Networks Algorithms

Project Work

Graph representation using Python and NetworkX

Shamkhal Guliyev - 22022745

Yusif Hajizade - 22022735

Introduction

In the ever-evolving landscape of network security, the analysis of network traffic data plays a pivotal role in identifying patterns, anomalies, and potential threats. As the volume and complexity of network data continue to grow, there is an increasing need for sophisticated tools and methodologies to extract meaningful insights.

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. By employing graph representation and analysis techniques, we seek to unravel the intricate interplay of entities, protocols, and states that characterize network transactions.

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 approach, we

navigate the dataset, create informative visualizations, and employ network analysis metrics to quantify and interpret the structure of the network.

Objectives

The primary objectives of this project are as follows:

1. **Data Exploration:** Load and preprocess the raw network traffic dataset, providing a comprehensive overview of the attributes and their significance.
2. **Graph Representation:** Utilize NetworkX to represent the network data as graphs, with nodes and edges representing key attributes such as IP addresses, protocols, and attack labels.
3. **Visualization:** Employ Matplotlib and NetworkX to create insightful visualizations that capture the inherent structure and relationships within the network.
4. **Path Finding:** Implement a path-finding algorithm, inspired by Depth-First Search (DFS), to discover routes between specified source and destination IP addresses.
5. **Graph Coloring:** Apply color coding to the graphs based on different attributes, allowing for the visual identification of patterns and distinctions within the network.
6. **Automated Representation:** Develop a user-friendly system to automate the process of graph representation, enabling users to define nodes and edges according to their preferences.
7. **Network Analysis:** Calculate and analyze network metrics, including Degree Connectivity, Closeness Centrality, Betweenness Centrality, Network Density, Network Diameter, and Average Path Length, to gain quantitative insights into the network structure.
8. **Customization:** Allow users to dynamically customize graph visualizations, including attribute-based coloring and flexible representation options.

Part 1: Data Description and Exploration

1. 1 Dataset Loading

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.

```
# 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, nrows = 10000)
  raw_data.columns = attributes
  return raw_data
```

2. 2 Attribute Description

3. The Attribute Description section provides a succinct overview of each dataset attribute, offering clear insights into the nature and significance of the network traffic data for comprehensive understanding and analysis.

3.1. pkSeqID (Row Identifier): Unique identifier for each row in the datas

3.2. Stime (Record start time): The timestamp when the network activity started.
3.3. Flgs (Flow state flags seen in transactions): Flags indicating the state of the network flow (e.g., SYN, ACK).
3.4. Flgs_number (Numerical representation of feature flags): Numeric representation of flow state flags.
3.5. Proto (Textual representation of transaction protocols present in the network flow): The protocol used in the network transaction (e.g., TCP, UDP).
3.6. Proto_number (Numerical representation of feature protocol): Numeric representation of the transaction protocol.
3.7. Saddr (Source IP address): IP address of the source in the network transaction.
3.8. Sport (Source port number): Port number on the source side of the transaction.
3.9. Daddr (Destination IP address): IP address of the destination in the network transaction.
3.10. Dport (Destination port number): Port number on the destination side of the transaction.
3.11. Pkts (Total count of packets in the transaction): The number of packets exchanged in the network transaction.
3.12. Bytes (Total number of bytes in the transaction): The total size of data exchanged in the network transaction.
3.13. State (Transaction state): The state of the network transaction (e.g., Connection Established).
3.14. State_number (Numerical representation of feature state): Numeric representation of the transaction state.
3.15. Ltime (Record last time): The timestamp when the last activity related to the record occurred.
3.16. Seq (Argus sequence number): Sequence number assigned by Argus.
3.17. Dur (Record total duration): The total duration of the network transaction.
3.18. Mean (Average duration of aggregated records): The average duration of all aggregated records.
3.19. Stddev (Standard deviation of aggregated records): The measure of variability among aggregated records' durations.
3.20. Min (Minimum duration of aggregated records): The shortest duration among aggregated records.
3.21. Max (Maximum duration of aggregated records): The longest duration among aggregated records.
3.22. Spkts (Source-to-destination packet count): The count of packets sent from the source to the destination.
3.23. Dpkts (Destination-to-source packet count): The count of packets sent from the destination to the source.
3.24. Sbytes (Source-to-destination byte count): The count of bytes sent from the source to the destination.
3.25. Dbytes (Destination-to-source byte count): The count of bytes sent from the destination to the source.
3.26. Rate (Total packets per second in the transaction): The rate of packets exchanged per second in the network transaction.
3.27. Srate (Source-to-destination packets per second): The rate of packets sent from the source to the destination per second.
3.28. Drate (Destination-to-source packets per second): The rate of packets sent from the destination to the source per second.
3.29. TnBPSrcIP (Total Number of bytes per source IP): The total number of bytes transmitted per source IP.
3.30. TnBPDstIP (Total Number of bytes per Destination IP): The total number of bytes transmitted per destination IP.
3.31. TnP_PSrcIP (Total Number of packets per source IP): The total number of packets transmitted per source IP.
3.32. TnP_PDstIP (Total Number of packets per Destination IP): The total number of packets transmitted per destination IP.
3.33. TnP_PerProto (Total Number of packets per protocol): The total number of packets transmitted per protocol.
3.34. TnP_Per_Dport (Total Number of packets per destination port): The total number of packets transmitted per destination port.
3.35. 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.
3.36. 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.
3.37. N_IN_Conn_P_SrcIP (Number of inbound connections per source IP): The count of inbound connections for a specific source IP.
3.38. N_IN_Conn_P_DstIP (Number of inbound connections per destination IP): The count of inbound connections for a specific destination IP.
3.39. 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.

3.40. 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.
3.41. 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 state of flows and protocols per destination IP.
3.42. 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 flows and protocols per source IP.
3.43. Attack (Class label: 0 for Normal traffic, 1 for Attack Traffic): Indicates whether the network traffic is normal (0) or an attack (1).
3.44. Category (Traffic category): The general category of the network traffic (e.g., Normal, Attack).
3.45. Subcategory (Traffic subcategory): A more specific subcategory of the network traffic.

4. 3 Attribute Relations

Understanding the relationships between attributes is crucial for unraveling the intricacies of network traffic data. In this section, we explore how different attributes interconnect and influence each other within the dataset. The analysis sheds light on the dependencies and correlations that exist, contributing to a holistic comprehension of the underlying network dynamics. By deciphering attribute relations, we gain valuable insights into the patterns and behaviors embedded in the network traffic data, forming a foundation for informed decision-making and enhanced network security.

5.

	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]

Part 2: Graph Representation Using NetworkX

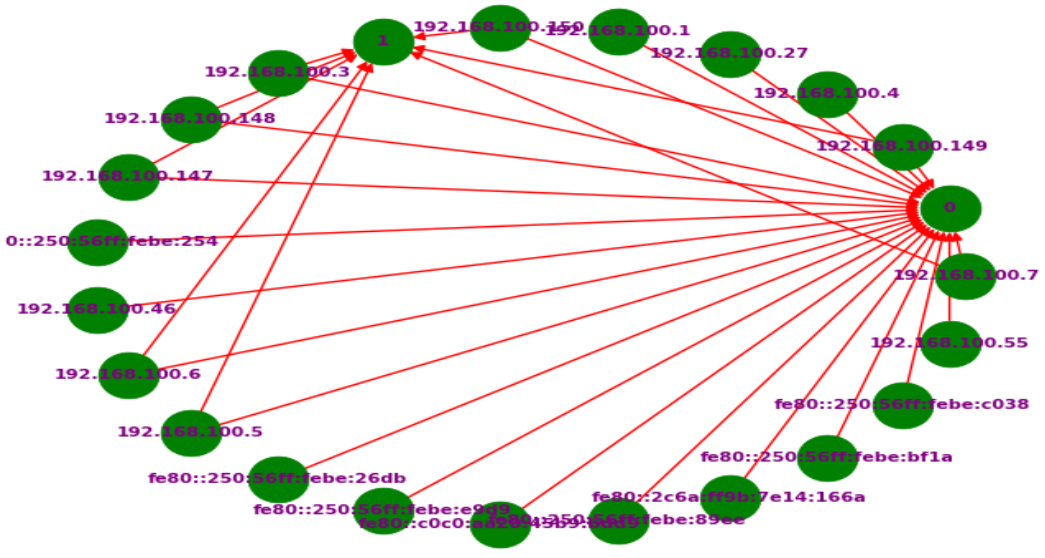
In this section, we employ the NetworkX library to present the network traffic data in graph format. Four distinct representations are created, each offering unique insights into the relationships within the dataset. Graph representations were created with

different attributes as nodes and edges. These representations allow visualizing network traffic patterns based on source IPs, attack labels, protocols, and user-defined attributes.

```
19  # Plots a graph with given data, and node and edge attributes
20  def representation(data, node, edge):
21      Graph = nx.from_pandas_edgelist(data, node, edge, create_using = nx.DiGraph())
22      pos = nx.circular_layout(Graph)
23      nx.draw(Graph, pos, with_labels = True, font_weight = "bold", node_size = 700, node
24      plt.show()
25
26  formatted_data = assign_attributes("UNSW_2018_IoT_Botnet_Dataset_1.csv")
27  representation(formatted_data, "SourceIP", "Attack") # First representation
28  representation(formatted_data, "Attack", "SourceIP") # Second representation
29  representation(formatted_data, "SourceIP", "Protocol") # Third representation
30
31  # # Fourth representation
32  user_node = input("Enter the attribute that will be used as a node: ")
33  user_edge = input("Enter the attribute that will be used as an edge: ")
34  representation(formatted_data, user_node, user_edge)
```

First representation: "IP" as Nodes and "Attack" as Edges

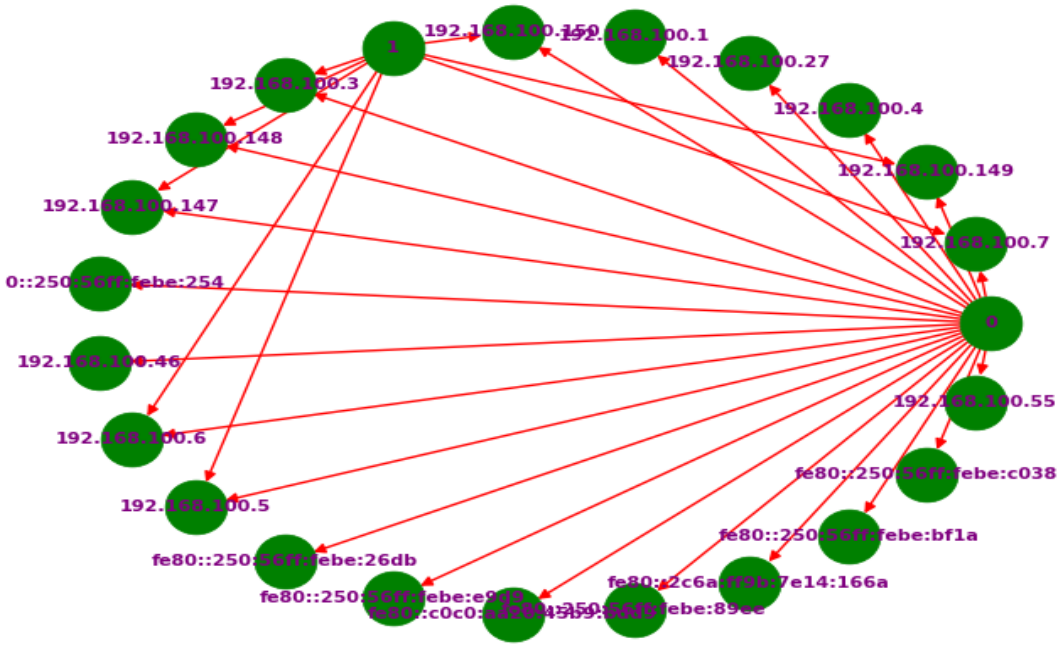
In the first representation, nodes correspond to unique "SourceIP" addresses, and directed edges signify the occurrence of "Attack" events. This graph sheds light on the connections between different source IPs and the presence of attacks within the network, providing a focused view of potentially malicious activities.



Second Representation: Event (Row) as Nodes

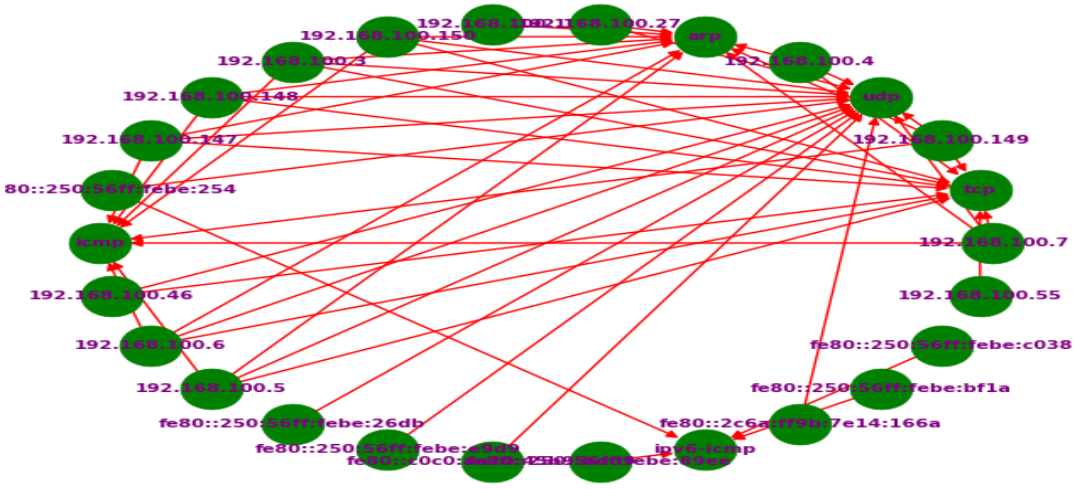
For the second representation, each network transaction event (row) is represented as a distinct node in the graph. This visualization emphasizes the individual transactions,

allowing for an overview of the entire dataset's activity pattern and aiding in the identification of specific events that might stand out.



Third Representation: User-Defined Attribute as Nodes

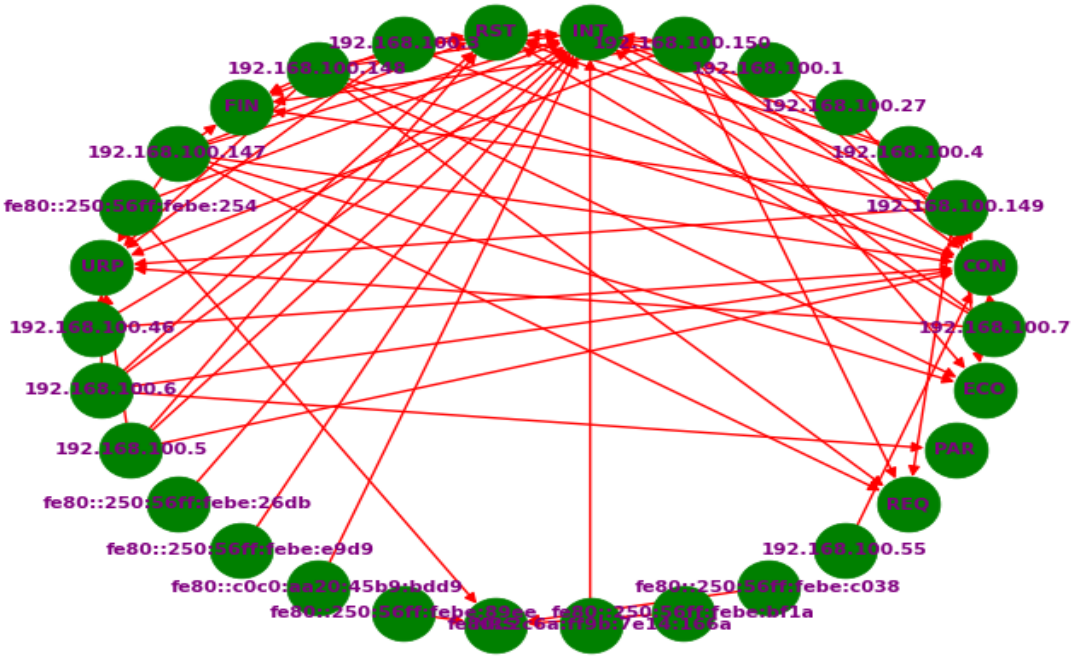
In the third representation, a user-selected attribute serves as the nodes, and "Protocol" is used as the edges. This flexible approach enables the exploration of network structures based on attributes chosen dynamically, facilitating a targeted analysis tailored to specific attributes of interest



Fourth Representation: User-Defined Node and Edge Attributes

```
Enter the attribute that will be used as a node: SourceIP
Enter the attribute that will be used as an edge: State
```

The fourth representation introduces interactivity, allowing users to input attributes for both nodes and edges. This dynamic approach empowers users to customize the graph, making it a versatile tool for exploring different facets of the network traffic data.



Part 3: Visualization and Path Finding

Matplotlib and NetworkX were used to successfully plot the graph, providing a visual representation of the network connections.A Depth-First Search (DFS) algorithm was implemented to find a path between specified source and destination IPs.

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

Function: `find_path(Graph, start, end)` - This function uses the NetworkX library's `shortest_path` method to determine the shortest path between two given IP addresses (`start` and `end`) within the directed graph (`Graph`) representing the network traffic.

Function: `calculate_packets(Graph, path)` -This function calculates the total number of packets sent along the identified path. It iterates through each pair of consecutive nodes in the path, checking for a direct edge between them in the graph. If an edge exists, the corresponding packet count is added to the cumulative total.

Path Finding and Packet Calculation: - The directed graph (`Graph`) is constructed from the dataset using the "SourceIP" and "DestinationIP" attributes as nodes and including the "Attack"

and "Packets" attributes as edge attributes. The DFS algorithm is applied to find a path between a specified origin ("192.168.100.150") and destination ("192.168.217.2"). If a path is found, it is printed, and the total size of sent packets is calculated and displayed. If no path exists, a relevant message is printed.

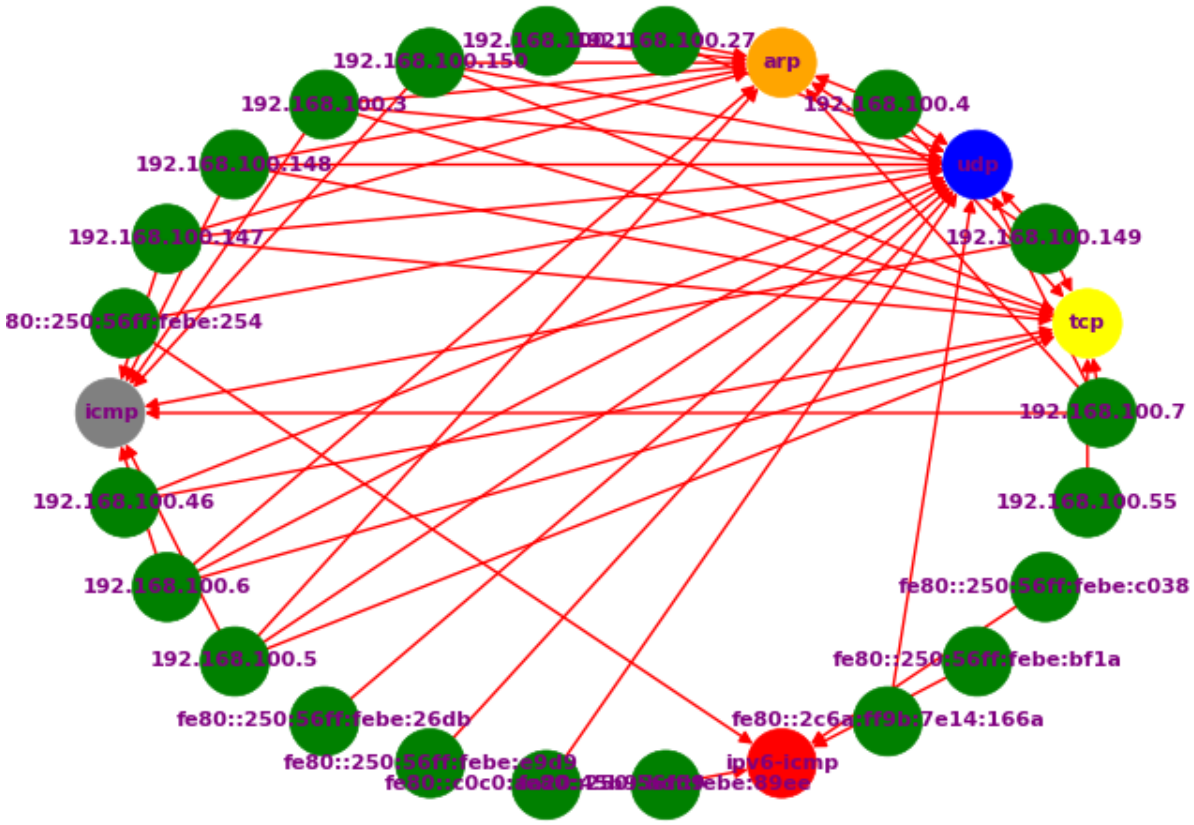
```
# ---------- Part 3: DFS and calculating packets ----------
def find_path(Graph, start, end):
    path = nx.shortest_path(Graph, source = start, target = end)
    return path

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", |
graph_dict = dict(Graph.adjacency())

origin = "192.168.100.150"
destination = "192.168.217.2"

path = find_path(Graph, origin, destination)

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 size of sent packets: {packets} bytes")
else:
    print(f"No path found from {origin} to {destination}")
```

Part 4: Graph Coloring

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.

```
# ---------- Part 4: Differentiating protocols ----------
def representation_with_colors(data, node, edge, colors):
    Graph = nx.from_pandas_edgelist(data, node, edge, create_using = nx.DiGraph())

    source_ip_colors = {ip: color for ip, color in zip(data["Protocol"].unique(), colors)}
    node_colors = [source_ip_colors.get(ip, "green") for ip in Graph.nodes()]

    pos = nx.circular_layout(Graph)
    nx.draw(Graph, pos, with_labels = True, font_weight = "bold", node_size = 700, node_color = node_colors,
            edge_color = "red", font_size = 8, linewidths = 0.5, font_color = "purple")

    plt.show()

colors = ["", "", "", "", ""] # For example: yellow, blue, orange, gray, red
protocols = ["arp", "tcp", "udp", "icmp", "ipv6-icmp"]

for index in range(5):
    color = input(f"Enter the color for the protocol {protocols[index]}: ")
    colors[index] = color

representation_with_colors(formatted_data, "SourceIP", "Protocol", colors)
```

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



Part 5: Graph Analysis, Network Analysis Metrics

Network analysis metrics, including Degree Connectivity, Closeness Centrality, Betweenness Centrality, Network Density, Network Diameter, and Average Path Length, were calculated and analyzed. These metrics provide insights into the network structure and characteristics.

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': 0.007507507507507074, '192.168.100.150': 0.015765765765765764, '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

Conclusion:

In conclusion, this project successfully navigated the complexities of network traffic analysis. The synergy of statistical tools and graphical representations provided deep insights. Automated features and user interactivity enhance adaptability. This project serves as a foundation for cybersecurity insights, showcasing the potency of Python, Pandas, and NetworkX in decoding network intricacies.