(PDF) method.pdf: Description of the approaches to construct the topology.

We set up the code in a way that it'll run traceroute using threading, so it'll run multiple traceroutes in an address space. In a portion of what one member were to traceroute, it would divide up the subrange of addresses assigned to each group member across all threads evenly. To record data for the process each traceroute outputted individually, we broke it down in steps.

- Record hops from a single traceroute output as a list of IP addresses.
- Create a Python dictionary object for each IP in the list, using two attributes: the IP address of the hop and a list of IP addresses that the hop is linked to.
- Essentially, for every hop to a valid IP address, record the address of the adjacent hop, which records the link to all IP addresses.
- Implement the traceroute using Scapy, with packets for traceroute using the EDP protocol.
- Once the data is collected, store it in an AWS cloud-based JSON database using MongoDB.
- Generate the graph using NetworkX, a Python library.
- To generate the graph, read all JSON objects from the MongoDB database to create a graph of all nodes and IP links between the nodes.
- Run the traceroute program in UGL, LKD, Douglass, HSL, and Blackburn.

Makes use of the data collected within the mongoDB database to generate a graph using the networkX library. Code will access a collection within the database called IP_Nodes. It then creates an empty undirected graph using the "networkx.graph" function and iterates through the documents in the IP_Nodes collection to construct the graph by adding an edge for each link that is a hop to a new IP address. Finally it draws a graph using the matplotlib library.

TRACEROUTE.PY FUNCTIONS

def traceroute(destination, max_hops=30, dst_port=33434):

This function performs a traceroute to a given destination IP address using Scapy, a Python library for network packet manipulation. It takes the following parameters:

- `destination`: The IP address of the target destination.
- `max_hops` (optional): The maximum number of hops or intermediate routers to probe before stopping the traceroute (default is 30).
- `dst_port` (optional): The destination port number used for sending UDP packets in the traceroute (default is 33434).

It returns a list of IP addresses representing the route taken to reach the destination. If the destination is reached, the list will include the destination IP.

def ip_is_valid(ip):

This function checks if an IP address belongs to a specific set of IP address ranges. It takes the following parameter:

- `ip`: The IP address to check.

It returns `True` if the IP address is in the specified ranges, and `False` otherwise.

def increase_ip(ip, dif, increments=1):

This function increments an IP address by a specified value and number of increments. It takes the following parameters:

- `ip`: The IP address to be incremented.
- `dif`: The value by which the IP address should be incremented.
- `increments` (optional): The number of times to increment the IP (default is 1).

It returns the modified IP address as a string.

def multi_traceroute(first_destination, max_hops, count, nodes):

This function performs multiple traceroutes to a range of IP addresses and updates information about the nodes in a MongoDB database. It takes the following parameters:

- `first_destination`: The initial IP address to start the traceroutes.
- `max_hops`: The maximum number of hops for each traceroute.
- `count`: The number of traceroutes to perform.
- `nodes`: A MongoDB collection where information about the nodes is stored.

This function iterates through a range of IP addresses and performs traceroutes. It updates or inserts information about each node in the MongoDB collection.

def mongo_client():

This function sets up a connection to a MongoDB database and returns a client object. It uses credentials to connect to a MongoDB Atlas cluster and sends a ping to confirm a successful connection.

It returns a MongoDB client object that can be used to interact with the database.