```python
"""
Assignment 1: RIP protocol
Team: Bach Vu (25082165), Charlie Hunter (27380476)
Router main program
"""
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
            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):
                continue

             self._ROUTING_TABLE[dest] = new_entry
             if new_entry[3] == "Shorter route":
                 # trigger update with small delay. Not needed for small delay network.
                 update_flag = True

             # updated dest entry could be in garbage collecting
             self._garbages.pop(dest, None)
         return update_flag

     def _need_update(self, new_entry, exist_entry):
         """ For fancy purpose of taking note when update an entry
             return True if new entry is valid to be updated
         """
         if exist_entry is None:
             if new_entry[1] == 16:
                 # Don't worry about dead link of unknown dest
                 return False
             new_entry[3] = "New dest."
         else:
             if new_entry[1] < exist_entry[1]:
                 new_entry[3] = "Shorter route"

             elif new_entry[1] == 16:
                 if exist_entry[1] == 16:
                     # already receive this link dead
                     return False
                 elif exist_entry[0] != new_entry[0]:
                     # link dead is not currently in route table
                     return False
                 # 1st time known dest (metric < 16) has dead link
                 new_entry[3] = "Link dead."

             elif new_entry[1] == exist_entry[1]:
                 new_entry[3] = "Reset timer"
                 if new_entry[0] != exist_entry[0]:
                     # New route, reset timer still
                     new_entry[3] = "Same cost"

             else:
                 # ["Slower route."], not update
                 return False

         return True

     def garbage_collection(self, gtime):
         if not self.timer.is_expired(RTimer.GARBAGES_TIMEOUT, gtime):
             return False

         for item, time in self._garbages.copy().items():
             if self.timer.is_expired(RTimer.GARBAGE_TIMEOUT, gtime, time):
                 self._ROUTING_TABLE.pop(item, None)
                 self._garbages.pop(item)
                 print(f"Removed dead link to {item} at {strCurrTime(gtime)}")
         self.timer.reset_timer(RTimer.GARBAGES_TIMEOUT)

     def has_expired_entry(self, etime):
         if not self.timer.is_expired(RTimer.ENTRIES_TIMEOUT, etime):
             """ Trigger once if multilink die in short period """
             return False

         garbage_found = 0
         for dest,entry in self._ROUTING_TABLE.items():
             if dest == self.ROUTER_ID:
                 continue

             _, metric, ttl, _ = entry
             if metric == 16:
                 if dest in self._garbages.keys():
                     # Waiting to be removed, skip to avoid sending same info to network
                     continue
                 self._garbages[dest] = etime.timestamp()
                 garbage_found += 1

             elif self.timer.is_expired(RTimer.ENTRY_TIMEOUT, etime, ttl):
                 entry[1], entry[3] = 16, "No response."
                 self._ROUTING_TABLE[dest][1] = 16 # set to infinity
                 self._garbages[dest] = etime.timestamp()
                 print(f"Found expired link to {dest} at {strCurrTime(etime)}")
                 garbage_found += 1

         self.timer.reset_timer(RTimer.ENTRIES_TIMEOUT)
         # print(garbage_found)
         return garbage_found > 0

     def is_expected_sender(self, sender, receiver):
         """ Avoid unwanted broadcast/malicious pecket """
         for link in self.OUTPUT_PORTS.values():
             if sender[1] == link[0] and receiver[1] == link[2]:
                 return True
         return False

     def print_hello(self):
         print("-"*66)
         print(f"Router {self.ROUTER_ID} is running ...")
         print("Input ports:", self.INPUT_PORTS)
         print("Output ports:")
         for dest, link in self.OUTPUT_PORTS.items():
             print(f"    {link} to Router ID {dest}")
         print("-"*66)
         print("Use Ctrl+C or Del to shutdown.")
         print()

     def print_route_table(self, ptime):
         if not self.timer.is_expired(RTimer.PRINT_TIMEOUT, ptime):
             return

         print("="*66)
         print("|{:16}--{} [{}]--{:16}|".format(" ", "ROUTING TABLE", strCurrTime(ptime
 ), " "))
         print("|{:^10}|{:^10}|{:^10}|{:^10}|{:^20}|".format(
             "Dest.", "Next Hop", "Metric", "Time (s)", "Notes"))
         print("|" + "-"*64 + "|")
         for dest, record in self._ROUTING_TABLE.items():
             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)


```