```
"""
Assignment 1: RIP protocol
Team: Bach Vu (25082165), Charlie Hunter (27380476)
Router support function
"""
import os, sys
import numpy as np
from datetime import datetime

FILE_EXTENSION = ".txt"

def read_config(filename):
    rID, inputs, outputs, timeout = None, None, None, None
    if filename.endswith(FILE_EXTENSION):
        config_file = open(filename)
    else:
        config_file = open(filename + FILE_EXTENSION)
    config_data = config_file.readlines()

    for line in config_data:
        head, data = line.split(':')
        if head == "router-id":
            rID = int(data)
            if not 0 < rID or rID > 64000:
                raise ValueError("Router ID must be between 1 and 64000.")
        elif head == "input-ports":
            inputs = [int(port) for port in data.rstrip().split(',')]
            if not is_valid_ports(inputs):
                raise ValueError("Invalid input port(s) in config data.\nPorts must be between 1024 and 64000.")
        elif head == "outputs":
            outputs = [port.strip() for port in data.rstrip().split(',')]
            ports = [int(output.split('-')[1]) for output in outputs]
            if not is_valid_ports(ports):
                raise ValueError("Invalid output port(s) in config data.\nPorts must be between 1024 and 64000.")
        elif head == "timer":
            timeout = int(data)
            if not 0 < timeout or timeout > 30:
                raise ValueError("Timeout must be between 1 and 30.")
    return rID, inputs, outputs, timeout

def is_valid_ports(ports):
    ports = np.array(ports)
    return np.all((ports ⩾ 1024) & (ports ⩽ 64000))

def create_rip_packet(table):
    header = create_rip_head()
    body = bytearray()
    for entry in table:
        new_entry = create_rip_entry(entry)
        body += new_entry
    return header + body

def create_rip_head(TTL=0):
    "Creates the 4 byte header"
    command = 1
    verison = 2
```

```
    command = command.to_bytes(1, byteorder='big')
    verison = verison.to_bytes(1, byteorder='big')
    reserve = (TTL+1).to_bytes(2, byteorder='big')
    return command + verison + reserve

def create_rip_entry(entry):
    "Creates the 20 byte body of packet"
    address_fam, zero = 0, 0
    afi         = address_fam.to_bytes(2, byteorder='big')
    route_tag   =        zero.to_bytes(2, byteorder='big')
    dest        =    entry[0].to_bytes(4, byteorder='big') # routerID
    subnet      =        zero.to_bytes(4, byteorder='big')
    next_hop    =    entry[1].to_bytes(4, byteorder='big')
    metric      =    entry[2].to_bytes(4, byteorder='big')
    return afi + route_tag + dest + subnet + next_hop + metric

def process_rip_packet(packet):
    command = int.from_bytes(packet[0:1], byteorder='big')
    version = int.from_bytes(packet[1:2], byteorder='big')
    if command ≠ 1 or version ≠ 2:
        return []

    routes = []
    entry_count = (len(packet)-4)//20
    for i in range(entry_count):
        si = i*20 + 4 # entry_start_index
        dest_id  = int.from_bytes(packet[si+4:si+8], byteorder='big')
        next_hop = int.from_bytes(packet[si+12:si+16], byteorder='big')
        metric   = int.from_bytes(packet[si+16:si+20], byteorder='big')
        routes.append((dest_id, next_hop, metric))

    return routes

def packet_check():
    pass

def strCurrTime(time=None):
    if time is None:
        return datetime.now().strftime('%H:%M:%S.%f')[:-3]
    else:
        return time.strftime('%H:%M:%S.%f')[:-3]

def getTime(as_float=False):
    """ Get current time as float or object """
    if as_float:
        return datetime.now().timestamp()
    else:
        return datetime.now()
```