Solution Review: Routing Information Protocol

In this lesson, we'll analyze the solution to the RIP challenge.

WE'LL COVER THE FOLLOWING

Solution

main.py

port.py

rip_packet.py

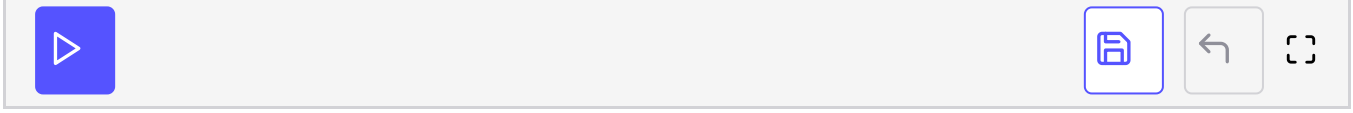
router.py

topology_reader.py

```
from rip_packet import RIP_entry
from rip_packet import RIP_packet
from port import port
from port import port_link

class router_base:
  def __init__(self, IP_address, rip_entries, ports):
    self.IP_address = IP_address
    self.rip_entries = rip_entries
    self.ports = ports

  def add_port(self, prt):
    self.ports.append(prt)

  def add_RIP_entry(self, port_IP, dest_IP, cost, next_hop_IP):
    new_rip_entry = RIP_entry(port_IP, cost, dest_IP, next_hop_IP)
    self.rip_entries.append(new_rip_entry)

  def find_RIP_entry(self, destination_IP_to_find):
```

```
  def find_RIP_entry(self, destination_IP_to_find):
    for entry in self.rip_entries:
      if(entry.dest_IP_address == destination_IP_to_find):
        return entry
    return None

  def set_RIP_entry_cost(self, destination_IP_to_find, new_cost):
    for i in self.rip_entries:
      if(entry.dest_IP_address == destination_IP_to_find):
        entry.set_cost(new_cost)
    return None

  def delete_RIP_entry(self, destination_IP_to_find):
    for entry in self.rip_entries:
      if(entry.dest_IP_address == destination_IP_to_find):
        self.rip_entries.remove(entry)

  def print_router(self):
    print("~~~~ Router IP address = " + str(self.IP_address) + "~~~~")
    print("---Ports---")
    print("Port IP | Destination Router IP | Destination Port IP | Cost")
    for p in self.ports:
      p.print_port()
    print("---RIP entries---")
    print("port IP | destination IP address | next hop | cost")
    for re in self.rip_entries:
      re.print_rip_entry()

  def return_router(self):
    r =[]
    r.append("~~~~ Router IP address = " + str(self.IP_address) + "~~~~")
    r.append("---Ports---")
    r.append("Port IP | Destination Router IP | Destination Port IP | Cost")
    for p in self.ports:
      r.append(p.return_port())
    r.append("---RIP entries---")
    r.append("port IP | destination IP address | next hop | cost")
    for re in self.rip_entries:
      r.append(re.return_rip_entry())
    return r

class router(router_base):
  def send_RIP_packets(self, routers):
    self.rip_packet = RIP_packet(self.rip_entries)
    for i in range(len(routers)): # Find all neighbors and send then RIP packet
      for j in range(len(self.ports)):
        if routers[i].IP_address == self.ports[j].link.dest_IP_address:
          routers = routers[i].receive_RIP_packets(self.rip_packet, self.ports[j].link, rout
    return routers

  def receive_RIP_packets(self, rip_packet, link_send_on, routers, next_hop_IP):
    found = 0
    new_rip_entries = []
    for i in range(len(rip_packet.rip_entries)):
      for j in range(len(self.rip_entries)):
        if(rip_packet.rip_entries[i].dest_IP_address == self.rip_entries[j].dest_IP_address)
          found = 1
          new_cost = link_send_on.cost + rip_packet.rip_entries[i].cost
          if(self.rip_entries[j].next_hop_IP == next_hop_IP and self.rip_entries[j].cost !=
            self.rip_entries[j].cost = min(16, new_cost)
            break
          elif(new_cost < self.rip_entries[j].cost):
              self.rip_entries[j].cost = new_cost
```

```
            self.rip_entries[j].next_hop_IP = next_hop_IP
            break
      # If entry does not already exist
      if(not found):
        new_rip_entries.append(RIP_entry(link_send_on.dest_port_IP, rip_packet.rip_entries[i
      found = 0
    self.rip_entries.extend(new_rip_entries)
    return routers
```



Explanation

`send_RIP_packets()`

Let's go through the solution line-by-line:

- **line 62**: We create the RIP packet that we'll send to all of our neighbors on this line. We pass the `rip_entries` list and the length of that list on this line.
- **lines 63-66**: we now find the neighbors of this router. We do this by iterating over the given list of routers in the network and checking to see if any of our ports have a link to them. We do this by iterating over our ports and checking each port's link's destination IP address against the router's IP address. If they match, the router is a neighbor and we send it our RIP packet by calling `receive_RIP_packets()` on it.

Finally, the routers list is returned.

`receive_RIP_packets()`

This function consists of the core of the Routing Information Protocol. It works as follows:

1. The receiving router checks for all RIP entries in the received RIP packet if they exist in its own RIP entries list.
2. If an entry is for a destination that's **not found** in the receiving router's RIP entries list, it adds it as done on **lines 85-88**.
3. If an entry is for a destination that **is found** in the receiving router's RIP entries list, it does one of two things:

1. If the entry is from a router whose IP address **is equal** to the next hop IP in the router's current RIP entry **and** the cost has changed, it simply sets the cost as the minimum of 16 and the new cost. Note that the new cost can be greater or lesser than the current cost.
2. Otherwise, if the cost advertised in this RIP entry is lesser than the one the router currently has, it sets its RIP entry to the one through the router that sent the RIP entry.

Finally, the routers list is returned.