****

**Course Name**: Computer & Communication Networks

**Course Number**: 14:332:423/544

**Protocol Design Project**

**Group 9**

**Submitted by**: Aditya Kesari (RUID: 201002035)

Dhruv Rana (RUID: 209001759)

Harsh Patel (RUID: 195007496)

Melissa Jara (RUID: 212002184)

**Date Submitted**: May 7, 2023

**Contents**

[**Introduction 2**](#_651uhtinra6s)

[1.1 System Design & Goals 2](#_yu3xcj58o83k)

[**Code Implementation 3**](#_cck6vr4tvs0k)

[2.1 NetworkNode.py 3](#_1dzsbuft15pd)

[2.2 Network.py 3](#_jsccme41ptrh)

[2.3 NetworkMap.py 5](#_f2somiku7v2e)

[2.4 Topo.py 5](#_779f4i8zqenp)

[2.5 cleanup.py 6](#_2zjil0ga4t8z)

[2.6 Packet.py 7](#_y1eiiwc9ze36)

[2.7 destination.py 7](#_tl4vzufzm37y)

[2.8 source.py 9](#_mkdjv2qgupu9)

[2.9 router.py 10](#_hl5ihachws11)

[2.10 multicastRP.py 12](#_gm0j3rwcdsb5)

[**Conclusion 15**](#_11d1q4p8ty48)

[3.1 Accomplishments & Evaluation 15](#_hoa2xgr7n6or)

# 

# 

# **Introduction**

## **1.1 System Design & Goals**

The objective of this project was to design a k-out-of-n packet datagram multicast network. To fulfill this, our system design followed the basic principles outlined in the technical specification document. The main principle was to implement a rendezvous point algorithm along with a dynamic selection algorithm to ensure that the protocol chooses the best multicast path for any given topology.

Initially the network will be flooded with link state packets to ensure that each router learns the network topology. Then, a router will be manually selected as the initial rendezvous point. This initial rendezvous point will use Djikstra’s algorithm to determine the location of the dynamic RP.

The dynamic rendezvous point algorithm uses a protocol called Protocol Independent Multicast (PIM) to discover the location of RPs in the network and select the best RP. The best rendezvous point in this case is defined as the router with the lowest-average distance to the k-destinations. After the D-RP is selected, the multicast packets will be routed from the initial RP to the D-RP. The D-RP will then break down the multicast packet and unicast to k out of n destinations.

One of our goals is to have a reliable network. Reliability refers to the ability of a network to operate without disruption or failure. A reliable network should have minimal downtime and packet loss. The network should be able to cast to both a single datagram packet (Uni-casting), or to multiple end clients (Multi-casting).

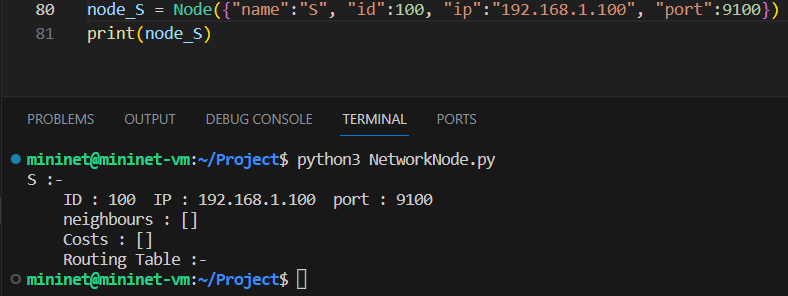
## 

# 

# **Code Implementation**

## **2.1 NetworkNode.py**

A node in the network holds all information about itself but still lacks the necessary information about its neighbor node(s).

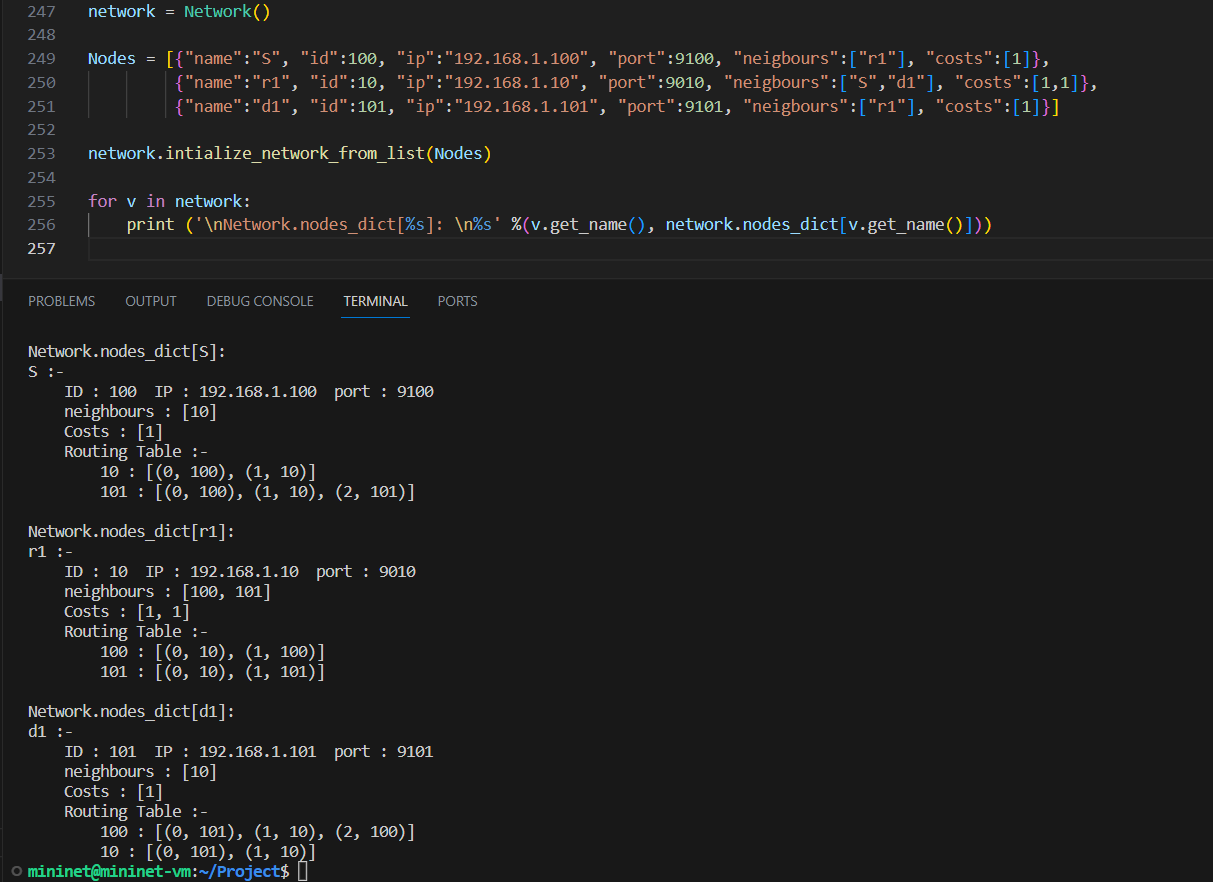


## 

## 

## **2.2 Network.py**

A network holds information about all the does in the network and allows the user to initialize the network using an array of dictionaries that hold all the necessary information about the node and its connection. After it has created all the nodes in the network, it runs Dijkstra on each node to set up the routing table of the node, where it is saved in the format “destination : [shortest path to destination]”. It also holds the function to find the dynamic RP of the network when given the src, destinations, and k value.





## **2.3 NetworkMap.py**

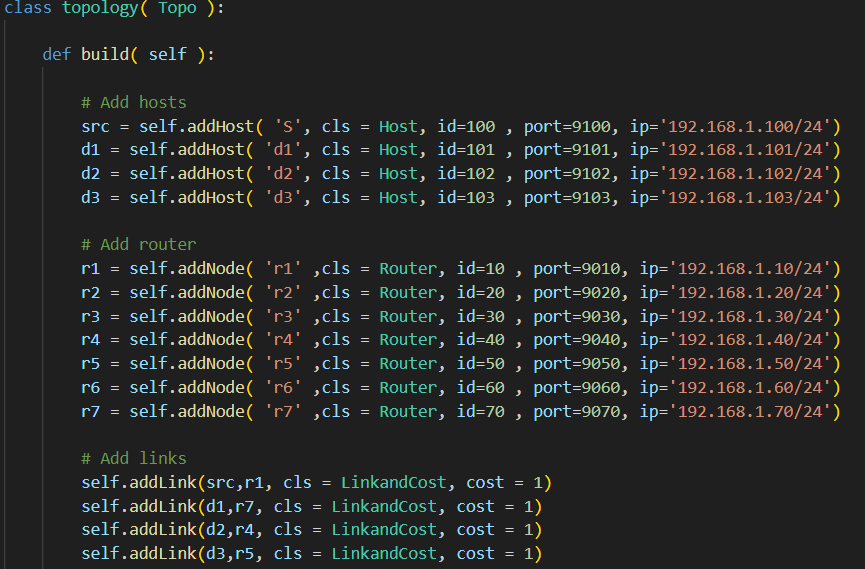
A network map is where all the information about the network is held for any process to access the network. We found that creating a dictionary for the network was getting a bit tedious, and after reading through the documentation of Mininet found that we can allow nodes to extra information using custom cls classes. However, all functionality for creating the network through a dictionary is still available but we prefer the use of custom cls classes which will be explained in the “Topo.py” file.



## **2.4 Topo.py**

The topo file holds all the information about the topology of the network. We utilized custom cls classes shown below to ensure that each node can hold extra information about itself. This removes the need for setting up a dictionary of the network and prevents any mistake that could occur when a user makes a typo in the dictionary. Thus, the use of a custom cls class is highly recommended, as it not only makes implication easier but allows the rest of the code to dynamically change from one single file.



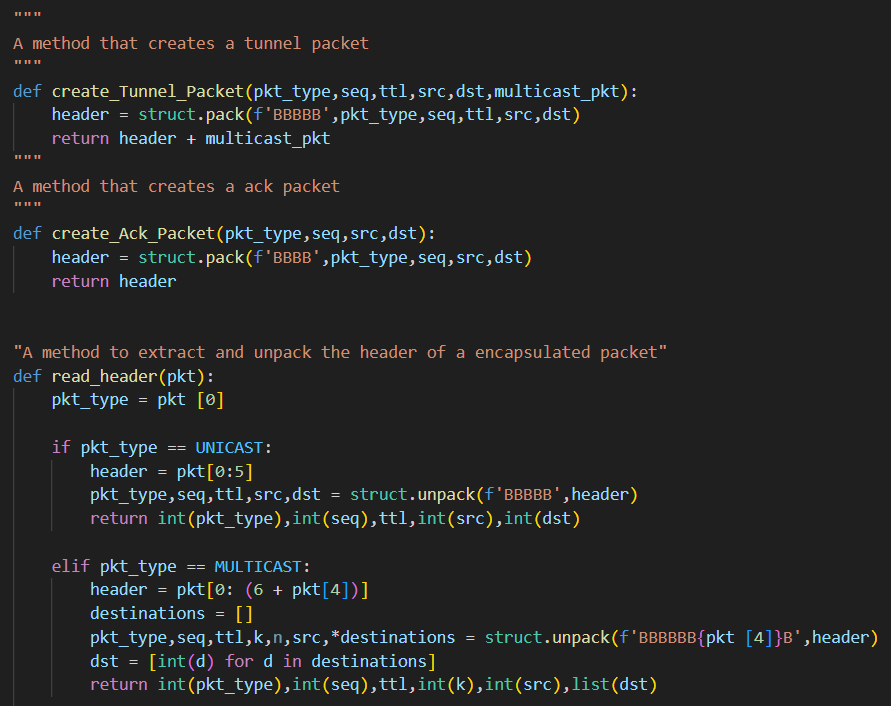


## **2.5 cleanup.py**

This file is responsible for cleaning all the left-behind processes and terminating all ssh connections to nodes. In short, freeing any resources that still may be running in the background even after the code has finished.

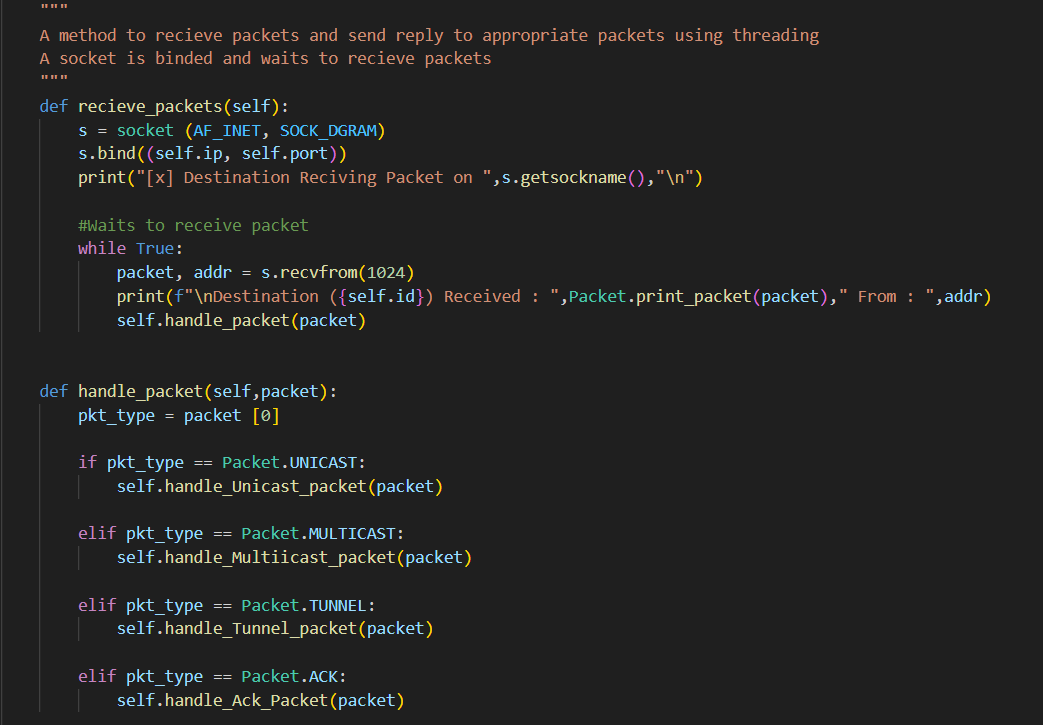
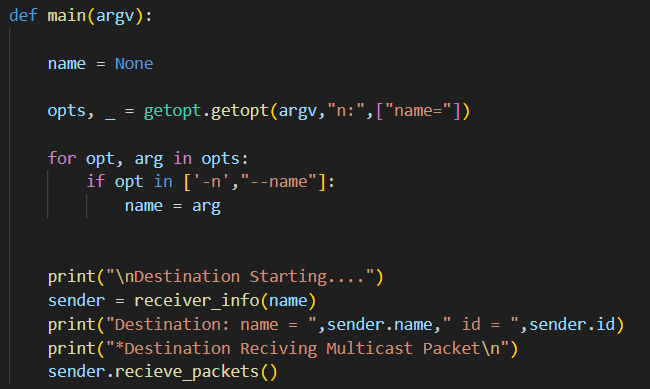
## **2.6 packet.py**

This file is responsible for the creation of all different types of packets and also implements a method to unpack encapsulated packets and read their content. Also, we added a feature where the header of the multicast packet can dynamically change based on the number of destinations to allow for future implementation to use more than three destinations.



## **2.7 destination.py**

This file initializes a node as a destination. Where it takes the name of the node as an argument, which allows the node to look up all its information from the network initialized in “NetworkMap.py” and store it internally. Once it is finished setting up it will start waiting to receive appropriate packets and handle them accordingly.



## **2.8 source.py**

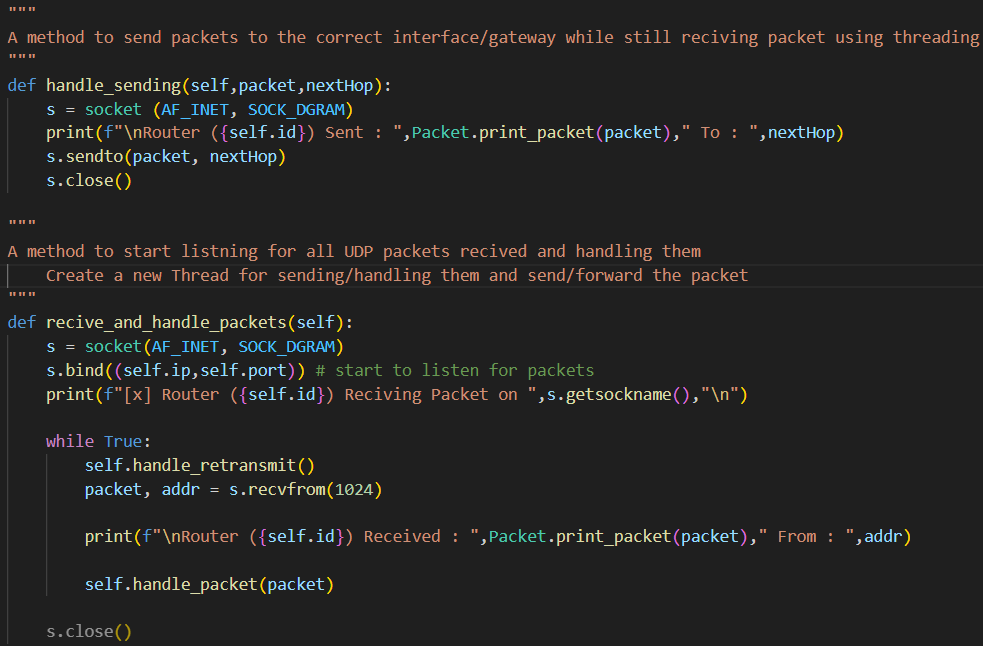
This file initializes a node as a source. Similar to the destination it takes all the necessary arguments to set up the source and creation of the packet. Once it is finished setting up it will send the created packet. However, it allows the user to choose between two options: sending the multicast packet or sending the multicast packet in a tunnel to the initial RP to see how the network behaves. The tunnel to the initial RP is made using the spec doc, on the hand multicast packet is sent into the network and the first router to receive it handles tunneling it to the initial RP.

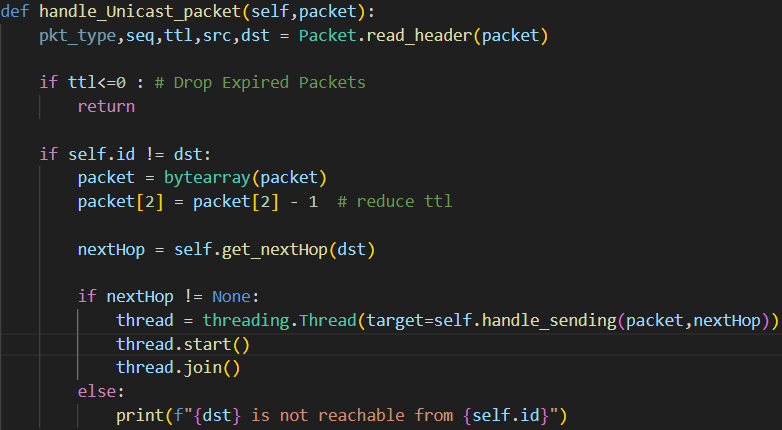




## **2.9 router.py**

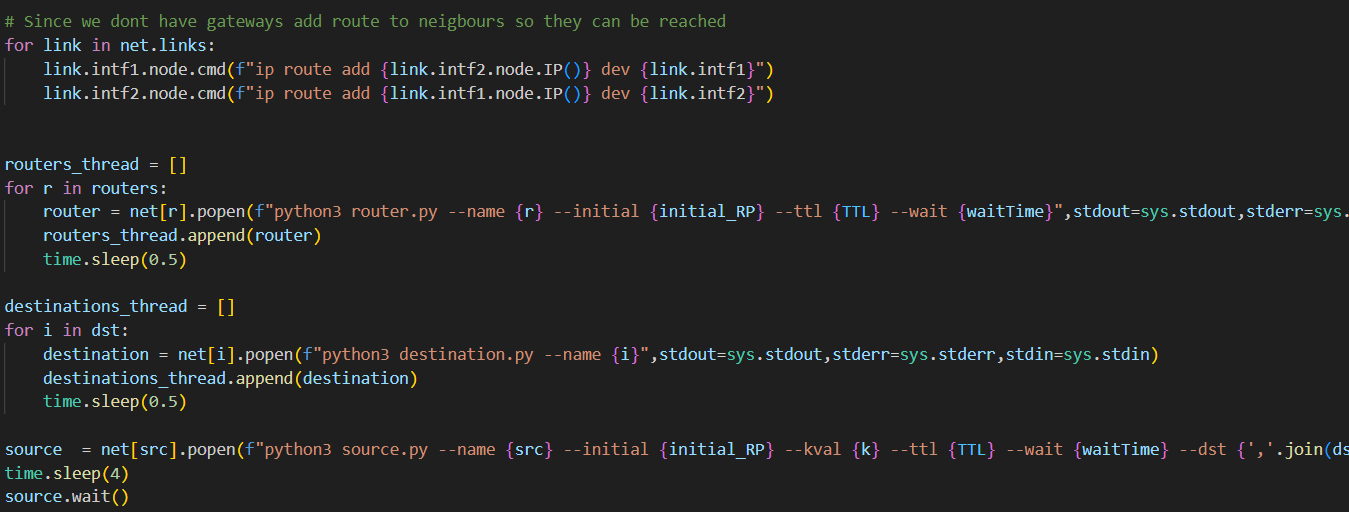
This file initializes a node as a router. Similar to the destination it takes all the necessary arguments to set up the router. Once it is finished setting up it will start waiting for the packet. Once it receives a packet it will be responsible for handling the packets appropriately. Such as when receiving a packet not meant for the node forwarding it through the correct gateway, sending ACK packets when receiving a packet meant for the node, reducing the TTL or max hops of the packets, etc.

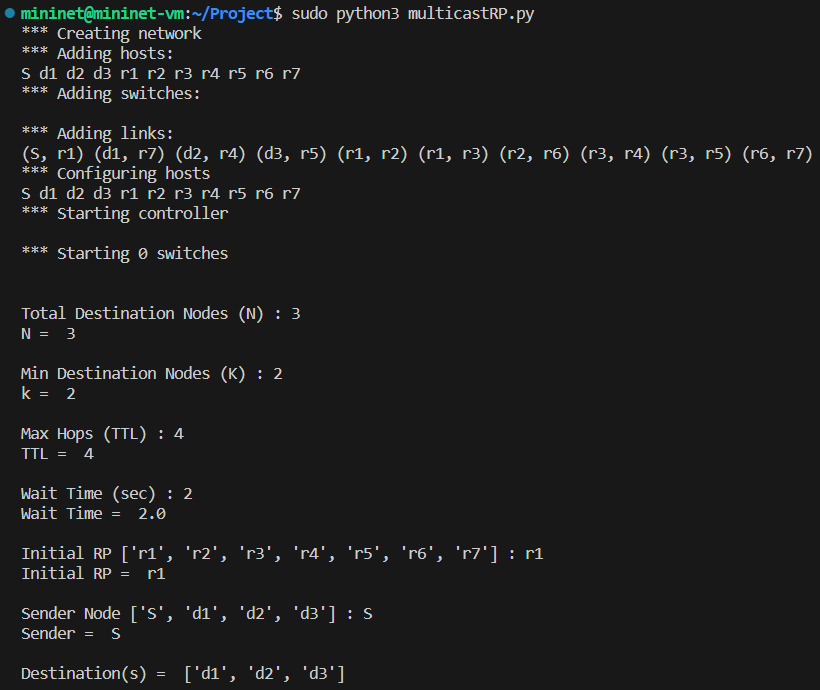


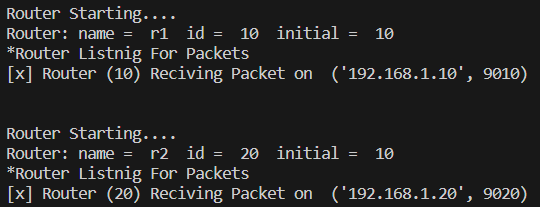


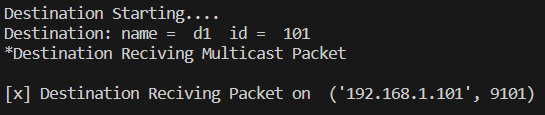
## **2.10 multicastRP.py**

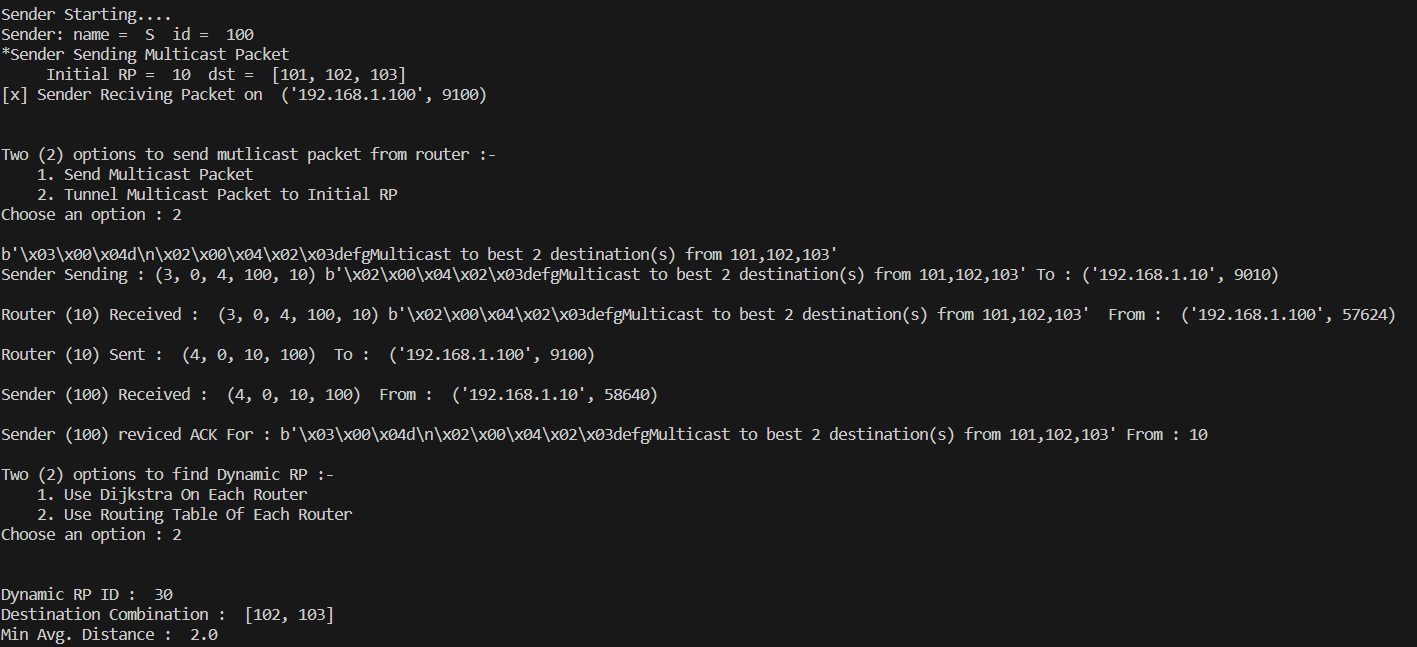
This file connects and utilizes all the other files to initialize the network and passes the appropriate field for each node and their respective Python file to start the correct role of either router, source, or destination on the node. Thus once a user has set up the topology in the “Topo.py” using the custom cls all they have to run is the command “sudo python3 multicastRP.py” while making sure to navigate to the folder that holds all the files in one place.



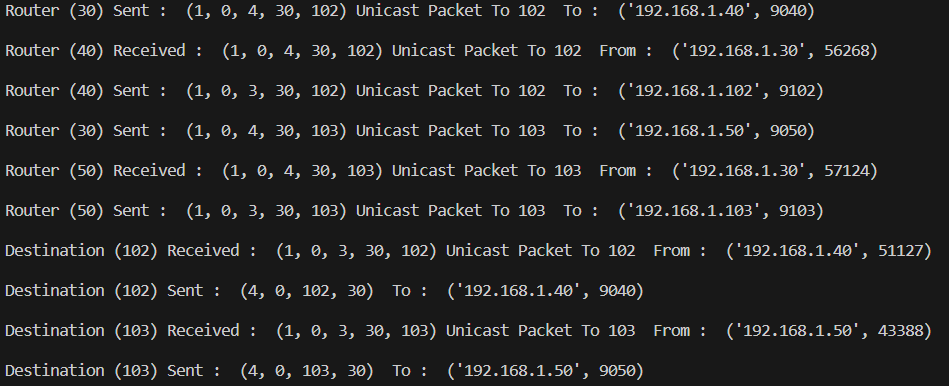




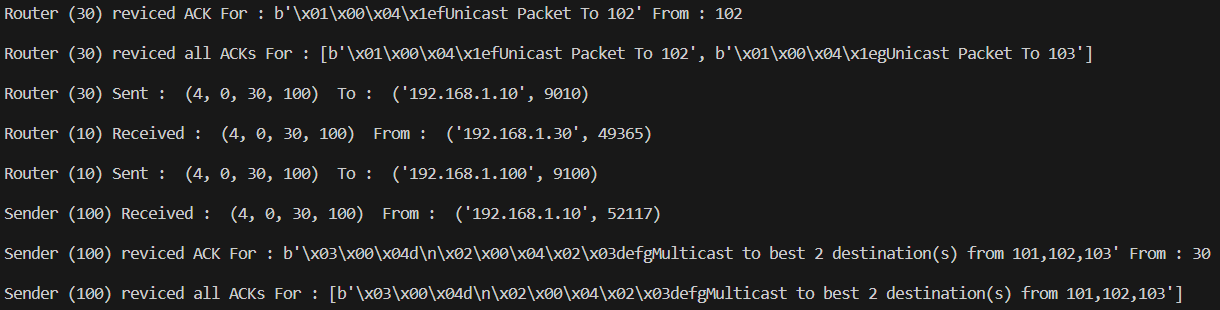




**Fig. Initial RP receiving a tunnel packet from source**



**Fig. Dynamic RP sending a unicast packet to destinations**



**Fig. Dynamic RP receiving all ACKs for unicast packet and sending ACK to source**

# 

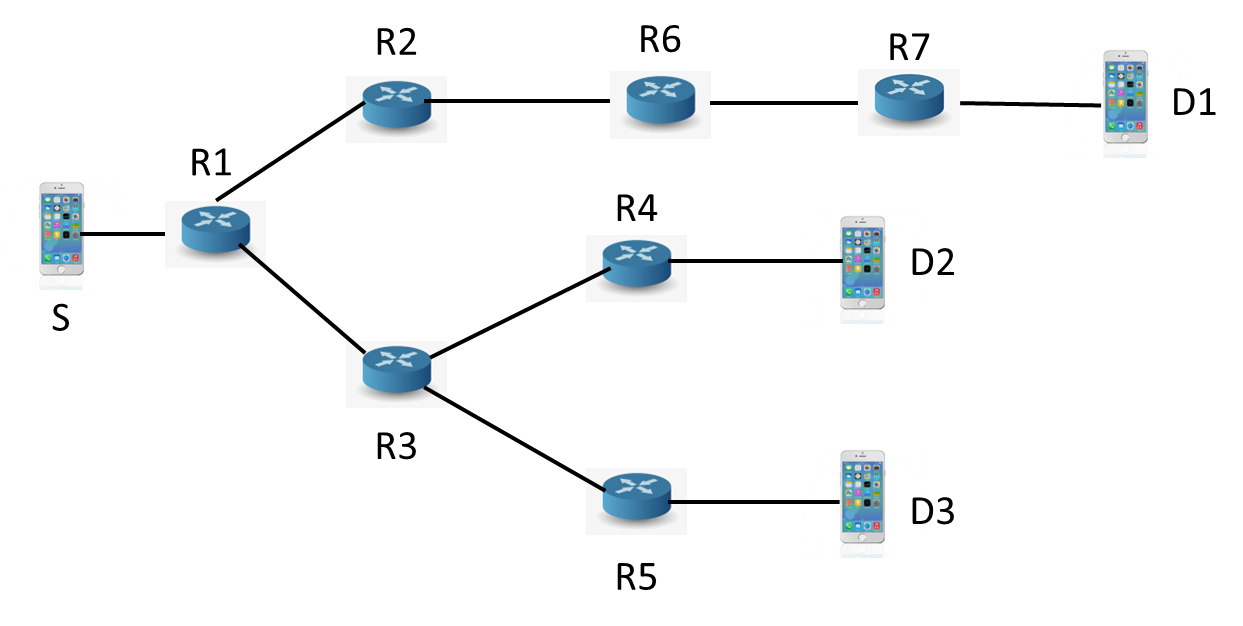
# 

# **Conclusion**

## **3.1 Accomplishments & Evaluation**

We were successfully able to implement the spec doc. Not only that we added a few more improvements, but we also kept all the low-level items such as packet format, routing algorithm, etc the same. However, we did add a source (src) field in the Multicast packet, to allow for a better reliability scheme of Sender (S) receiving the appropriate ACK in time. This addition was due to using end-to-end ACK messages. Thus, our group decided that Initial RP should receive an ack for sending the tunnel packet to dynamic RP, and if it does not receive an ack in a set amount of time (waitTime) it will retransmit the packet. For these to happen, the source of the tunnel packet must be the initial RP’s ID. Leading to the addition of a source field in the multicast packet to keep track of the original sender. We also suggest a different technique for finding the Dynamic RP, instead of rerunning Dijkstra on each router. When the network is initialized, each router uses LSA to update its routing tables, which are created using Dijkstra. We can utilize this created routing table that holds the pair (destination, nextHop, Cost). Thus, if we utilize the reading of the routing table instead of rerunning Dijkstra we save a lot of compute resources on large networks.

**Given Topology for Implementation:**

****