26 | 20
=====

```

```

=====
Router ID: 3
=====
Destination | Next Hop | Cost | Garbage Flag | Route Timer | Garbage Timer
=====
1 | 2 | 16 | 1 | 8 | 0
2 | 2 | 3 | 0 | 26 | 20
=====

Deleting Entry of destination 1 from the routing table...

Router ID: 3
=====
Destination | Next Hop | Cost | Garbage Flag | Route Timer | Garbage Timer
=====
2 | 2 | 3 | 0 | 23 | 20
=====

```

Case 3: receive triggered update from a neighbor after the failure link has detected already

Start router 1, router 2, and finally router 3. Each of them has a direct link to others of cost 1. After 20 seconds, turn off router 2.

Expect: Router 3 detects the link failure between itself and router 2, and then modifies its routing table and generates triggered updates for all neighbors. After sending the created triggered update packets, it receives the triggered update from router 1. Router 3 will ignore the triggered update sent by router 1 and continue its garbage timing.

Actual:

```
=====
Router ID: 3
=====
Destination | Next Hop | Cost | Garbage Flag | Route Timer | Garbage Timer
=====
1           | 1        | 1    | 0            | 26         | 20
=====
2           | 2        | 16   | 1            | 0          | 19
=====

Reading packets from Router 1...
=====
Router ID: 3
=====
Destination | Next Hop | Cost | Garbage Flag | Route Timer | Garbage Timer
=====
1           | 1        | 1    | 0            | 26         | 20
=====
2           | 2        | 16   | 1            | 0          | 13
=====
```

Case 4: receive triggered update from a neighbor which is not the next-hop of the failed entry

Start four routers. Router 1 is connected to router 2 and router 3 with a cost of 1. Router 4 has a direct link to router 2 of cost 1, and router 4 is also attached to router 3 with a cost of 2. Wait for 30 seconds and switch off router 4.

Expected: Router 2 and router 3 detect link failure and send triggered updates. If the triggered update generated by router 3 is received by router 1, router 1 will drop the packet as the next hop of its entry to destination 4 is not router 3. Router 1 will not update its routing table until it receives the triggered update form router 2.

Actual:

```
Reading packets from Router 3...
=====
Router ID: 1
=====
Destination | Next Hop | Cost | Garbage Flag | Route Timer | Garbage Timer
=====
2           | 2        | 1    | 0            | 26         | 20
=====
3           | 3        | 1    | 0            | 26         | 20
=====
4           | 2        | 2    | 0            | 26         | 20
=====

Reading packets from Router 2...
Generating triggered packet for router 3...
Sending the packet to router 3
```

Conclusion:

Triggered update packets are generated when routers detect link failures or when routers receive triggered updates relevant to themselves. Route Flag filed in Route Entry is used to indicate if a route is experiencing a timeout. An entry with cost 16 and route flag 0 is considered as a destination is not reachable through a particular router, but a cost of 16 and a route flag of 1 represents the originated router has started its garbage timer. According to the test cases shown above, functions implemented to support triggered update events work as expected.

Timer event tests

Case 1: restart the router before the route timeout

Start three routers. Each of them has a direct link to others of cost 1. Wait for 20 seconds and turn off router 2. Before the route timeout, restart router 2.

Expected: Route timers in entries of destination 2 keep dropping and will be set to ROUTE_TIMEOUT after router 2 restarting.

Actual:

Router ID: 1						
Destination	Next Hop	Cost	Garbage Flag	Route Timer	Garbage Timer	
2	2	1	0	11	20	
3	3	1	0	26	20	

Router ID: 1						
Destination	Next Hop	Cost	Garbage Flag	Route Timer	Garbage Timer	
2	2	1	0	29	20	
3	3	1	0	27	20	

Router ID: 3						
Destination	Next Hop	Cost	Garbage Flag	Route Timer	Garbage Timer	
1	1	1	0	26	20	
2	2	1	0	8	20	

Router ID: 3						
Destination	Next Hop	Cost	Garbage Flag	Route Timer	Garbage Timer	
1	1	1	0	26	20	
2	2	1	0	26	20	

Case 2: restart the router before the garbage timeout

Start three routers. Each of them has a direct link to others of cost 1. Wait until both garbage flags in entries of destination 2 are changed and the garbage timers start counting, switch off router 2. Restart the router 2 before the garbage timers get down to 0.

Expected:

When the route timer of destination 2 entry reaches 0, router 3 changes the cost of entry to 16. It also sets the garbage flag and starts the garbage timer. After router 2 is switched on, route timer and garbage timer are changed to their initial value. The garbage flag and cost of the entry will change as well.

Actual:

Router ID: 3						
Destination	Next Hop	Cost	Garbage Flag	Route Timer	Garbage Timer	
1	1	1	0	26	20	
2	2	16	1	0	13	

Router ID: 3						
Destination	Next Hop	Cost	Garbage Flag	Route Timer	Garbage Timer	
1	1	1	0	26	20	
2	2	1	0	26	20	

Case 3: restart the router after deleting the entry

Start three routers. Each of them has a direct link to others of cost 1. After 20 seconds, turn off router 2. Wait until the entries of destination 2 are removed from routing tables, switch on router 2.

Expected: The entry of destination 2 is added to the routing table as a new entry.

Actual:

Router ID: 1						
Destination	Next Hop	Cost	Garbage Flag	Route Timer	Garbage Timer	
2	2	16	1	0	1	
3	3	1	0	26	20	

Router ID: 1						
Destination	Next Hop	Cost	Garbage Flag	Route Timer	Garbage Timer	
2	2	16	1	0	0	
3	3	1	0	26	20	

Deleting Entry of destination 2 from the routing table...

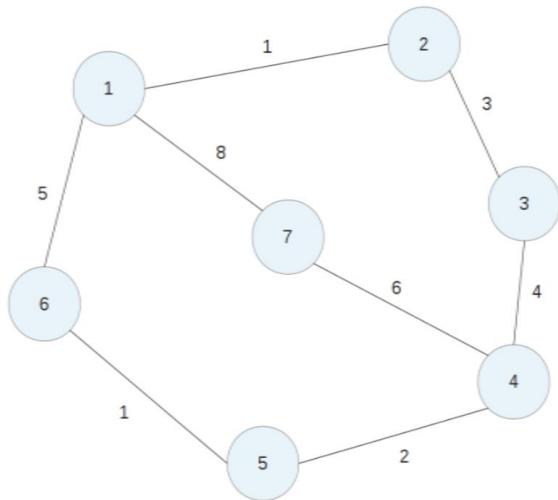
Router ID: 1						
Destination	Next Hop	Cost	Garbage Flag	Route Timer	Garbage Timer	
3	3	1	0	26	20	

Router ID: 1						
Destination	Next Hop	Cost	Garbage Flag	Route Timer	Garbage Timer	
2	2	1	0	26	20	
3	3	1	0	26	20	

Conclusion:

Timers are vital for ensuring atomicity of event processing. It is responsible for when to generate triggered updates and when to garbage entries in routing tables. Test cases above show how timers are used to achieve atomicity of event processing in the program.

Configuration files



router-id 1
input-ports 20021 20061 20071
outputs 20012-1-2 20016-5-6 20017-8-7

router-id 2
input-ports 20012 20032
outputs 20021-1-1 20023-3-3

router-id 3
input-ports 20023 20043
outputs 20032-3-2 20034-4-4

router-id 4
input-ports 20034 20054 20074
outputs 20043-4-3 20045-2-5 20047-6-7

router-id 5
input-ports 20045 20065
outputs 20054-2-4 20056-1-6

router-id 6
input-ports 20016 20056
outputs 20061-5-1 20065-1-5

router-id 7
input-ports 20017 20047
outputs 20071-8-1 20074-6-4