```python
"""
Assignment 1: RIP protocol
Team: Bach Vu (25082165), Charlie Hunter (27380476)
Router main program/router.py
"""
from timer import RTimer
from daemon_sup import strCurrTime

class Router:
    EXPIRED_UPDATE = "expired"
    REGULAR_UPDATE = "periodic"
    FAST_ROUTE_UPDATE = "Poison enhance"
    def __init__(self, rID, inputs, outputs, startTime, timeout):
        _timeout = timeout if timeout is not None else 5
        self.timer = RTimer(_timeout)
        self._garbages = {} # (dest, time since expired)

        self.ROUTER_ID = rID
        self.INPUT_PORTS = inputs

        self._ROUTING_TABLE = {}  # {Dest: nxt Hop, metric, time, note}
        self._ROUTING_TABLE[rID] = ["-", 0, startTime, "Time Active"]

        self.OUTPUT_PORTS = {} # (dest, cost, port_to_send)
        for output in outputs:
            from_port, to_port, cost, dest = output.split('-')
            from_port, to_port, cost, dest = int(from_port), int(to_port), int(cost), int
(dest)
            self.OUTPUT_PORTS[dest] = (to_port, cost, from_port)

    def get_routing_table(self, dest, mode):
        entries = []
        for key, val in self._ROUTING_TABLE.items():
            if dest == val[0] or dest == key:
                # don't re-advertise info from a hop (Split horizon)
                # dest == key not needed, but can reduce packet size
                continue
            if mode == "expired" and val[1] != 16:
                # triggered update, contain expired entries only (pg29)
                continue
            if mode == "Poison enhance" and val[3] != "Shorter route":
                continue

            new_metric = val[1] + self.OUTPUT_PORTS[dest][1]
            if new_metric > 15 and val[1] != 16:
                continue
            new_metric = min(new_metric, 16)
            entries.append((key, self.ROUTER_ID, new_metric))
        return entries

    def update_route_table(self, routes, utime):
        update_flag = False
        for route in routes:
            dest, nxtHop, metric = route
            new_entry = [nxtHop, metric, utime.timestamp(), ""]
            exist_entry = self._ROUTING_TABLE.get(dest, None)

            if not self._need_update(new_entry, exist_entry):
```

```
58                 continue
59
60             self._ROUTING_TABLE[dest] = new_entry
61             if new_entry[3] == "Shorter route":
62                 # trigger update with small delay. Not needed for small delay network.
63                 update_flag = True
64
65             # updated dest entry could be in garbage collecting
66             self._garbages.pop(dest, None)
67         return update_flag
68
69     def _need_update(self, new_entry, exist_entry):
70         """ For fancy purpose of taking note when update an entry
71             return True if new entry is valid to be updated
72         """
73         if exist_entry is None:
74             if new_entry[1] == 16:
75                 # Don't worry about dead link of unknown dest
76                 return False
77             new_entry[3] = "New dest."
78         else:
79             if new_entry[1] < exist_entry[1]:
80                 new_entry[3] = "Shorter route"
81
82             elif new_entry[1] == 16:
83                 if exist_entry[1] == 16:
84                     # already receive this link dead
85                     return False
86                 elif exist_entry[0] != new_entry[0]:
87                     # link dead is not currently in route table
88                     return False
89                 # 1st time known dest (metric < 16) has dead link
90                 new_entry[3] = "Link dead."
91
92             elif new_entry[1] == exist_entry[1]:
93                 new_entry[3] = "Reset timer"
94                 if new_entry[0] != exist_entry[0]:
95                     # New route, reset timer still
96                     new_entry[3] = "Same cost"
97
98             else:
99                 # ["Slower route."], not update
100                return False
101
102        return True
103
104    def garbage_collection(self, gtime):
105        if not self.timer.is_expired(RTimer.GARBAGES_TIMEOUT, gtime):
106            return False
107
108        for item, time in self._garbages.copy().items():
109            if self.timer.is_expired(RTimer.GARBAGE_TIMEOUT, gtime, time):
110                self._ROUTING_TABLE.pop(item, None)
111                self._garbages.pop(item)
112                print(f"Removed dead link to {item} at {strCurrTime(gtime)}")
113        self.timer.reset_timer(RTimer.GARBAGES_TIMEOUT)
114
115    def has_expired_entry(self, etime):
```

```
116         if not self.timer.is_expired(RTimer.ENTRIES_TIMEOUT, etime):
117             """ Trigger once if multilink die in short period """
118             return False
119 
120         garbage_found = 0
121         for dest,entry in self._ROUTING_TABLE.items():
122             if dest == self.ROUTER_ID:
123                 continue
124 
125             _, metric, ttl, _ = entry
126             if metric == 16:
127                 if dest in self._garbages.keys():
128                     # Waiting to be removed, skip to avoid sending same info to network
129                     continue
130                 self._garbages[dest] = etime.timestamp()
131                 garbage_found += 1
132 
133             elif self.timer.is_expired(RTimer.ENTRY_TIMEOUT, etime, ttl):
134                 entry[1], entry[3] = 16, "No response."
135                 self._ROUTING_TABLE[dest][1] = 16 # set to infinity
136                 self._garbages[dest] = etime.timestamp()
137                 print(f"Found expired link to {dest} at {strCurrTime(etime)}")
138                 garbage_found += 1
139 
140         self.timer.reset_timer(RTimer.ENTRIES_TIMEOUT)
141         # print(garbage_found)
142         return garbage_found > 0
143 
144     def is_expected_sender(self, sender, receiver):
145         """ Avoid unwanted broadcast/malicious pecket """
146         for link in self.OUTPUT_PORTS.values():
147             if sender[1] == link[0] and receiver[1] == link[2]:
148                 return True
149         return False
150 
151     def print_hello(self):
152         print("-"*66)
153         print(f"Router {self.ROUTER_ID} is running ...")
154         print("Input ports:", self.INPUT_PORTS)
155         print("Output ports:")
156         for dest, link in self.OUTPUT_PORTS.items():
157             print(f"    {link} to Router ID {dest}")
158         print("-"*66)
159         print("Use Ctrl+C or Del to shutdown.")
160         print()
161 
162     def print_route_table(self, ptime):
163         if not self.timer.is_expired(RTimer.PRINT_TIMEOUT, ptime):
164             return
165 
166         print("="*66)
167         print("|{:16}--{} [{}]--{:16}|".format(" ", "ROUTING TABLE", strCurrTime(ptime
    ), " "))
168         print("|{:^10}|{:^10}|{:^10}|{:^10}|{:^20}|".format(
169             "Dest.", "Next Hop", "Metric", "Time (s)", "Notes"))
170         print("|" + "-"*64 + "|")
171         for dest, record in self._ROUTING_TABLE.items():
172             hop, cost, log_time, note = record
```

```
            duration = ptime.timestamp() - log_time
            print("|{:^10}|{:^10}|{:^10}|{:^10.3f}|{:^20}|".format(
                dest, hop, cost, duration, str(note)))
        print("="*66)
        self.timer.reset_timer(RTimer.PRINT_TIMEOUT)

    def reset_timer(self, mode):
        self.timer.reset_timer(mode)

    def is_expired(self, mode, curr_time):
        return self.timer.is_expired(mode, curr_time)


```