Network Algorithms Final Project

Graph representation using Python and NetworkX

Shamkhal Guliyev | shamkhal.guliyev@ufaz.az | 22022745
Yusif Hajizade | yusif.hajizade@ufaz.az | 22022735

Introduction

This project delves into the exploration and analysis of network traffic data using Python and the NetworkX library. Leveraging a comprehensive dataset, the aim is to unveil hidden relationships, visualize communication patterns, and gain a deeper understanding of the dynamics within the network. From attribute exploration to graph representation and path finding, this project offers a comprehensive journey into understanding and interpreting intricate network interactions.

Project Objectives

The project utilizes key Python libraries, including Pandas for data manipulation, Matplotlib for visualization, and NetworkX for graph-based analysis. Through a step-by-step we present objectives of the project

1. **Attribute Exploration:**
2. **Graph Visualization**
3. **Path Finding**
4. **Packet Calculation**
5. **Centrality Metrics**
6. **Interactive Visualization**
7. **Color Coding**
8. **Automated Analysis**
9. **Network Density and Diameter**
10. **Network Average Path Length**

Part 1: Data Description and Exploration

Part 1 of the project involves delving into the network traffic dataset. This includes reading the dataset from CSV files, describing each attribute succinctly, and establishing relationships between attributes. Additionally, statistical tools are applied to represent the data visually using charts. This initial exploration lays the groundwork for subsequent in-depth analyses and visualizations.

Dataset Loading

This step involves loading the network traffic dataset from CSV files. The dataset was loaded using the assign_attributes function, and the columns were mapped to attributes such as "pkSeqID," "Stime," "Flgs," "Proto," "Saddr," "Sport," "Daddr," "Dport," and others.

Attribute Description

Each attribute within the dataset is described in simple terms. This step aims to provide a clear understanding of the dataset's structure, laying the foundation for subsequent analyses and interpretations.

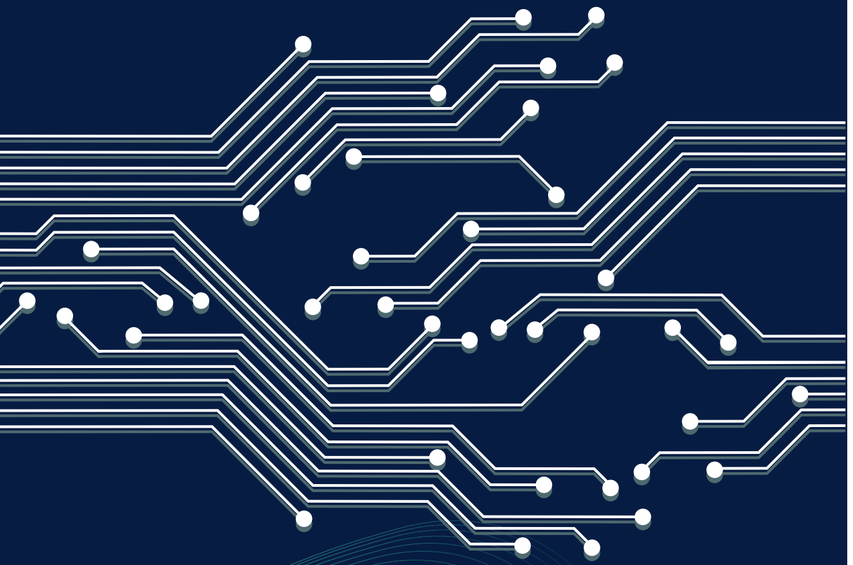
3 Attribute Relations

The relationships between attributes are explored and articulated. This step involves identifying connections, dependencies, and correlations among different attributes, offering insights into the inherent structure of the network traffic data.

Part 1: Dataset Loading

```python
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt

# Assigns attributes to columns in data
def assign_attributes(file_name):
  attributes = [
  "ID", "STime", "Flags", "Protocol", "SourceIP", "SourcePort", "DestinationIP", "DestinationPort",
  "Packets", "Bytes", "State", "LastTime", "Sequence", "Duration", "Mean", "Stddev", "Min", "Max",
  "Spkts", "Dpkts", "Sbytes", "Dbytes", "Rate", "Srate", "Drate", "TnBPSrcIP", "TnBPDstIP", "TnP_PSrcIP",
  "TnP_PDstIP", "AR_P_Proto_P_Sport", "Pkts_P_State_P_Protocol_P_DestIP", "Pkts_P_State_P_Protocol_P_SrcIP",
  "Attack", "Category", "Subcategory"
  ]

  raw_data = pd.read_csv(file_name, low_memory = False)
  raw_data.columns = attributes
  return raw_data

formatted_data = assign_attributes("UNSW_2018_IoT_Botnet_Dataset_1.csv")

# ---------- Part 1 ----------
correlation_matrix = formatted_data.corr()
print(correlation_matrix)
```

Part 1: Attribute Description



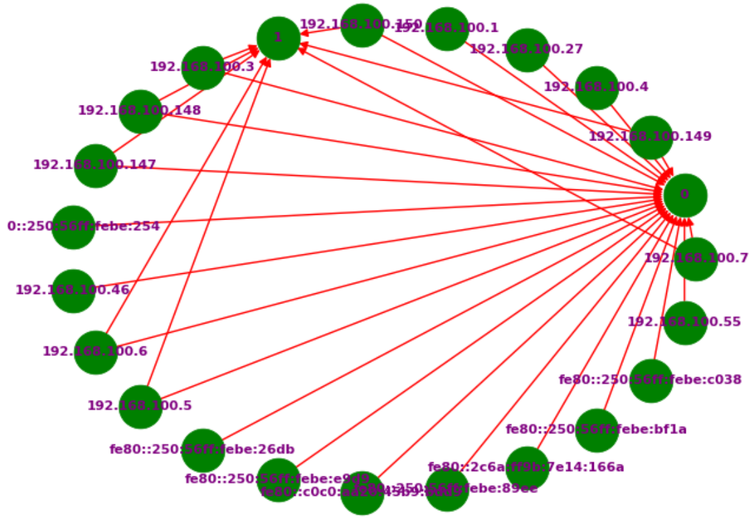
1. **pkSeqID (Row Identifier): Unique identifier for each row in the datas**
2. **Stime (Record start time): The timestamp when the network activity started.**
3. **Flgs (Flow state flags seen in transactions): Flags indicating the state of the network flow (e.g., SYN, ACK).**
4. **Flgs_number (Numerical representation of feature flags): Numeric representation of flow state flags.**
5. **Proto (Textual representation of transaction protocols present in the network flow): The protocol used in the network transaction (e.g., TCP, UDP).**
6. **Proto_number (Numerical representation of feature protocol): Numeric representation of the transaction protocol.**
7. **Saddr (Source IP address): IP address of the source in the network transaction.**
8. **Sport (Source port number): Port number on the source side of the transaction.**
9. **Daddr (Destination IP address): IP address of the destination in the network transaction.**
10. **Dport (Destination port number): Port number on the destination side of the transaction.**
11. **Pkts (Total count of packets in the transaction): The number of packets exchanged in the network transaction.**
12. **Bytes (Total number of bytes in the transaction): The total size of data exchanged in the network transaction.**
13. **State (Transaction state): The state of the network transaction (e.g., Connection Established).**
14. **State_number (Numerical representation of feature state): Numeric representation of the transaction state.**
15. **Ltime (Record last time): The timestamp when the last activity related to the record occurred.**
16. **Seq (Argus sequence number): Sequence number assigned by Argus.**
17. **Dur (Record total duration): The total duration of the network transaction.**
18. **Mean (Average duration of aggregated records): The average duration of all aggregated records.**
19. **Stddev (Standard deviation of aggregated records): The measure of variability among aggregated records' durations.**
20. **Min (Minimum duration of aggregated records): The shortest duration among aggregated records.**
21. **Max (Maximum duration of aggregated records): The longest duration among aggregated records.**
22. **Spkts (Source-to-destination packet count): The count of packets sent from the source to the destination.**
23. **Dpkts (Destination-to-source packet count): The count of packets sent from the destination to the source.**
24. **Sbytes (Source-to-destination byte count): The count of bytes sent from the source to the destination.**
25. **Dbytes (Destination-to-source byte count): The count of bytes sent from the destination to the source.**
26. **Rate (Total packets per second in the transaction): The rate of packets exchanged per second in the network transaction.**
27. **Srate (Source-to-destination packets per second): The rate of packets sent from the source to the destination per second.**
28. **Drate (Destination-to-source packets per second): The rate of packets sent from the destination to the source per second.**
29. **TnBPSrcIP (Total Number of bytes per source IP): The total number of bytes transmitted per source IP.**
30. **TnBPDstIP (Total Number of bytes per Destination IP): The total number of bytes transmitted per destination IP.**
31. **TnP_PSrcIP (Total Number of packets per source IP): The total number of packets transmitted per source IP.**
32. **TnP_PDstIP (Total Number of packets per Destination IP): The total number of packets transmitted per destination IP.**
33. **TnP_PerProto (Total Number of packets per protocol): The total number of packets transmitted per protocol.**
34. **TnP_Per_Dport (Total Number of packets per destination port): The total number of packets transmitted per destination port.**
35. **AR_P_Proto_P_SrcIP (Average rate per protocol per Source IP): The average rate of packets per second for a specific protocol and source IP.**
36. **AR_P_Proto_P_DstIP (Average rate per protocol per Destination IP): The average rate of packets per second for a specific protocol and destination IP.**
37. **N_IN_Conn_P_SrcIP (Number of inbound connections per source IP): The count of inbound connections for a specific source IP.**
38. **N_IN_Conn_P_DstIP (Number of inbound connections per destination IP): The count of inbound connections for a specific destination IP.**
39. **AR_P_Proto_P_Sport (Average rate per protocol per source port): The average rate of packets per second for a specific protocol and source port.**
40. **AR_P_Proto_P_Dport (Average rate per protocol per destination port): The average rate of packets per second for a specific protocol and destination port.**
41. **Pkts_P_State_P_Protocol_P_DestIP (Number of packets grouped by state of flows and protocols per destination IP): The number of packets grouped by the sta destination IP.**
42. **Pkts_P_State_P_Protocol_P_SrcIP (Number of packets grouped by state of flows and protocols per source IP): The number of packets grouped by the state of fl**
43. **Attack (Class label: 0 for Normal traffic, 1 for Attack Traffic): Indicates whether the network traffic is normal (0) or an attack (1).**
44. **Category (Traffic category): The general category of the network traffic (e.g., Normal, Attack).**
45. **Subcategory (Traffic subcategory): A more specific subcategory of the network traffic.**

Part 1: Attribute Relations

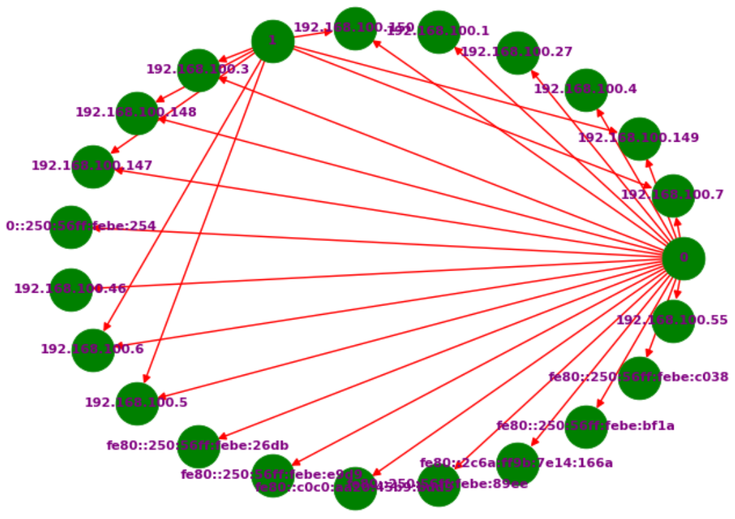
	ID	STime	...	Pkts_P_State_P_Protocol_P_SrcIP	Attack
ID	1.000000	0.685501	...	0.014947	0.031183
STime	0.685501	1.000000	...	0.014070	0.055247
Packets	-0.003328	-0.005368	...	-0.000213	-0.257240
Bytes	-0.002740	-0.004328	...	-0.000855	-0.231486
LastTime	0.685502	1.000000	...	0.014069	0.055203
Sequence	0.306167	0.380580	...	0.177663	0.038614
Duration	-0.050665	-0.080030	...	-0.004510	-0.206986
Mean	-0.075512	-0.151341	...	-0.016876	-0.120998
Stddev	-0.053546	-0.095405	...	-0.003999	-0.021728
Min	NaN	NaN	...	NaN	NaN
Max	NaN	NaN	...	NaN	NaN
Spkts	-0.004971	-0.008130	...	-0.001546	-0.313296
Dpkts	-0.060505	-0.126432	...	-0.016797	-0.097839
Sbytes	-0.080211	-0.156963	...	-0.015399	-0.108625
Dbytes	NaN	NaN	...	NaN	NaN
Rate	NaN	NaN	...	NaN	NaN
Srate	NaN	NaN	...	NaN	NaN
Drate	NaN	NaN	...	NaN	NaN
TnBPSrcIP	-0.003614	-0.005757	...	-0.000587	-0.281676
TnBPDstIP	-0.002113	-0.003520	...	0.000421	-0.159694
TnP_PSrcIP	-0.002900	-0.004536	...	-0.000953	-0.252590
TnP_PDstIP	-0.001999	-0.003223	...	-0.000554	-0.158031
AR_P_Proto_P_Sport	-0.028427	-0.016769	...	0.135738	0.005731
Pkts_P_State_P_Protocol_P_DestIP	0.006764	0.004743	...	0.693289	-0.050382
Pkts_P_State_P_Protocol_P_SrcIP	0.014947	0.014070	...	1.000000	0.004376
Attack	0.031183	0.055247	...	0.004376	1.000000

[26 rows x 26 columns]

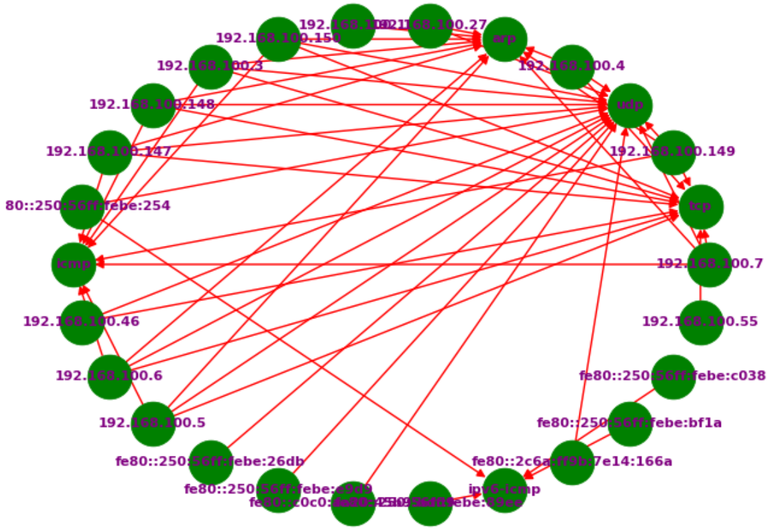
First representation



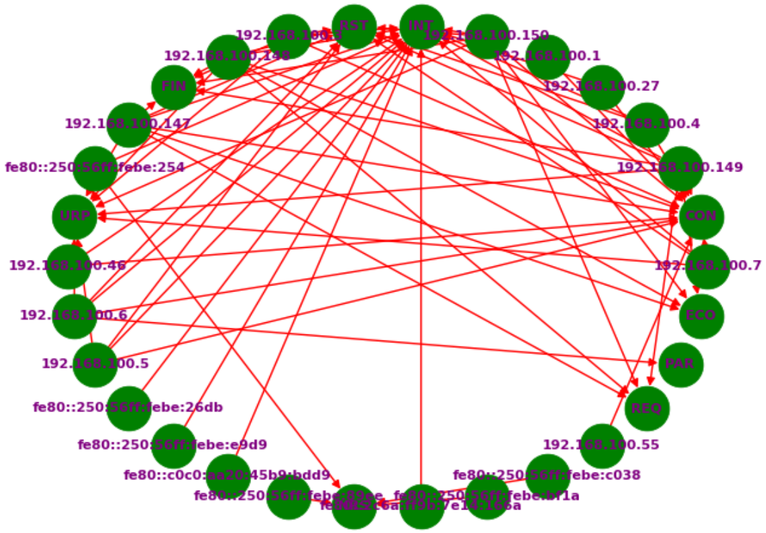
Second representation



Third representation



Fourth representation



Part 2: Graph Representation

In this section, we employ the NetworkX library to present the network traffic data in graph format. Four distinct representations are created, each offering unique insights into the relationships within the dataset. Graph representations were created with 5 different attributes as nodes and edges. These representations allow visualizing network traffic patterns based on source IPs, attack labels, protocols, and user-defined attributes.

Part 2: Graph Representation Code

```
# ---------- Part 2: Representing the graph using different nodes and edges ----------
# Plots a graph with given data, and node and edge attributes
def representation(data, node, edge):
    Graph = nx.from_pandas_edgelist(data, node, edge, create_using = nx.DiGraph())
    pos = nx.circular_layout(Graph)
    nx.draw(Graph, pos, with_labels = True, font_weight = "bold", node_size = 700, node_color = "green",
            font_size = 8, edge_color = "red", linewidths = 0.5, font_color = "purple")
    plt.show()

representation(formatted_data, "SourceIP", "Attack") # First representation
representation(formatted_data, "Attack", "SourceIP") # Second representation
representation(formatted_data, "SourceIP", "Protocol") # Third representation

# Fourth representation
user_node = input("Enter the attribute that will be used as a node: ") # For example: SourceIP
user_edge = input("Enter the attribute that will be used as an edge: ") # For example: State
representation(formatted_data, user_node, user_edge)
```

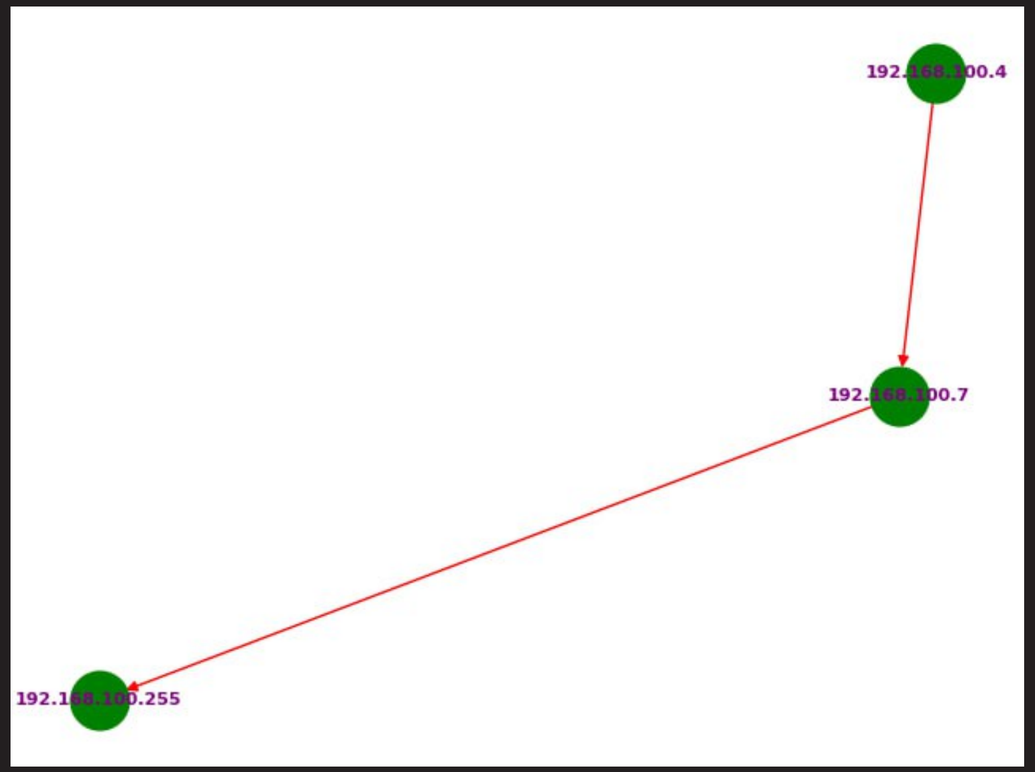
Part 3: Implementation of DFS

In Part 3, the project employs Depth-First Search (DFS) to identify and visualize paths within the network. The algorithm facilitates the exploration of communication routes between specific nodes. Additionally, the project evaluates the total size of sent packets along these paths, providing a quantitative measure of data exchange.

```
Path from 192.168.100.4 to 192.168.100.255: 192.168.100.4 -> 192.168.100.7 -> 192.168.100.255
Total number of sent packets: 8
```

```
Path from 192.168.100.150 to 192.168.217.2: 192.168.100.150 -> 192.168.217.2
Total number of sent packets: 2
```

```
Path from 192.168.100.150 to 192.168.217.2: 192.168.100.150 -> 192.168.217.2
Total number of sent packets: 2
```



Part 3: Visualized plot of the graph of paths

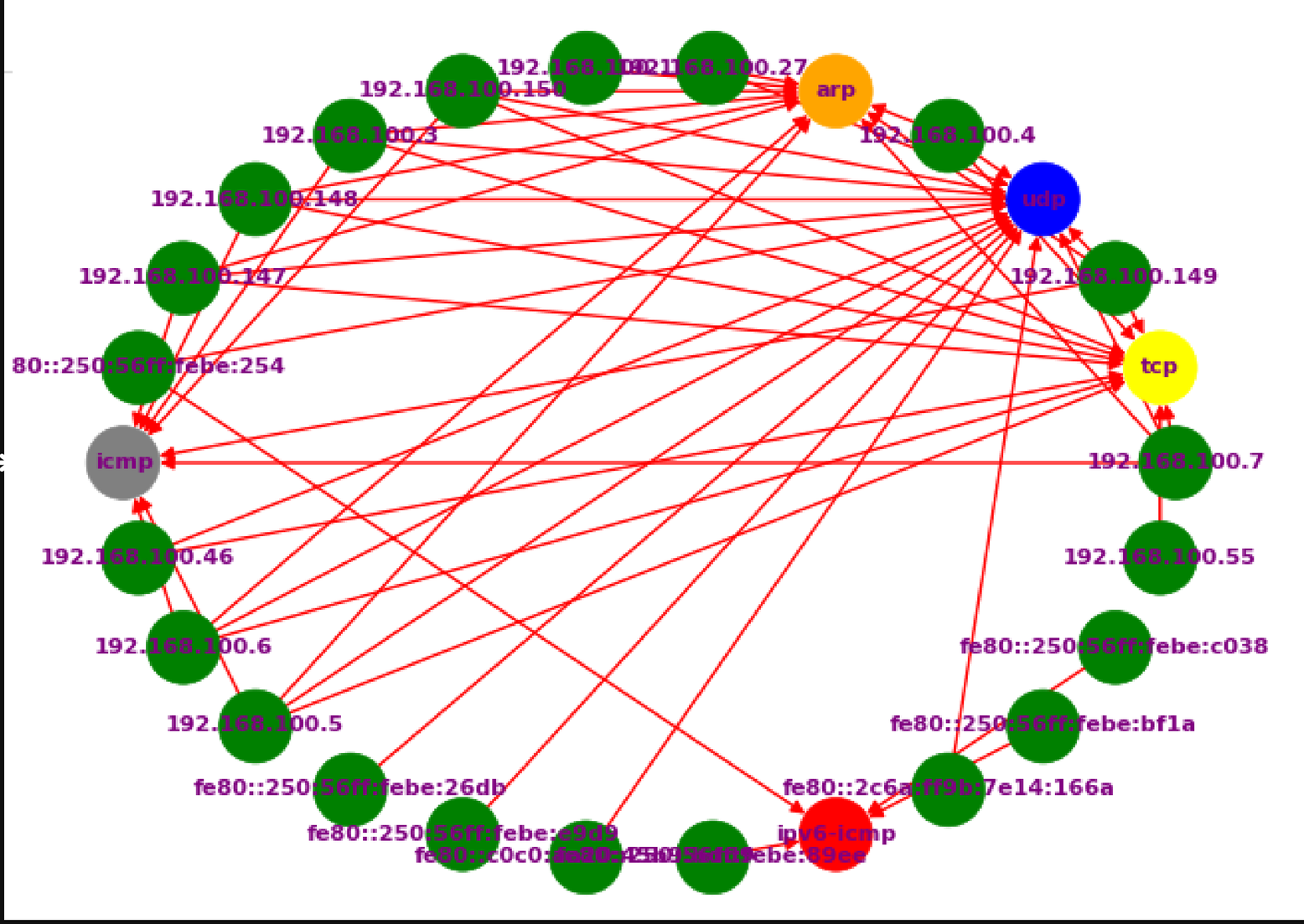
Part 3: Implementation of DFS Code

```python
# ---------- Part 3: DFS and calculating packets ----------
def find_path(Graph, start, end):
    path = nx.shortest_path(Graph, source = start, target = end)
    return path

def calculate_packets(Graph, path):
    packets = 0

    for i in range(len(path) - 1):
        origin = path[i]
        destination = path[i + 1]

        if Graph.has_edge(origin, destination):
            packets += Graph[origin][destination]["Packets"]
        else:
            print(f"Warning: No direct edge between {origin} and {destination}")

    return packets

Graph = nx.from_pandas_edgelist(formatted_data, "SourceIP", "DestinationIP", ["Attack",
                                create_using = nx.DiGraph())
graph_dict = dict(Graph.adjacency())

origin = "192.168.100.150"
destination = "192.168.217.2"
path = find_path(Graph, origin, destination)

print("\n", end = "")
if path:
    print(f"Path from {origin} to {destination}: ", end = "")
    for ip in path:
        if ip != path[-1]:
            print(f"{ip} -> ", end = "")
        else:
            print(ip)

    packets = calculate_packets(Graph, path)
    print(f"Total number of sent packets: {packets}")
else:
    print(f"No path found from {origin} to {destination}")
print("\n", end = "")

# Drawing the graph of the path
path_edges = [(path[i], path[i + 1]) for i in range(len(path) - 1)]
path_graph = Graph.edge_subgraph(path_edges)
pos = nx.spring_layout(path_graph)
nx.draw(path_graph, pos, with_labels=True, font_weight="bold", font_size = 8, node_color="green",
        node_size = 700, edge_color = "red", font_color = "purple")
plt.show()
```

Part 4: Graph Coloring

Explore the world of network traffic through visualizations. Part 4 introduces graph representations using Python's NetworkX library, showcasing various perspectives. From IP-centric views to user-defined attributes, dynamic graphs reveal intricate connections and patterns within the dataset.

Graphs were colored based on different protocols, facilitating differentiation in the visualization. Users can now automate the coloring process by inputting attributes for nodes and coloring, enhancing customization and interpretation




```
Enter the color for the protocol arp: yellow
Enter the color for the protocol tcp: blue
Enter the color for the protocol udp: orange
Enter the color for the protocol icmp: gray
Enter the color for the protocol ipv6-icmp: red
```

Part 4: Graph Coloring Code

```
86  # ---------- Part 4: Differentiating protocols ----------
87  def representation_with_colors(data, node, edge, colors):
88      Graph = nx.from_pandas_edgelist(data, node, edge, create_using = nx.DiGraph())
89
90      source_ip_colors = {ip: color for ip, color in zip(data["Protocol"].unique(), colors)}
91      node_colors = [source_ip_colors.get(ip, "green") for ip in Graph.nodes()]
92
93      pos = nx.circular_layout(Graph)
94      nx.draw(Graph, pos, with_labels = True, font_weight = "bold", node_size = 700, node_color = node_colors,
95          edge_color = "red", font_size = 8, linewidths = 0.5, font_color = "purple")
96
97      plt.show()
98
99  colors = ["", "", "", "", ""] # For example: yellow, blue, orange, gray, red
100 protocols = ["arp", "tcp", "udp", "icmp", "ipv6-icmp"]
101
102 for index in range(5):
103     color = input(f"Enter the color for the protocol {protocols[index]}: ")
104     colors[index] = color
105
106 representation_with_colors(formatted_data, "SourceIP", "Protocol", colors)
```

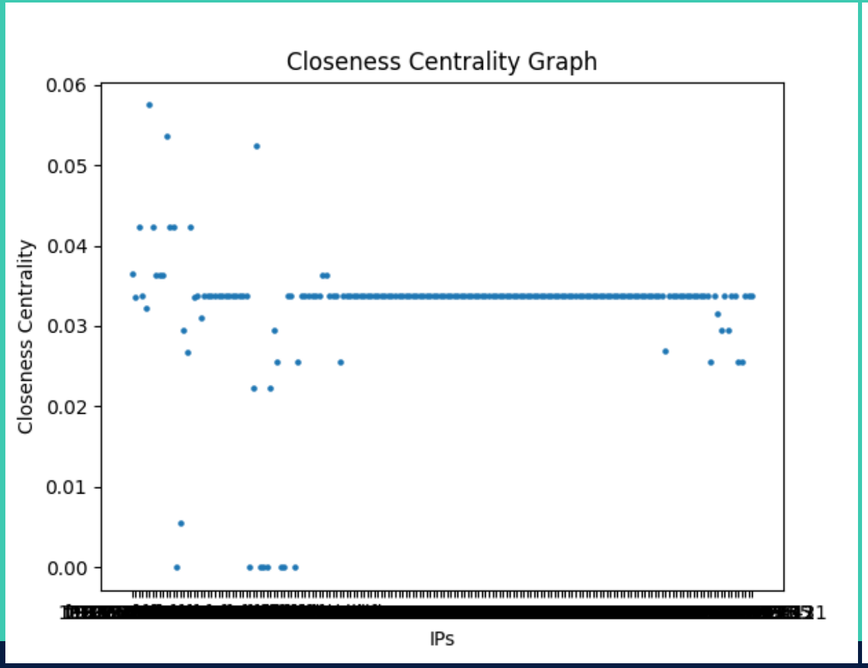
Part 5: Graph Analysis, Network Analysis Metrics

Building upon network analysis metrics, the project meticulously calculated and analyzed vital parameters—Degree Connectivity, Closeness Centrality, Betweenness Centrality, Network Density, Network Diameter, and Average Path Length. These metrics serve as windows into the structural intricacies and defining characteristics of the network, enriching our understanding of its architecture and functionality.

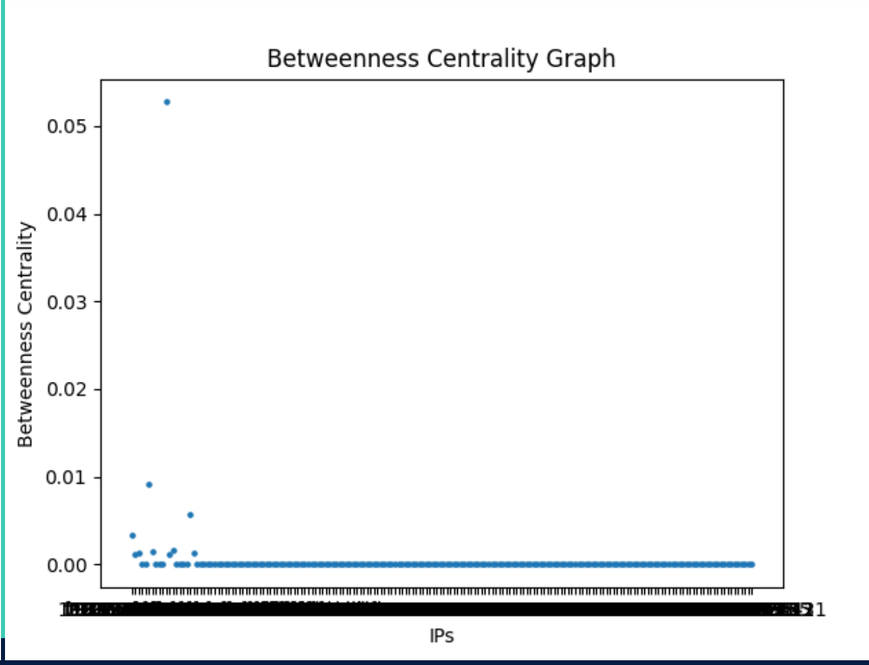
Degree Connectivity:	{'192.168.100.7': 0.10810810810810811, '192.168.100.4': 0.05405405405405406, '192.168.100.149': 0.16216216216216217, '27.124.125.250': 0.02702702702702703, '192.168.100.27': 0.05405405405405406, '192.168.100.1': 0.16216216216216217, '192.168.100.150': 0.1891891891891892, '192.168.217.2': 0.10810810810810811, '192.168.100.255': 0.08108108108108109, '8.8.8.8': 0.10810810810810811, '192.168.100.3': 0.7297297297297298, '192.168.100.148': 0.13513513513513514, '192.168.100.147': 0.13513513513513514, 'fe80::250:56ff:febe:254': 0.02702702702702703, 'ff02::1': 0.02702702702702703, '255.255.255.255': 0.02702702702702703, '192.168.100.46': 0.02702702702702703, '192.168.100.5': 0.08108108108108109, '192.168.100.6': 0.05405405405405406, '13.55.154.73': 0.02702702702702703, '192.168.100.55': 0.02702702702702703, '184.85.248.65': 0.02702702702702703, '205.251.194.167': 0.02702702702702703, '192.5.5.241': 0.02702702702702703, '192.35.51.30': 0.02702702702702703, '205.251.195.97': 0.02702702702702703, '205.251.196.236': 0.02702702702702703, '199.7.91.13': 0.02702702702702703, '192.12.94.30': 0.02702702702702703, '52.35.35.13': 0.02702702702702703, '216.239.38.10': 0.02702702702702703, '199.7.83.42': 0.02702702702702703, '192.42.93.30': 0.02702702702702703, '172.217.25.170': 0.02702702702702703, 'fe80::250:56ff:febe:26db': 0.02702702702702703, 'ff02::fb': 0.05405405405405406, '224.0.0.251': 0.05405405405405406, 'fe80::250:56ff:febe:e9d9': 0.02702702702702703}
Closeness Centrality:	{'192.168.100.7': 0.02702702702702703, '192.168.100.4': 0.02702702702702703, '192.168.100.149': 0.11522048364153627, '27.124.125.250': 0.09319664492078286, '192.168.100.27': 0.09618441971383149, '192.168.100.1': 0.13626126126126126, '192.168.100.150': 0.11522048364153627, '192.168.217.2': 0.11750881316098707, '192.168.100.255': 0.11750881316098707, '8.8.8.8': 0.11750881316098707, '192.168.100.3': 0.19901719901719905, '192.168.100.148': 0.11522048364153627, '192.168.100.147': 0.11522048364153627, 'fe80::250:56ff:febe:254': 0.0, 'ff02::1': 0.02702702702702703, '255.255.255.255': 0.09319664492078286, '192.168.100.46': 0.0, '192.168.100.5': 0.02702702702702703, '192.168.100.6': 0.0, '13.55.154.73': 0.1287001287001287, '192.168.100.55': 0.1287001287001287, '184.85.248.65': 0.1287001287001287, '205.251.194.167': 0.1287001287001287, '192.5.5.241': 0.1287001287001287, '192.35.51.30': 0.1287001287001287, '205.251.195.97': 0.1287001287001287, '205.251.196.236': 0.1287001287001287, '199.7.91.13': 0.1287001287001287, '192.12.94.30': 0.1287001287001287, '52.35.35.13': 0.1287001287001287, '216.239.38.10': 0.1287001287001287, '199.7.83.42': 0.1287001287001287, '192.42.93.30': 0.1287001287001287, '172.217.25.170': 0.1287001287001287, 'fe80::250:56ff:febe:26db': 0.0, 'ff02::fb': 0.05405405405405406, '224.0.0.251': 0.1422475106685633, 'fe80::250:56ff:febe:e9d9': 0.0}
Betweenness Centrality	{'192.168.100.7': 0.02102102102102102, '192.168.100.4': 0.0, '192.168.100.149': 0.01126126126126126, '27.124.125.250': 0.0, '192.168.100.27': 0.0, '192.168.100.1': 0.007507507507507507, '192.168.100.150': 0.01576576576576576, '192.168.217.2': 0.0, '192.168.100.255': 0.0, '8.8.8.8': 0.0, '192.168.100.3': 0.1614114114114114, '192.168.100.148': 0.0045045045045045045, '192.168.100.147': 0.0045045045045045045, 'fe80::250:56ff:febe:254': 0.0, 'ff02::1': 0.0, '255.255.255.255': 0.0, '192.168.100.46': 0.0, '192.168.100.5': 0.02102102102102102, '192.168.100.6': 0.0, '13.55.154.73': 0.0, '192.168.100.55': 0.0, '184.85.248.65': 0.0, '205.251.194.167': 0.0, '192.5.5.241': 0.0, '192.35.51.30': 0.0, '205.251.195.97': 0.0, '205.251.196.236': 0.0, '199.7.91.13': 0.0, '192.12.94.30': 0.0, '52.35.35.13': 0.0, '216.239.38.10': 0.0, '199.7.83.42': 0.0, '192.42.93.30': 0.0, '172.217.25.170': 0.0, 'fe80::250:56ff:febe:26db': 0.0, 'ff02::fb': 0.0, '224.0.0.251': 0.0, 'fe80::250:56ff:febe:e9d9': 0.0}
Network Density:	0.03769559032716927
Network Diameter:	2
Network Average Path Length:	1.6

Part 5: Graph Analysis, Network Analysis Metrics

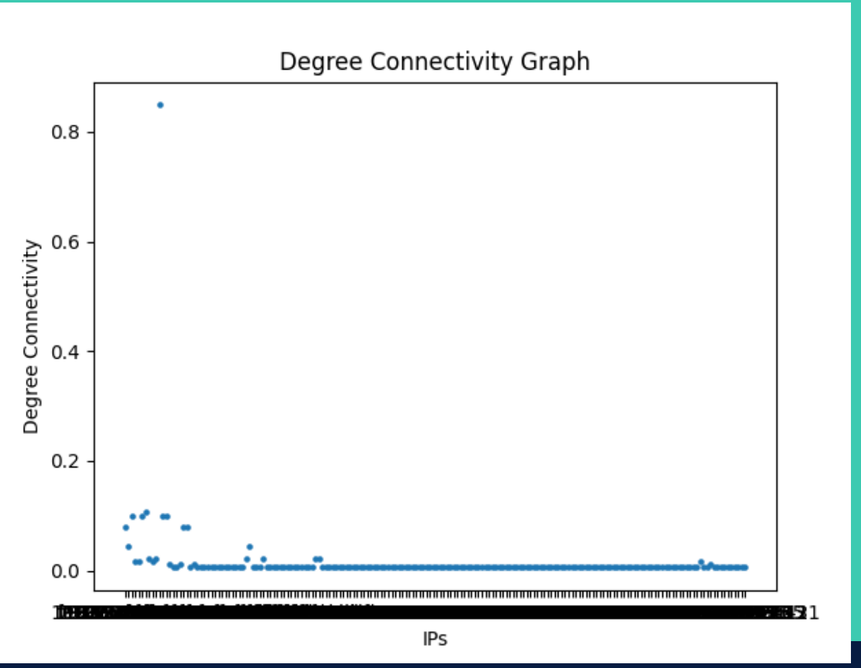
Closeness Centrality



Betweenness Centrality



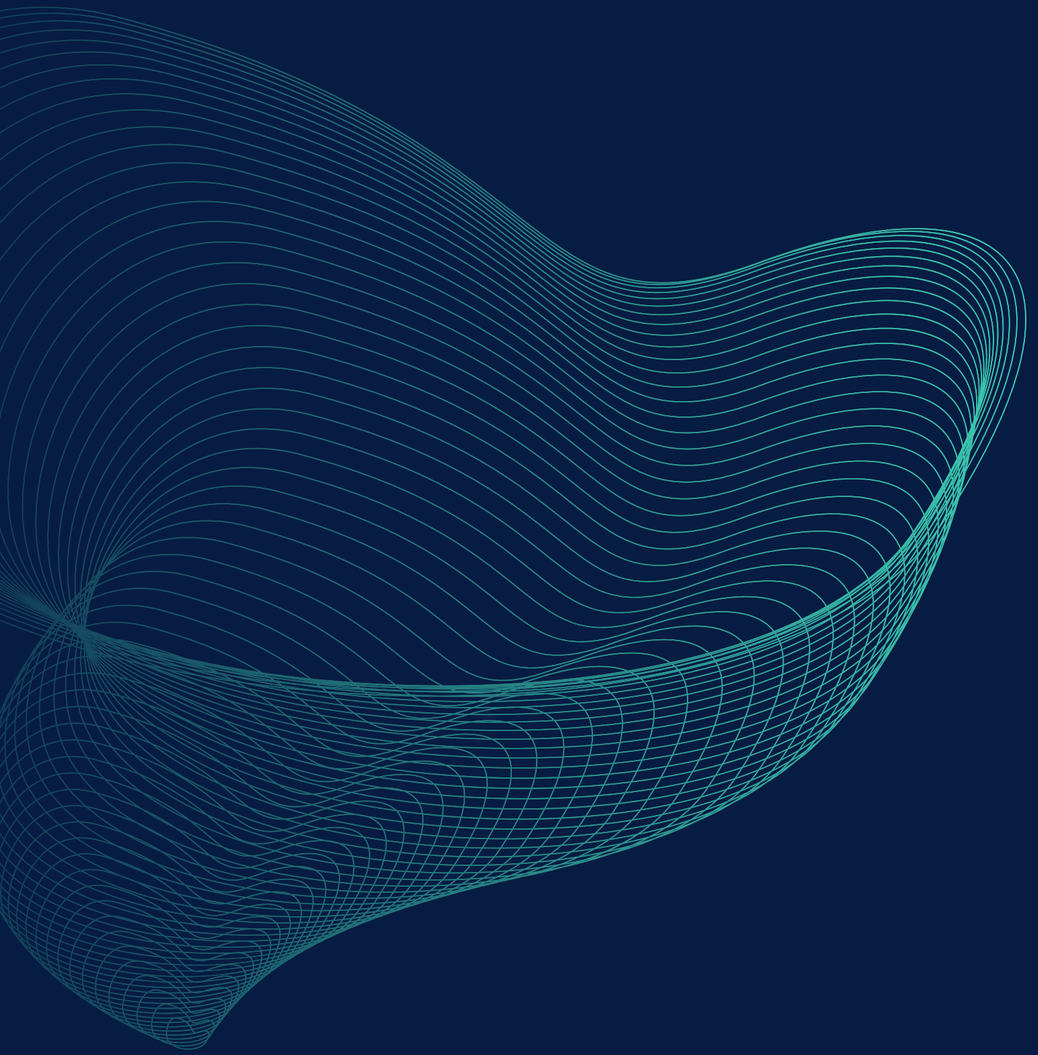
Degree Connectivity



NETWORK DENSITY	**0.0376959032716927**
NETWORK DIAMETER	**2**
NETWORK AVERAGE PATH LENGTH	**1.6**

Part 5: Graph Analysis, Network Analysis Metrics

```python
# ---------- Part 5: Values of parameters of the Graph ----------
def plot_xy_graph(quantity, graph_name, x_label, y_label):
    quantity_nodes = list(quantity.keys())
    quantity_values = list(quantity.values())

    plt.scatter(quantity_nodes, quantity_values, s = 5)
    plt.title(graph_name)
    plt.xlabel(x_label)
    plt.ylabel(y_label)
    plt.show()
# Degree connectivity
degree_connectivity = nx.degree_centrality(Graph)
plot_xy_graph(degree_connectivity, "Degree Connectivity Graph", "IPs", "Degree Connectivity")
# Closeness connectivity
closeness_centrality = nx.closeness_centrality(Graph)
plot_xy_graph(closeness_centrality, "Closeness Centrality Graph", "IPs", "Closeness Centrality")
# Betweenness centrality
betweenness_centrality = nx.betweenness_centrality(Graph)
plot_xy_graph(betweenness_centrality, "Betweenness Centrality Graph", "IPs", "Betweenness Centra
# Network Density
network_density = nx.density(Graph)
print("Network Density:", network_density)
# Network Diameter and Network average Path Length
largest_strongly_connected_component = max(nx.strongly_connected_components(Graph), key=len)
subgraph = Graph.subgraph(largest_strongly_connected_component)
network_diameter = nx.diameter(subgraph)
average_path_length = nx.average_shortest_path_length(subgraph)
print("Network Diameter:", network_diameter)
print("Network Average Path Length:", average_path_length)
```



Thanks for Attention

Students worked on project

Shamkhal Guliyev
Yusif Hajizade

Students ID

22022745
22022735

Email Adress

shamkhal.guliyev@ufaz.az
yusif.hajizade@ufaz.az