K-out-N Multicast System

Group 2

Members:
Yuan Cheng
Amandip Kaler
Michael Diaz
Xiaotian Qiang

Introduction: We created a k-out-of-N packet datagram multicast network. The entire system has max 3 destinations and each packet is delivered to k out of n destinations. K could be 1, 2, or 3. Based on specific topology, we can determine the best multicast path according to the routing protocol.

Design Requirements:

Assumptions:
- Unreliable network with packet loss prob. per link of p
- Each end node is attached to only one router
- All links have the same characteristics: hop cost 1 and same MTU 1500 bytes
- Small maximum number of nodes (50)
- IDs 100-199 are destination nodes
- IDs 200-254 are routers
- Packets are going to be delivered to k out of n destinations
- The maximum value of k=3
- All routing tables are stored at a centralized location, so the core router can access them.
- Rendezvous Point is topology selected and not k-out-n selected
- Hosts have pre-configured IPs/IDs

Addressing:
1. All the nodes have the fixed IP addresses
2. All the nodes have prefigured IDs: For hosts and source, ID: 100-199
   For routers, ID: 200-254

Protocol Design:

Packet Structure: OSPF packets (Hello packet, LSPacket, DATApacket, DATA ACK packet).
Routing Protocol: Use LS protocol based on Dijkstra's shortest path algorithm. Dijkstra's shortest path algorithm will be applied on every router, hosts, source so that every node will create a routing table including every other nodes' information.

Implementation:

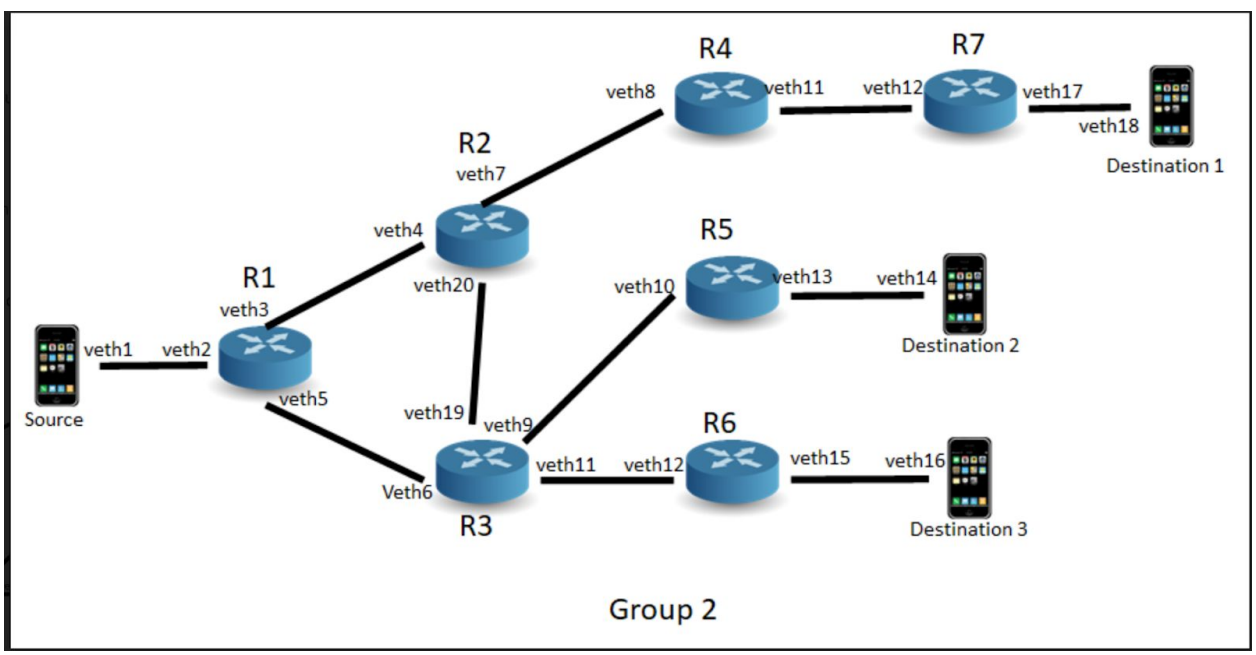
The implementation is focusing on using Dijkstra's routing algorithm to create a routing table for each router, host, and source. With completed routing tables, we can select the router with the least total cost as RP and use RP to implement the k-out-of-N multicast.

In the beginning, with given graph information such as the nodes' IP addresses and all the connections in the graph, we directly apply Dijkstra's algorithm for all the routers, hosts, and source.

Secondly, with the output from Dijkstra's algorithm, we easily compute the total cost of each router connecting to other routers. The router with the least total cost will be selected as a Rendezvous Point (RP).

Last but not least, after the selection of RP, the source will determine the value of k, and this k value is manually adjusted. If k = 1, the source will unicast the nearest destination; if k = 2 or k = 3, the source will unicast the data packet to the RP, and RP will copy the data packets and forward the packets to the destinations. Whenever the destinations receive the data packets, they will respond with ACK packets back to the source.

Our topology used for the project and the associated ID's:



Demo/Results:

Routing Tables:

```python

import sys
from collections import deque
import numpy as np
class Graph():

    def __init__(self, vertices):
        self.V = vertices
        self.graph = [[0 for column in range(vertices)] for row in range(vertices)]

    def printSolution(self,dist,nodelist):
        print "Vertex tDistance from Source"
        for node in range(self.V):
            print nodelist[node], "t", dist[node]

    # A utility function to find the vertex with
    # minimum distance value, from the set of vertices
    # not yet included in shortest path tree
    def minDistance(self, dist, sptSet):

        # Initilaize minimum distance for next node
        min = sys.maxint

        # Search not nearest vertex not in the
        # shortest path tree
        for v in range(self.V):
            if dist[v] < min and sptSet[v] == False:
                min = dist[v]
                min_index = v

        return min_index

    # Funtion that implements Dijkstra's single source
    # shortest path algorithm for a graph represented
    # using adjacency matrix representation
    def dijkstra(self, src, nodelist):

        dist = [sys.maxint] * self.V
        dist[src] = 0
        sptSet = [False] * self.V
        neigh = [[None for i in range(mat_cols)] for j in range(mat_rows)]

        for cout in range(self.V):

            # Pick the minimum distance vertex from
            # the set of vertices not yet processed.
            # u is always equal to src in first iteration
            u = self.minDistance(dist, sptSet)
```

```
            # the set of vertices not yet processed.
            # u is always equal to src in first iteration
            u = self.minDistance(dist, sptSet)

            # Put the minimum distance vertex in the
            # shotest path tree
            sptSet[u] = True

            # Update dist value of the adjacent vertices
            # of the picked vertex only if the current
            # distance is greater than new distance and
            # the vertex in not in the shotest path tree
            for v in range(self.V):
                if self.graph[u][v] > 0 and sptSet[v] == False and  dist[v] > dist[u] + self.graph[u][v]:
                    dist[v] = dist[u] + self.graph[u][v]
                    neigh[u][v] = nodelist[v]

        #self.printSolution(dist,nodelist)
        return neigh

def dict_to_mat(graph,node_list):
    mat_graph = [[0 for i in range(len(node_list))] for j in range(len(node_list))]
    for i in range(0,len(nodes)):
        for j in range(0,len(nodes)):
            if(nodes[i] == nodes[j]):
                mat_graph[i][j] = 0
            elif(nodes[j] not in network[nodes[i]]):
                mat_graph[i][j]=0
            else:
                mat_graph[i][j]=1
    return mat_graph

def router_table(router,m_graph,nodes):
    print
    g = Graph(len(nodes))
    g.graph = mat_graph
    mat= g.dijkstra(router,nodes)
    print 'route for ' + nodes[router]
    for i in range(len(nodes)):
        stack = list()
        if(nodes[i] == nodes[router]):
            stack.append(i)
            route = []
            for k in range(len(stack)-1,-1,-1):
                route.append(nodes[(stack[k])])
            print 'destination: %s | path : %s | distance: %3d' %(nodes[i],route,len(stack)-1)
            continue
        stack.append(i)
        src_col = [sub[i] for sub in mat]
```

```python
        stack.append(i)
        src_col = [sub[i] for sub in mat]
        while True:
           # src_col = mat[:,0]
            for j in range(len(src_col)):
                if src_col[j] is not None:
                    addr_index = j
                    stack.append(j)
                    break
            if nodes[j] == nodes[router]:
                break
            src_col = [sub[j] for sub in mat]

        route = []
        for k in range(len(stack)-1,-1,-1):
            route.append(nodes[(stack[k])])
        print 'destination: %s | path : %s | distance: %3d' %(nodes[i],route,len(stack)-1)



r1,r2,r3,r4,r5,r6,r7 = '192.168.4.2 (r1)','192.168.1.3 (r2)','192.168.2.3 (r3)','192.168.3.3 (r4)','192.168.1.2 (r5)','192.168.2.2 (r6)','192.168.3.2 (r7)'
h1,h2,h3 =  '192.168.1.1 (h1)','192.168.2.1 (h2)','192.168.3.1 (h3)'
src = '192.168.4.3 (src)'
routers = [r1,r2,r3,r4,r5,r6,r7]
hosts = [h1,h2,h3]
nodes = [src,r1,r2,r3,r4,r5,r6,r7,h1,h2,h3]


network = {src: {r1},
        r1: {src,r2, r3},
        r2: {r1,r4},
        r3:{r1,r5,r6},
        r4:{r2,r7},
        r5:{r3,h2},
        r6:{r3,h3},
        r7:{r4,h1},
        h1:{r7},
        h2:{r5},
        h3:{r6},}

mat_rows,mat_cols = (len(nodes), len(nodes))
mat_graph = [[0 for i in range(mat_cols)] for j in range(mat_rows)]
mat_graph = dict_to_mat(network,nodes)
for i in range(len(nodes)):
    router_table(i,mat_graph,nodes)
```

Routing table output:

```
route for 192.168.4.3
destination: 192.168.4.2 | path : ['192.168.4.3'] | distance:   0
destination: 192.168.4.2 | path : ['192.168.4.3', '192.168.4.2'] | distance:   1
destination: 192.168.1.3 | path : ['192.168.4.3', '192.168.4.2', '192.168.1.3'] | distance:   2
destination: 192.168.2.2 | path : ['192.168.4.3', '192.168.4.2', '192.168.2.2'] | distance:   2
destination: 192.168.3.3 | path : ['192.168.4.3', '192.168.4.2', '192.168.3.3'] | distance:   2
destination: 192.168.1.2 | path : ['192.168.4.3', '192.168.4.2', '192.168.1.3', '192.168.1.2'] | distance:   3
destination: 192.168.2.2 | path : ['192.168.4.3', '192.168.4.2', '192.168.2.3', '192.168.2.2'] | distance:   3
destination: 192.168.3.2 | path : ['192.168.4.3', '192.168.4.2', '192.168.3.3', '192.168.3.2'] | distance:   3
destination: 192.168.1.1 | path : ['192.168.4.3', '192.168.4.2', '192.168.1.3', '192.168.1.2', '192.168.1.1'] | distance:   4
destination: 192.168.2.1 | path : ['192.168.4.3', '192.168.4.2', '192.168.2.3', '192.168.2.2', '192.168.2.1'] | distance:   4
destination: 192.168.3.1 | path : ['192.168.4.3', '192.168.4.2', '192.168.3.3', '192.168.3.2', '192.168.3.1'] | distance:   4
route for 192.168.4.2
destination: 192.168.4.3 | path : ['192.168.4.2', '192.168.4.3'] | distance:   1
destination: 192.168.4.2 | path : ['192.168.4.2'] | distance:   0
destination: 192.168.1.3 | path : ['192.168.4.2', '192.168.1.3'] | distance:   1
destination: 192.168.2.3 | path : ['192.168.4.2', '192.168.2.3'] | distance:   1
destination: 192.168.3.2 | path : ['192.168.4.2', '192.168.3.2'] | distance:   1
destination: 192.168.1.2 | path : ['192.168.4.2', '192.168.1.3', '192.168.1.2'] | distance:   2
destination: 192.168.2.2 | path : ['192.168.4.2', '192.168.2.3', '192.168.2.2'] | distance:   2
destination: 192.168.3.2 | path : ['192.168.4.2', '192.168.3.3', '192.168.3.2'] | distance:   2
destination: 192.168.1.1 | path : ['192.168.4.2', '192.168.1.3', '192.168.1.2', '192.168.1.1'] | distance:   3
destination: 192.168.2.1 | path : ['192.168.4.2', '192.168.2.3', '192.168.2.2', '192.168.2.1'] | distance:   3
destination: 192.168.3.1 | path : ['192.168.4.2', '192.168.3.3', '192.168.3.2', '192.168.3.1'] | distance:   3
route for 192.168.1.3
destination: 192.168.4.3 | path : ['192.168.1.3', '192.168.4.2', '192.168.4.3'] | distance:   2
destination: 192.168.4.2 | path : ['192.168.1.3', '192.168.4.2'] | distance:   1
destination: 192.168.1.3 | path : ['192.168.1.3'] | distance:   0
destination: 192.168.2.3 | path : ['192.168.1.3', '192.168.4.2', '192.168.2.3'] | distance:   2
destination: 192.168.3.3 | path : ['192.168.1.3', '192.168.4.2', '192.168.3.3'] | distance:   2
destination: 192.168.1.2 | path : ['192.168.1.3', '192.168.1.2'] | distance:   1
destination: 192.168.2.2 | path : ['192.168.1.3', '192.168.1.2', '192.168.2.2'] | distance:   2
destination: 192.168.3.2 | path : ['192.168.1.3', '192.168.4.2', '192.168.3.3', '192.168.3.2'] | distance:   3
destination: 192.168.1.1 | path : ['192.168.1.3', '192.168.1.2', '192.168.1.1'] | distance:   2
destination: 192.168.2.1 | path : ['192.168.1.3', '192.168.1.2', '192.168.2.2', '192.168.2.1'] | distance:   3
destination: 192.168.3.1 | path : ['192.168.1.3', '192.168.4.2', '192.168.3.3', '192.168.3.2', '192.168.3.1'] | distance:   4
```

```
destination: 192.168.4.3 | path : ['192.168.2.2', '192.168.2.3', '192.168.4.2', '192.168.4.3'] | distance:   3
destination: 192.168.4.2 | path : ['192.168.2.2', '192.168.2.3', '192.168.4.2'] | distance:   2
destination: 192.168.1.3 | path : ['192.168.2.2', '192.168.1.2', '192.168.1.3'] | distance:   2
destination: 192.168.2.3 | path : ['192.168.2.2', '192.168.2.3'] | distance:   1
destination: 192.168.3.3 | path : ['192.168.2.2', '192.168.3.2', '192.168.3.3'] | distance:   2
destination: 192.168.1.2 | path : ['192.168.2.2', '192.168.1.2'] | distance:   1
destination: 192.168.2.2 | path : ['192.168.2.2'] | distance:   0
destination: 192.168.3.2 | path : ['192.168.2.2', '192.168.3.2'] | distance:   1
destination: 192.168.1.1 | path : ['192.168.2.2', '192.168.1.2', '192.168.1.1'] | distance:   2
destination: 192.168.2.1 | path : ['192.168.2.2', '192.168.2.1'] | distance:   1
destination: 192.168.3.1 | path : ['192.168.2.2', '192.168.3.2', '192.168.3.1'] | distance:   2
route for 192.168.3.2
destination: 192.168.4.3 | path : ['192.168.3.2', '192.168.3.3', '192.168.4.2', '192.168.4.3'] | distance:   3
destination: 192.168.4.2 | path : ['192.168.3.2', '192.168.3.3', '192.168.4.2'] | distance:   2
destination: 192.168.1.3 | path : ['192.168.3.2', '192.168.3.3', '192.168.4.2', '192.168.1.3'] | distance:   3
destination: 192.168.2.3 | path : ['192.168.3.2', '192.168.2.2', '192.168.2.3'] | distance:   2
destination: 192.168.3.3 | path : ['192.168.3.2', '192.168.3.3'] | distance:   1
destination: 192.168.1.2 | path : ['192.168.3.2', '192.168.2.2', '192.168.1.2'] | distance:   2
destination: 192.168.2.2 | path : ['192.168.3.2', '192.168.2.2'] | distance:   1
destination: 192.168.3.2 | path : ['192.168.3.2'] | distance:   0
destination: 192.168.1.1 | path : ['192.168.3.2', '192.168.2.2', '192.168.1.2', '192.168.1.1'] | distance:   3
destination: 192.168.2.1 | path : ['192.168.3.2', '192.168.2.2', '192.168.2.1'] | distance:   2
destination: 192.168.3.1 | path : ['192.168.3.2', '192.168.3.1'] | distance:   1
route for 192.168.1.1
destination: 192.168.4.3 | path : ['192.168.1.1', '192.168.1.2', '192.168.1.3', '192.168.4.2', '192.168.4.3'] | distance:   4
destination: 192.168.4.2 | path : ['192.168.1.1', '192.168.1.2', '192.168.1.3', '192.168.4.2'] | distance:   3
destination: 192.168.1.3 | path : ['192.168.1.1', '192.168.1.2', '192.168.1.3'] | distance:   2
destination: 192.168.2.3 | path : ['192.168.1.1', '192.168.1.2', '192.168.2.2', '192.168.2.3'] | distance:   3
destination: 192.168.3.3 | path : ['192.168.1.1', '192.168.1.2', '192.168.1.3', '192.168.4.2', '192.168.3.3'] | distance:   4
destination: 192.168.1.2 | path : ['192.168.1.1', '192.168.1.2'] | distance:   1
destination: 192.168.2.2 | path : ['192.168.1.1', '192.168.1.2', '192.168.2.2'] | distance:   2
destination: 192.168.3.2 | path : ['192.168.1.1', '192.168.1.2', '192.168.2.2', '192.168.3.2'] | distance:   3
destination: 192.168.1.1 | path : ['192.168.1.1'] | distance:   0
destination: 192.168.2.1 | path : ['192.168.1.1', '192.168.1.2', '192.168.2.2', '192.168.2.1'] | distance:   3
destination: 192.168.3.1 | path : ['192.168.1.1', '192.168.1.2', '192.168.2.2', '192.168.3.2', '192.168.3.1'] | distance:   4
route for 192.168.2.1
```

```
route for 192.168.2.1
destination: 192.168.4.3 | path : ['192.168.2.1', '192.168.2.2', '192.168.2.3', '192.168.4.2', '192.168.4.3'] | distance:   4
destination: 192.168.4.2 | path : ['192.168.2.1', '192.168.2.2', '192.168.2.3', '192.168.4.2'] | distance:   3
destination: 192.168.1.3 | path : ['192.168.2.1', '192.168.2.2', '192.168.1.2', '192.168.1.3'] | distance:   3
destination: 192.168.2.3 | path : ['192.168.2.1', '192.168.2.2', '192.168.2.3'] | distance:   2
destination: 192.168.3.3 | path : ['192.168.2.1', '192.168.2.2', '192.168.3.2', '192.168.3.3'] | distance:   3
destination: 192.168.1.2 | path : ['192.168.2.1', '192.168.2.2', '192.168.1.2'] | distance:   2
destination: 192.168.2.2 | path : ['192.168.2.1', '192.168.2.2'] | distance:   1
destination: 192.168.3.2 | path : ['192.168.2.1', '192.168.2.2', '192.168.3.2'] | distance:   2
destination: 192.168.1.1 | path : ['192.168.2.1', '192.168.2.2', '192.168.1.2', '192.168.1.1'] | distance:   3
destination: 192.168.2.1 | path : ['192.168.2.1'] | distance:   0
destination: 192.168.3.1 | path : ['192.168.2.1', '192.168.2.2', '192.168.3.2', '192.168.3.1'] | distance:   3
route for 192.168.3.1
destination: 192.168.4.3 | path : ['192.168.3.1', '192.168.3.2', '192.168.3.3', '192.168.4.2', '192.168.4.3'] | distance:   4
destination: 192.168.4.2 | path : ['192.168.3.1', '192.168.3.2', '192.168.3.3', '192.168.4.2'] | distance:   3
destination: 192.168.1.3 | path : ['192.168.3.1', '192.168.3.2', '192.168.3.3', '192.168.4.2', '192.168.1.3'] | distance:   4
destination: 192.168.2.3 | path : ['192.168.3.1', '192.168.3.2', '192.168.2.2', '192.168.2.3'] | distance:   3
destination: 192.168.3.3 | path : ['192.168.3.1', '192.168.3.2', '192.168.3.3'] | distance:   2
destination: 192.168.1.2 | path : ['192.168.3.1', '192.168.3.2', '192.168.2.2', '192.168.1.2'] | distance:   3
destination: 192.168.2.2 | path : ['192.168.3.1', '192.168.3.2', '192.168.2.2'] | distance:   2
destination: 192.168.3.2 | path : ['192.168.3.1', '192.168.3.2'] | distance:   1
destination: 192.168.1.1 | path : ['192.168.3.1', '192.168.3.2', '192.168.2.2', '192.168.1.2', '192.168.1.1'] | distance:   4
destination: 192.168.2.1 | path : ['192.168.3.1', '192.168.3.2', '192.168.2.2', '192.168.2.1'] | distance:   3
destination: 192.168.3.1 | path : ['192.168.3.1'] | distance:   0
```

Here is the algorithm for the routing tables. How we did this is by using dijkstra [1] on a matrix represented graph, where the nodes are order as:

```
r1,r2,r3,r4,r5,r6,r7 = '192.168.4.2 (r1)','192.168.1.3 (r2)','192.168.2.3 (r3)','192.168.3.3 (r4)','192.168.1.2 (r5)','192.168.2.2 (r6)','192.168.3.2 (r7)'
h1,h2,h3 =  '192.168.1.1 (h1)','192.168.2.1 (h2)','192.168.3.1 (h3)'
src = '192.168.4.3 (src)'
```

```
nodes = [src,r1,r2,r3,r4,r5,r6,r7,h1,h2,h3]
```

Afterwards we created a routing table function. Here is the function and on top of this text is the output.

```
def router_table(router,m_graph,nodes):
    print
    g = Graph(len(nodes))
    g.graph = mat_graph
    mat= g.dijkstra(router,nodes)
    print 'route for ' + nodes[router]
    for i in range(len(nodes)):
        stack = list()
        if(nodes[i] == nodes[router]):
            stack.append(i)
            route = []
            for k in range(len(stack)-1,-1,-1):
                route.append(nodes[(stack[k])])
            print 'destination: %s | path : %s | distance: %3d' %(nodes[i],route,len(stack)-1)
            continue
        stack.append(i)
        src_col = [sub[i] for sub in mat]
        while True:
           # src_col = mat[:,0]
            for j in range(len(src_col)):
                if src_col[j] is not None:
                    addr_index = j
                    stack.append(j)
                    break
            if nodes[j] == nodes[router]:
                break
            src_col = [sub[j] for sub in mat]

        route = []
        for k in range(len(stack)-1,-1,-1):
            route.append(nodes[(stack[k])])
        print 'destination: %s | path : %s | distance: %3d' %(nodes[i],route,len(stack)-1)
```

Multicast:

For the **source**, it will unicast the data packet to one of three destinations if k=1 otherwise, it will unicast the data packet to the RP. After sending, the source will start receiving the ACK packets.

```
if __name__ == '__main__':

    router_receive = socket(AF_INET, SOCK_DGRAM)
    router_receive.bind(('192.168.4.3', 1))
    k_val = input('Please input the number of k: ')
    host = ['192.168.1.1','192.168.2.1','192.168.3.1']
    Data_packet = create_datapacket(3, 1, 104, k_val, 1, 101, 102, 103, 'DATA')

    if k_val == 1:
        #unicast to one dest
        dest = random.sample(host, 1)
        send_packet(Data_packet, dest)
    elif k_val == 2:
        #send to RP
        send_packet(Data_packet, '192.168.4.1')

    elif k_val == 3:
        #send to RP
        send_packet(Data_packet, '192.168.4.1')


    #start receiving ack packets
    while True:
        receive_router(router_receive)
```

Rendezvous Point (RP) is chosen as **Router 3** since it has the least total cost to the entire graph. The RP will receive the packets first and then read the NDEST to

determine the value of k. If k = 2, the RP will randomly select two of three destinations to multicast; if k = 3, it will send to all the destinations.

```
if __name__ == '__main__':
    #send data packet to dest2
    router_receive = socket(AF_INET, SOCK_DGRAM)
    router_receive.bind(('192.168.4.1', 1))

    data_packet = receive_router(router_receive)

    #send ack back to R3
    data_ACK = create_dataACK (4, 1, 206, 104)
    send_packet(data_ACK,'192.168.4.2')

    NDEST = read_header(data_packet)
    #from RP, send packet to R5, R7, D2
    Host = ['192.168.2.2','192.168.1.4','192.168.3.2']
    if NDEST == 2:
        selected_host = random.sample(Host, 2)

        for dest in selected_host:
            send_packet(data_packet, dest)

    elif NDEST == 3:
        for dest in Host:
            send_packet(data_packet, dest)

    #start receiving and sending the ack packet
    while True:
        ACK_packet = receive_router(router_receive)
        send_packet(ACK_packet.encode('GBK'), '192.168.4.2')
```

R5, R6, R2 are three routers directly connected to the destinations, so these two routers can forward the data from RP.

R6:

```
if __name__ == '__main__':

    #forward data packet to dest 1
    router_receive = socket(AF_INET, SOCK_DGRAM)
    router_receive.bind(('192.168.3.2', 1))
    #receive forward packet
    data_packet = receive_router(router_receive)
    send_packet(data_packet.encode('GBK'), '192.168.3.1')


    #repeatively forward the ack packet to R2
    data_ACK = receive_router(router_receive)
    send_packet(data_ACK.encode('GBK'), '192.168.4.1')
```

R5:

```
if __name__ == '__main__':
    #forward data packet to dest 1
    router_receive = socket(AF_INET, SOCK_DGRAM)
    router_receive.bind(('192.168.2.2', 1))
    #receive forward packet
    data_packet = receive_router(router_receive)
    send_packet(data_packet.encode('GBK'), '192.168.2.1')


    #repeatively forward the ack packet to R2
    data_ACK = receive_router(router_receive)
    send_packet(data_ACK.encode('GBK'), '192.168.4.1')
```

R2:

```
if __name__ == '__main__':
    #forward data packet to dest 1
    router_receive = socket(AF_INET, SOCK_DGRAM)
    router_receive.bind(('192.168.1.4', 1))
    #receive forward packet
    data_packet = receive_router(router_receive)
    send_packet(data_packet.encode('GBK'), '192.168.1.3')


    #repeatively forward the ack packet to R2
    data_ACK = receive_router(router_receive)
    send_packet(data_ACK.encode('GBK'), '192.168.4.2')
```

Destinations are designed to receive the data packets and respond with ACK packets

```
if __name__ == '__main__':
    router_receive = socket(AF_INET, SOCK_DGRAM)
    router_receive.bind(('192.168.1.1', 1))
    receive_router(router_receive)
    ACK_packet = create_dataACK(4, 1, 101, 104)

    send_packet(ACK_packet, '192.168.1.2')
```

When k =3, the source will unicast the data packet to the RP (Router3) and RP will forward the data packet to its neighbors:

"Node: r3" (root)@mininet-vm

```
ACK_packet = receive_router(router_receive)
send_packet(ACK_packet.encode('GBK'), '192.168.4.2')
```

[Wrote 118 lines]

```
root@mininet-vm:~# sudo python Router3.py
('Received packet', '\x03\x01\x04\x00h\x03\x01efgDATA', 'from source', ('192.168
.4.1', 56220))
('Sent packet to the destination: ', '192.168.4.2')
('Sent packet to the destination: ', '192.168.2.2')
('Sent packet to the destination: ', '192.168.1.4')
('Sent packet to the destination: ', '192.168.3.2')
('Received packet', '\x04\x01fh', 'from source', ('192.168.4.1', 55481))
('Sent packet to the destination: ', '192.168.4.2')
('Received packet', '\x04\x01gh', 'from source', ('192.168.4.1', 58919))
('Sent packet to the destination: ', '192.168.4.2')
```

The destinations will receive the data packet and respond with ACK packets:

"Node: h2" (root)@mininet-vm

```
root@mininet-vm:~# sudo python Dest2.py
('Received packet', '\x03\x01\x04\x00h\x03\x01efgDATA', 'from source', ('192.168
.2.1', 51108))
('Sent packet to the destination: ', '192.168.2.2')
root@mininet-vm:~# sudo python Dest2.py
('Received packet', '\x03\x01\x04\x00h\x03\x01efgDATA', 'from source', ('192.168
.2.1', 57221))
('Sent packet to the destination: ', '192.168.2.2')
root@mininet-vm:~#
```

"Node: h3" (root)@mininet-vm

```
root@mininet-vm:~# sudo python Dest3.py
('Received packet', '\x03\x01\x04\x00h\x03\x01efgDATA', 'from source', ('192.168
.3.1', 60207))
('Sent packet to the destination: ', '192.168.3.2')
root@mininet-vm:~# sudo python Dest3.py
('Received packet', '\x03\x01\x04\x00h\x03\x01efgDATA', 'from source', ('192.168
.3.1', 49496))
('Sent packet to the destination: ', '192.168.3.2')
root@mininet-vm:~#
```

"Node: h1" (root)@mininet-vm

```
root@mininet-vm:~# sudo python Dest1.py
('Received packet', '\x03\x01\x04\x00h\x03\x01efgDATA', 'from source', ('192.168
.1.1', 43182))
('Sent packet to the destination: ', '192.168.1.2')
root@mininet-vm:~# sudo python Dest1.py
('Received packet', '\x03\x01\x04\x00h\x03\x01efgDATA', 'from source', ('192.168
.1.1', 51750))
('Sent packet to the destination: ', '192.168.1.2')
root@mininet-vm:~#
```

Finally, the source will receive the 3 ACK packets:

Reference:

[1] Dijkstra Algorithm
https://www.geeksforgeeks.org/python-program-for-dijkstras-shortest-path-algorithm-greedy-algo-7/