# README

November 21, 2021

Jacob Valdez 1001628688
CSE 4344 Project 2
Network Simulator

This program simulates a network of DV-routing routers with support for ASCII message passing (in-thread only)

1. Please start the program with the following command: `python3 1001628688.py` (You should not need to install any dependencies other than the base Python 3.7+ interpreter.)
2. Enter the file name of the topology file (with the extension)
3. Select the options by entering a number corresponding to the option

*Please note that this program is not intended to be a fully-functional network simulator. It is intended to be a tool for demonstrating packet dropping, routing, and distance vector table updates.*

**My observations:** - Using the `network_example.txt` file provided from canvas, I observed that 3 iterations were required to reach convergence. - Programming this, I realized that you cannot determine if the network is converged at instance $t$ until time $t+1$. Therefore, I implemented a 'is not converged', 'might be converging', and 'is converging' approach to convergence analysis. (See option 8.) - I also noticed many similarities between the competitive inhibitory activation in biological networks and the winner-take-route minimum-cost competition in DV-routing networks. I will apply lessons learned from this project to my research in deep neural network architecture design. I started researching this and found several interesting papers: - Can Information Flows Suggest Targets for Interventions in Neural Circuits? In this paper, the authors performed 'surgery' on a neural network to reduce its racial bias. This seems related to the dynamic restructuring I observed in DV-routing networks. Of course, the Internet is mostly beyond DV-based routing mechanisms so I should think about how to apply similar techniques from BGP, EGP, and other routing protocols to neural network topology design. - Training Competitive Binary Neural Networks from Scratch This paper is older than the one above (2018), but it seems seminal on the topic of binary neural networks. The authors claim they have found a way to train a binary neural networks directly (i.e.: without starting from a floating point neural network). This seems very important on the topic of efficient route updates.

[ ]:
```
from __future__ import annotations

from queue import Queue
from typing import NamedTuple, Union, Mapping, Tuple
```

[ ]:
```python
inf = 16  # infinity constant

DV = Mapping[str, Tuple[str, float]]  # distance vector {dst: (next_hop, cost)}}

class Packet:
    """Packet represents the unit of communication at the network layer"""
    src: int
    dst: int

    def __init__(self, src: int, dst: int):
        self.src = src
        self.dst = dst

class DVPacket(Packet):
    """Packet containing distance vector"""
    dv: DV

    def __init__(self, src: int, dst: int, dv: DV):
        super().__init__(src, dst)
        self.dv = dv

class HTTPPacket(Packet):
    """Message represents the unit of communication at the application layer"""
    msg: str

    def __init__(self, src: int, dst: int, msg: str):
        super().__init__(src, dst)
        self.msg = msg
```

[ ]:
```python
class Router:

  def __init__(self, addr, costs, in_queue, out_queues):
    self.addr = addr
    self.costs = costs
    self.in_queue = in_queue
    self.out_queues = out_queues
    self.reset()

  def reset(self):
    """Reset the router to initial state"""
    self.dv = {dst: (dst, cost) for dst, cost in self.costs.items()}
    self.converged = 2
    while not self.in_queue.empty():
      self.in_queue.get()

  def route(self):
    """Route the packet to the next hop"""
```

```
    while not self.in_queue.empty():
      packet = self.in_queue.get()
      if packet.dst == self.addr:
        # packet is for this router
        self.read_packet(packet)
      elif packet.dst not in self.dv:
        # drop packet
        print(f'{self.addr}: dropped packet from {packet.src} to {packet.dst}.␣
↪Destination not in routing table.')
      else:
        # forward packet
        next_hop, _ = self.dv[packet.dst]
        self.out_queues[next_hop].put(packet)
        print(f'{self.addr}: forwarding packet to {next_hop}')

  def read_packet(self, packet: Packet):
    """Read packet and maybe update distance vector"""
    if isinstance(packet, DVPacket):
      print(f'{self.addr}: received DV from {packet.src}')
      self.update_dv(packet.src, packet.dv)
    elif isinstance(packet, HTTPPacket):
      print(f'{self.addr}: received message from {packet.src}: {packet.msg}')
    else:
      raise ValueError(f'Unknown packet type: {type(packet)}')

  def update_dv(self, src: str, dv_table: DV):
    """Maybe update distance vector"""
    if src not in self.dv:
      self.dv[src] = (src, inf)

    # add new entries
    for dst, (next_hop, cost) in dv_table.items():
      if dst not in self.dv:
        self.dv[dst] = (next_hop, self.dv[src][1] + cost)
        self.converged = 2

    # upper-bound entries
    for dst, (next_hop, cost) in self.dv.items():
      if cost > inf:
        self.dv[dst] = (next_hop, inf)
      if dst == self.addr:
        self.dv[dst] = (dst, 0)

    # implementation of Bellman-Ford algorithm
    # Dx(y) = min { C(x,v) + Dv(y), Dx(y) } for each node y   N
    # https://www.geeksforgeeks.org/distance-vector-routing-dvr-protocol/
    for dst, (_, cost) in dv_table.items():
```

```
      if dst == self.addr:
        continue
      # update cost if necessary
      # x = self.addr
      # v = src
      # y = dst
      # C(x,v) = self.dv[src][1]
      # Dv(y) = cost
      # Dx(y) = self.dv[dst][1]
      if self.dv[src][1] + cost < self.dv[dst][1]:
        self.dv[dst] = (src, self.dv[src][1] + cost)
        self.converged = 2

    # upper-bound entries again
    for dst, (next_hop, cost) in self.dv.items():
      if cost > inf:
        self.dv[dst] = (next_hop, inf)
      if dst == self.addr:
        self.dv[dst] = (dst, 0)

  def send_dv(self):
    """Send distance vector to all neighbors"""
    if self.converged == 0:
      return

    # assume converged after 2 iterations with no change
    self.converged -= 1
    # send dv to all neighbors
    for dst, queue in self.out_queues.items():
      queue.put(DVPacket(src=self.addr, dst=dst, dv=self.dv))
```

[ ]:
```
class Network(object):

  def __init__(self, file):
    addrs = set()
    costs = dict()
    for line in open(file):
      (src, dst, cost) = line.split()
      addrs = addrs | {src, dst}
      costs[(src, dst)] = int(cost)

    # queues represent the physical layer
    self.queues = {addr: Queue() for addr in addrs}

    # routers represent the network layer
    self.routers = {addr: Router(
      addr=addr,
```

```
      costs={dst: costs[(src, dst)] for (src, dst) in costs.keys() if src ==␣
↪addr},
      in_queue=self.queues[addr],
      out_queues={dst: self.queues[dst] for (src, dst) in costs.keys() if src␣
↪== addr}
    ) for addr in addrs}

  def reset(self):
    """Resets all routers to their initial state."""
    for router in self.routers.values():
      router.reset()

  def send_dv(self):
    """Asks each router to send their distance vectors.
    NOTE: Recieving routers do not update their dv table until `route` is␣
↪called."""
    for router in self.routers.values():
      router.send_dv()

  def route(self):
    """Asks each router to empty its incoming queue.
    If the queue contains distance vectors, the router updates its own table."""
    for router in self.routers.values():
      router.route()

  def is_stable(self):
    """Returns true if the distance vectors have not changed since the last␣
↪call to .route."""
    return all(router.converged==0 for router in self.routers.values())

  def step(self):
    """Has all routers send their dv's and then route (which may include␣
↪routing non-dv messages)"""
    self.send_dv()
    self.route()

  def run_until_converged(self) -> int:
    """Runs until the distance vectors converge.
    Returns the number of steps taken."""
    steps = 0
    while not self.is_stable():
      self.step()
      steps += 1
    return steps
```

```
[ ]: def main():

  print('Jacob Valdez 1001628688')
  print('CSE 4344 Project 2')
  print('Network Simulator')
  print('')
  print('This program simulates a network of DV-routing routers')
  print('with support for ASCII message passing (in-thread only)')
  print('')

  network = Network(input('Enter a file name to simulate: '))

  while True:
    print(128*'\n')
    print('Please select an option')
    print('0. Run until convergence')
    print('1. Run one step (routing and DV updating)')
    print('2. Reset')
    print('3. Send message')
    print('4. Change link cost')
    print('5. Route (don\'t send DV\'s)')
    print('6. Send DV\'s (don\'t route buffered packets)')
    print('7. view DV tables')
    print('8. view convergence status')
    print('9. Exit')

    print('')
    choice = input('Enter your choice: ')
    print('')

    if choice == '0':
      steps = network.run_until_converged()
      print(f'Converged after {steps} steps')
    elif choice == '1':
      network.step()
    elif choice == '2':
      network.reset()
    elif choice == '3':
      src = input('Enter source address: ')
      dst = input('Enter destination address: ')
      msg = input('Enter message: ')
      network.queues[src].put(HTTPPacket(src=src, dst=dst, msg=msg))
    elif choice == '4':
      src = input('Enter source address: ')
      dst = input('Enter destination address: ')
      cost = input('Enter new cost: ')
      cost = int(cost)
```

```
      network.routers[src].costs[dst] = int(cost)
      if dst not in network.routers[src].dv:
        network.routers[src].dv[dst] = (dst, inf)
      old_cost = network.routers[src].dv[dst][1]
      if cost < old_cost:
        network.routers[src].converged = 2
        network.routers[src].dv[dst] = (dst, cost)
    elif choice == '5':
      network.route()
    elif choice == '6':
      network.send_dv()
    elif choice == '7':
      for router in network.routers.values():
        for dst, (next_hop, cost) in router.dv.items():
          print(f'{router.addr} -> {dst} (next_hop={next_hop}, cost={cost})')
    elif choice == '8':
      print('network is converged:', network.is_stable())
      for addr, router in network.routers.items():
        if router.converged >= 2:
          print(f'{addr} is not converged')
        elif router.converged == 1:
          print(f'{addr} might be converged next step')
        else:
          print(f'{addr} is converged')
    elif choice == '9':
      break
    else:
      print('Invalid choice')

    print('')
    input('Press enter to continue')

  print('Exiting...')

main()
```