1_2_data_preparation

December 20, 2024

[1]:
```
# Network Flow Analysis and Anomaly Detection Script
# This script processes PCAP files to extract, analyze and detect network flow anomalies
# It implements feature engineering for network security analysis
```

[2]:
```
import sys
import os  # For interacting with the file system
import pandas as pd  # For handling dataframes and CSVs
import numpy as np
from datetime import datetime
from tqdm import tqdm  # Import tqdm for progress tracking
from scapy import all
from nfstream import NFStreamer  # For working with PCAP files and flow analysis
import csv
import pandas as pd
import numpy as np
from scipy import stats
from sklearn.preprocessing import StandardScaler
```

[3]:
```
print("Starting network flow analysis and anomaly detection process...")

# Specify the directory containing the .pcap files
pcap_directory = 'pcap_files'
print(f"Looking for PCAP files in directory: {pcap_directory}")

# Initialize an empty list to store all flows data
all_flows_data = []

# Iterate through each .pcap file in the directory
for file in os.listdir(pcap_directory):
    # Check if the file is a .pcap file
    if file.endswith('.pcap'):
        full_path = os.path.join(pcap_directory, file)  # Get the full path of the file
        print(f"Processing file: {full_path}")

        # Create an NFStreamer instance with statistical analysis enabled
```

```
        # Set timeouts to handle both short and long-lived flows
        my_streamer = NFStreamer(
            source=full_path,
            statistical_analysis=True,
            idle_timeout=60,  # 60 seconds idle timeout
            active_timeout=120  # 120 seconds active timeout
        )

        # List to store extracted flow data for this file
        file_flows_data = []
        total_flows = len(list(my_streamer))  # Count total flows for tqdm
        my_streamer = NFStreamer(source=full_path, statistical_analysis=True,
                                 idle_timeout=60, active_timeout=120)  #␣
↪Re-create the streamer

        print(f"Found {total_flows} flows in {file}")

        # Using tqdm to track progress for flow extraction
        with tqdm(total=total_flows, desc=f"Extracting flows from {file}",␣
↪unit="flows") as pbar:
            for flow in my_streamer:
                # Extract comprehensive flow metrics including statistical␣
↪features
                flow_data = {
                    'src_ip': flow.src_ip,
                    'dst_ip': flow.dst_ip,
                    'src_port': flow.src_port,
                    'dst_port': flow.dst_port,
                    'protocol': flow.protocol,
                    'application_name': flow.application_name,
                    'bidirectional_packets': flow.bidirectional_packets,
                    'bidirectional_bytes': flow.bidirectional_bytes,
                    'bidirectional_first_seen_ms': flow.
↪bidirectional_first_seen_ms,
                    'bidirectional_last_seen_ms': flow.
↪bidirectional_last_seen_ms,

                    # Statistical features for anomaly detection
                    'bidirectional_mean_ps': flow.bidirectional_mean_ps,  #␣
↪Packet size statistics
                    'bidirectional_stddev_ps': flow.bidirectional_stddev_ps,
                    'src2dst_mean_ps': flow.src2dst_mean_ps,
                    'src2dst_stddev_ps': flow.src2dst_stddev_ps,
                    'dst2src_mean_ps': flow.dst2src_mean_ps,
                    'dst2src_stddev_ps': flow.dst2src_stddev_ps,
```

```
                    # Packet Inter-Arrival Time (PIAT) statistics
                    'bidirectional_mean_piat_ms': flow.
 ↪bidirectional_mean_piat_ms,
                    'bidirectional_stddev_piat_ms': flow.
 ↪bidirectional_stddev_piat_ms,
                    'src2dst_mean_piat_ms': flow.src2dst_mean_piat_ms,
                    'src2dst_stddev_piat_ms': flow.src2dst_stddev_piat_ms,
                    'dst2src_mean_piat_ms': flow.dst2src_mean_piat_ms,
                    'dst2src_stddev_piat_ms': flow.dst2src_stddev_piat_ms
                }
                file_flows_data.append(flow_data)
                pbar.update(1)  # Update progress bar

        # Append the current file's data to the all_flows_data list
        all_flows_data.extend(file_flows_data)

print("Flow extraction completed. Creating DataFrame...")
```

```
Starting network flow analysis and anomaly detection process…
Looking for PCAP files in directory: pcap_files
Processing file: pcap_files\trace_a_1.pcap
Found 23019 flows in trace_a_1.pcap

Extracting flows from trace_a_1.pcap: 100%|      | 23019/23019 [00:09<00:00,
2531.41flows/s]

Processing file: pcap_files\trace_a_10.pcap
Found 21963 flows in trace_a_10.pcap

Extracting flows from trace_a_10.pcap: 100%|      | 21963/21963
[00:09<00:00, 2370.76flows/s]

Processing file: pcap_files\trace_a_11.pcap
Found 18030 flows in trace_a_11.pcap

Extracting flows from trace_a_11.pcap: 100%|      | 18030/18030
[00:08<00:00, 2066.67flows/s]

Processing file: pcap_files\trace_a_12.pcap
Found 19826 flows in trace_a_12.pcap

Extracting flows from trace_a_12.pcap: 100%|      | 19826/19826
[00:08<00:00, 2233.05flows/s]

Processing file: pcap_files\trace_a_13.pcap
Found 17582 flows in trace_a_13.pcap

Extracting flows from trace_a_13.pcap: 100%|      | 17582/17582
[00:08<00:00, 2155.04flows/s]

Processing file: pcap_files\trace_a_14.pcap
Found 18516 flows in trace_a_14.pcap
```

```
Extracting flows from trace_a_14.pcap: 100%|     | 18516/18516
[00:08<00:00, 2226.86flows/s]

Processing file: pcap_files\trace_a_15.pcap
Found 21081 flows in trace_a_15.pcap

Extracting flows from trace_a_15.pcap: 100%|     | 21081/21081
[00:08<00:00, 2453.51flows/s]

Processing file: pcap_files\trace_a_16.pcap
Found 20535 flows in trace_a_16.pcap

Extracting flows from trace_a_16.pcap: 100%|     | 20535/20535
[00:09<00:00, 2229.78flows/s]

Processing file: pcap_files\trace_a_17.pcap
Found 20082 flows in trace_a_17.pcap

Extracting flows from trace_a_17.pcap: 100%|     | 20082/20082
[00:08<00:00, 2327.55flows/s]

Processing file: pcap_files\trace_a_18.pcap
Found 19273 flows in trace_a_18.pcap

Extracting flows from trace_a_18.pcap: 100%|     | 19273/19273
[00:08<00:00, 2369.48flows/s]

Processing file: pcap_files\trace_a_19.pcap
Found 17677 flows in trace_a_19.pcap

Extracting flows from trace_a_19.pcap: 100%|     | 17677/17677
[00:08<00:00, 2057.37flows/s]

Processing file: pcap_files\trace_a_2.pcap
Found 16798 flows in trace_a_2.pcap

Extracting flows from trace_a_2.pcap: 100%|     | 16798/16798 [00:07<00:00,
2131.71flows/s]

Processing file: pcap_files\trace_a_20.pcap
Found 20510 flows in trace_a_20.pcap

Extracting flows from trace_a_20.pcap: 100%|     | 20510/20510
[00:08<00:00, 2351.34flows/s]

Processing file: pcap_files\trace_a_21.pcap
Found 17665 flows in trace_a_21.pcap

Extracting flows from trace_a_21.pcap: 100%|     | 17665/17665
[00:08<00:00, 2062.72flows/s]

Processing file: pcap_files\trace_a_22.pcap
Found 18072 flows in trace_a_22.pcap

Extracting flows from trace_a_22.pcap: 100%|     | 18072/18072
[00:07<00:00, 2298.56flows/s]
```

```
Processing file: pcap_files\trace_a_23.pcap
Found 18576 flows in trace_a_23.pcap
Extracting flows from trace_a_23.pcap: 100%|     | 18576/18576
[00:08<00:00, 2090.48flows/s]
Processing file: pcap_files\trace_a_24.pcap
Found 20074 flows in trace_a_24.pcap
Extracting flows from trace_a_24.pcap: 100%|     | 20074/20074
[00:08<00:00, 2331.91flows/s]
Processing file: pcap_files\trace_a_25.pcap
Found 20802 flows in trace_a_25.pcap
Extracting flows from trace_a_25.pcap: 100%|     | 20802/20802
[00:08<00:00, 2431.02flows/s]
Processing file: pcap_files\trace_a_26.pcap
Found 20633 flows in trace_a_26.pcap
Extracting flows from trace_a_26.pcap: 100%|     | 20633/20633
[00:08<00:00, 2329.22flows/s]
Processing file: pcap_files\trace_a_27.pcap
Found 17501 flows in trace_a_27.pcap
Extracting flows from trace_a_27.pcap: 100%|     | 17501/17501
[00:08<00:00, 2085.60flows/s]
Processing file: pcap_files\trace_a_28.pcap
Found 18294 flows in trace_a_28.pcap
Extracting flows from trace_a_28.pcap: 100%|     | 18294/18294
[00:08<00:00, 2176.39flows/s]
Processing file: pcap_files\trace_a_29.pcap
Found 18727 flows in trace_a_29.pcap
Extracting flows from trace_a_29.pcap: 100%|     | 18727/18727
[00:08<00:00, 2205.66flows/s]
Processing file: pcap_files\trace_a_3.pcap
Found 18430 flows in trace_a_3.pcap
Extracting flows from trace_a_3.pcap: 100%|     | 18430/18430 [00:08<00:00,
2275.39flows/s]
Processing file: pcap_files\trace_a_30.pcap
Found 19023 flows in trace_a_30.pcap
Extracting flows from trace_a_30.pcap: 100%|     | 19023/19023
[00:08<00:00, 2134.05flows/s]
Processing file: pcap_files\trace_a_31.pcap
Found 24345 flows in trace_a_31.pcap
```

```
Extracting flows from trace_a_31.pcap: 100%|     | 24345/24345
[00:08<00:00, 2850.56flows/s]

Processing file: pcap_files\trace_a_32.pcap
Found 17665 flows in trace_a_32.pcap

Extracting flows from trace_a_32.pcap: 100%|     | 17665/17665
[00:08<00:00, 2107.87flows/s]

Processing file: pcap_files\trace_a_33.pcap
Found 17960 flows in trace_a_33.pcap

Extracting flows from trace_a_33.pcap: 100%|     | 17960/17960
[00:08<00:00, 2119.10flows/s]

Processing file: pcap_files\trace_a_34.pcap
Found 18563 flows in trace_a_34.pcap

Extracting flows from trace_a_34.pcap: 100%|     | 18563/18563
[00:08<00:00, 2246.73flows/s]

Processing file: pcap_files\trace_a_35.pcap
Found 16779 flows in trace_a_35.pcap

Extracting flows from trace_a_35.pcap: 100%|     | 16779/16779
[00:08<00:00, 2009.32flows/s]

Processing file: pcap_files\trace_a_36.pcap
Found 22591 flows in trace_a_36.pcap

Extracting flows from trace_a_36.pcap: 100%|     | 22591/22591
[00:08<00:00, 2647.54flows/s]

Processing file: pcap_files\trace_a_37.pcap
Found 21634 flows in trace_a_37.pcap

Extracting flows from trace_a_37.pcap: 100%|     | 21634/21634
[00:08<00:00, 2683.79flows/s]

Processing file: pcap_files\trace_a_38.pcap
Found 20679 flows in trace_a_38.pcap

Extracting flows from trace_a_38.pcap: 100%|     | 20679/20679
[00:08<00:00, 2483.85flows/s]

Processing file: pcap_files\trace_a_39.pcap
Found 19001 flows in trace_a_39.pcap

Extracting flows from trace_a_39.pcap: 100%|     | 19001/19001
[00:07<00:00, 2395.98flows/s]

Processing file: pcap_files\trace_a_4.pcap
Found 18902 flows in trace_a_4.pcap

Extracting flows from trace_a_4.pcap: 100%|     | 18902/18902 [00:07<00:00,
2509.09flows/s]
```

```
Processing file: pcap_files\trace_a_40.pcap
Found 20941 flows in trace_a_40.pcap
Extracting flows from trace_a_40.pcap: 100%|     | 20941/20941
[00:08<00:00, 2565.50flows/s]
Processing file: pcap_files\trace_a_41.pcap
Found 27382 flows in trace_a_41.pcap
Extracting flows from trace_a_41.pcap: 100%|     | 27382/27382
[00:09<00:00, 3020.53flows/s]
Processing file: pcap_files\trace_a_42.pcap
Found 18396 flows in trace_a_42.pcap
Extracting flows from trace_a_42.pcap: 100%|     | 18396/18396
[00:07<00:00, 2305.97flows/s]
Processing file: pcap_files\trace_a_43.pcap
Found 16281 flows in trace_a_43.pcap
Extracting flows from trace_a_43.pcap: 100%|     | 16281/16281
[00:07<00:00, 2136.51flows/s]
Processing file: pcap_files\trace_a_44.pcap
Found 17909 flows in trace_a_44.pcap
Extracting flows from trace_a_44.pcap: 100%|     | 17909/17909
[00:07<00:00, 2241.55flows/s]
Processing file: pcap_files\trace_a_45.pcap
Found 19257 flows in trace_a_45.pcap
Extracting flows from trace_a_45.pcap: 100%|     | 19257/19257
[00:08<00:00, 2279.10flows/s]
Processing file: pcap_files\trace_a_46.pcap
Found 18156 flows in trace_a_46.pcap
Extracting flows from trace_a_46.pcap: 100%|     | 18156/18156
[00:08<00:00, 2267.40flows/s]
Processing file: pcap_files\trace_a_47.pcap
Found 19268 flows in trace_a_47.pcap
Extracting flows from trace_a_47.pcap: 100%|     | 19268/19268
[00:08<00:00, 2259.33flows/s]
Processing file: pcap_files\trace_a_48.pcap
Found 21783 flows in trace_a_48.pcap
Extracting flows from trace_a_48.pcap: 100%|     | 21783/21783
[00:08<00:00, 2682.75flows/s]
Processing file: pcap_files\trace_a_49.pcap
Found 21207 flows in trace_a_49.pcap
```

```
Extracting flows from trace_a_49.pcap: 100%|     | 21207/21207
[00:08<00:00, 2632.49flows/s]

Processing file: pcap_files\trace_a_5.pcap
Found 20015 flows in trace_a_5.pcap

Extracting flows from trace_a_5.pcap: 100%|     | 20015/20015 [00:07<00:00,
2521.97flows/s]

Processing file: pcap_files\trace_a_50.pcap
Found 21986 flows in trace_a_50.pcap

Extracting flows from trace_a_50.pcap: 100%|     | 21986/21986
[00:08<00:00, 2523.98flows/s]

Processing file: pcap_files\trace_a_51.pcap
Found 18619 flows in trace_a_51.pcap

Extracting flows from trace_a_51.pcap: 100%|     | 18619/18619
[00:08<00:00, 2205.42flows/s]

Processing file: pcap_files\trace_a_52.pcap
Found 17665 flows in trace_a_52.pcap

Extracting flows from trace_a_52.pcap: 100%|     | 17665/17665
[00:08<00:00, 2033.66flows/s]

Processing file: pcap_files\trace_a_53.pcap
Found 13030 flows in trace_a_53.pcap

Extracting flows from trace_a_53.pcap: 100%|     | 13030/13030
[00:05<00:00, 2204.85flows/s]

Processing file: pcap_files\trace_a_6.pcap
Found 18724 flows in trace_a_6.pcap

Extracting flows from trace_a_6.pcap: 100%|     | 18724/18724 [00:07<00:00,
2369.70flows/s]

Processing file: pcap_files\trace_a_7.pcap
Found 21135 flows in trace_a_7.pcap

Extracting flows from trace_a_7.pcap: 100%|     | 21135/21135 [00:08<00:00,
2430.04flows/s]

Processing file: pcap_files\trace_a_8.pcap
Found 18742 flows in trace_a_8.pcap

Extracting flows from trace_a_8.pcap: 100%|     | 18742/18742 [00:08<00:00,
2251.27flows/s]

Processing file: pcap_files\trace_a_9.pcap
Found 18143 flows in trace_a_9.pcap

Extracting flows from trace_a_9.pcap: 100%|     | 18143/18143 [00:08<00:00,
2218.07flows/s]
```

```
Flow extraction completed. Creating DataFrame…
```

[4]:
```python
# Convert the list of flow data into a pandas DataFrame
flows_df = pd.DataFrame(all_flows_data)

# Display the DataFrame's summary
print("DataFrame created with the following structure:")
print(flows_df.info())
```

```
DataFrame created with the following structure:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1029447 entries, 0 to 1029446
Data columns (total 22 columns):
 #   Column                        Non-Null Count    Dtype
---  ------                        --------------    -----
 0   src_ip                        1029447 non-null  object
 1   dst_ip                        1029447 non-null  object
 2   src_port                      1029447 non-null  int64
 3   dst_port                      1029447 non-null  int64
 4   protocol                      1029447 non-null  int64
 5   application_name              1029447 non-null  object
 6   bidirectional_packets         1029447 non-null  int64
 7   bidirectional_bytes           1029447 non-null  int64
 8   bidirectional_first_seen_ms   1029447 non-null  int64
 9   bidirectional_last_seen_ms    1029447 non-null  int64
 10  bidirectional_mean_ps         1029447 non-null  float64
 11  bidirectional_stddev_ps       1029447 non-null  float64
 12  src2dst_mean_ps               1029447 non-null  float64
 13  src2dst_stddev_ps             1029447 non-null  float64
 14  dst2src_mean_ps               1029447 non-null  float64
 15  dst2src_stddev_ps             1029447 non-null  float64
 16  bidirectional_mean_piat_ms    1029447 non-null  float64
 17  bidirectional_stddev_piat_ms  1029447 non-null  float64
 18  src2dst_mean_piat_ms          1029447 non-null  float64
 19  src2dst_stddev_piat_ms        1029447 non-null  float64
 20  dst2src_mean_piat_ms          1029447 non-null  float64
 21  dst2src_stddev_piat_ms        1029447 non-null  float64
dtypes: float64(12), int64(7), object(3)
memory usage: 172.8+ MB
None
```

[5]:
```python
# Save raw flows to CSV
output_path = "csv_files/raw_flows.csv"
flows_df.to_csv(output_path, index=False)
print(f"Raw flows DataFrame saved to {output_path}")
```

```
Raw flows DataFrame saved to csv_files/raw_flows.csv
```

[6]:
```
# Define the time window T in milliseconds (used for fan-in/fan-out calculation)
T = 10  # 10 seconds window
print(f"\nCalculating network features using {T} second time window...")

def calculate_features(df, T):
    """
    Calculate fan-in and fan-out metrics for each flow within a sliding time␣
 ↪window

    Parameters:
    df (DataFrame): Input flows DataFrame
    T (int): Time window in seconds

    Returns:
    DataFrame: Enriched DataFrame with fan-in/fan-out metrics
    """
    # Convert period T to milliseconds
    T_ms = T * 1000

    # Initialize new columns for fan metrics
    df['fan_out_src'] = 0  # Number of unique destinations from source
    df['fan_in_dst'] = 0   # Number of unique sources to destination
    df['fan_in_src'] = 0   # Number of unique sources to source
    df['fan_out_dst'] = 0  # Number of unique destinations from destination

    for i, row in tqdm(df.iterrows(), total=df.shape[0], desc="Calculating fan␣
 ↪metrics", unit="flows"):
        # Define sliding time window
        mid_timestamp = row['bidirectional_first_seen_ms']
        time_window_start = mid_timestamp - (T_ms // 2)
        time_window_end = mid_timestamp + (T_ms // 2)

        # Filter flows within time window
        window_df = df[(df['bidirectional_first_seen_ms'] >= time_window_start)␣
 ↪&
                       (df['bidirectional_first_seen_ms'] <= time_window_end)].
 ↪copy()

        # Calculate fan metrics
        df.at[i, 'fan_in_src'] = window_df[window_df['dst_ip'] ==␣
 ↪row['src_ip']]['src_ip'].nunique()
        df.at[i, 'fan_out_src'] = window_df[window_df['src_ip'] ==␣
 ↪row['src_ip']]['dst_ip'].nunique()
        df.at[i, 'fan_in_dst'] = window_df[window_df['dst_ip'] ==␣
 ↪row['dst_ip']]['src_ip'].nunique()
```

```
        df.at[i, 'fan_out_dst'] = window_df[window_df['src_ip'] ==␣
↪row['dst_ip']]['dst_ip'].nunique()

    return df

# Apply the feature calculation function
print("Calculating fan-in/fan-out metrics...")
df = calculate_features(flows_df, T)
df.head()
```

```
Calculating network features using 10 second time window…
Calculating fan-in/fan-out metrics…

Calculating fan metrics: 100%|      | 1029447/1029447 [54:58<00:00,
312.13flows/s]
```

[6]:

	src_ip	dst_ip	src_port	dst_port	protocol	\
0	59.166.0.5	149.171.126.5	3593	53	17	
1	59.166.0.0	149.171.126.9	33661	1024	17	
2	175.45.176.0	149.171.126.16	13284	80	6	
3	59.166.0.6	149.171.126.7	1464	53	17	
4	59.166.0.0	149.171.126.9	32119	111	17	

	application_name	bidirectional_packets	bidirectional_bytes	\
0	DNS	4	360	
1	NFS	8	960	
2	HTTP	20	1950	
3	DNS	4	388	
4	NFS	8	1008	

	bidirectional_first_seen_ms	bidirectional_last_seen_ms	…	\
0	1421927414035	1421927414036	…	
1	1421927414236	1421927414272	…	
2	1421927413887	1421927416277	…	
3	1421927414121	1421927414122	…	
4	1421927414221	1421927414299	…	

	bidirectional_mean_piat_ms	bidirectional_stddev_piat_ms	\
0	0.333333	0.577350	
1	5.142857	6.890297	
2	125.789474	487.795105	
3	0.333333	0.577350	
4	11.142857	21.388693	

	src2dst_mean_piat_ms	src2dst_stddev_piat_ms	dst2src_mean_piat_ms	\
0	0.000000	0.000000	0.000000	
1	7.000000	12.124356	7.333333	

```
2            183.538462              587.939001            474.200000
3              0.000000                0.000000              0.000000
4             21.000000               36.373067             24.333333

   dst2src_stddev_piat_ms  fan_out_src  fan_in_dst  fan_in_src  fan_out_dst
0                0.000000            6           7           0            0
1               12.701706            7           8           0            0
2             1053.091259            2           2           0            0
3                0.000000            7           3           0            0
4               42.146570            7           8           0            0

[5 rows x 26 columns]
```

[7]:
```
# Save the enriched dataframe to a CSV file
csv_filename = 'csv_files/enriched_flows.csv'
df.to_csv(csv_filename, index=False)
print(f"Enriched flows saved to {csv_filename}")
```

```
Enriched flows saved to csv_files/enriched_flows.csv
```

[8]:
```
print("\nLoading ground truth data for flow labeling...")
# Load ground truth file for labeling
gt_file = 'pcap_files/flows/TRAIN.gt'
train_gt_df = pd.read_csv(gt_file)
```

```
Loading ground truth data for flow labeling…
```

[9]:
```
# Convert necessary columns to compatible types
df['src_port'] = df['src_port'].astype(int)
df['dst_port'] = df['dst_port'].astype(int)
```

[11]:
```
def match_label(row):
    """
    Match flow with ground truth data to determine if it's anomalous
    Returns 1 for anomalous flows, 0 for normal flows
    """
    # Match based on ports, protocol, and timestamp overlap
    matched = train_gt_df[
        (train_gt_df['src_port'] == row['src_port']) &
        (train_gt_df['dst_port'] == row['dst_port']) &
        (train_gt_df['protocol'] == row['protocol']) &
        (train_gt_df['first_timestamp_ms'] <=␣
↪row['bidirectional_last_seen_ms']) &
        (train_gt_df['last_timestamp_ms'] >= row['bidirectional_first_seen_ms'])
    ]
```

```
    return 1 if not matched.empty else 0

print("Labeling flows based on ground truth data...")

# Use tqdm to apply the function with a progress bar
tqdm.pandas(desc="Labeling rows")
df['label'] = df.progress_apply(match_label, axis=1)

# Display the first few rows of the labeled DataFrame
df.head()
```

```
Labeling flows based on ground truth data…
Labeling rows: 100%|      | 1029447/1029447 [07:37<00:00, 2252.26it/s]
```

[11]:

	src_ip	dst_ip	src_port	dst_port	protocol	\
0	59.166.0.5	149.171.126.5	3593	53	17	
1	59.166.0.0	149.171.126.9	33661	1024	17	
2	175.45.176.0	149.171.126.16	13284	80	6	
3	59.166.0.6	149.171.126.7	1464	53	17	
4	59.166.0.0	149.171.126.9	32119	111	17	

	application_name	bidirectional_packets	bidirectional_bytes	\
0	DNS	4	360	
1	NFS	8	960	
2	HTTP	20	1950	
3	DNS	4	388	
4	NFS	8	1008	

	bidirectional_first_seen_ms	bidirectional_last_seen_ms	…	\
0	1421927414035	1421927414036	…	
1	1421927414236	1421927414272	…	
2	1421927413887	1421927416277	…	
3	1421927414121	1421927414122	…	
4	1421927414221	1421927414299	…	

	bidirectional_stddev_piat_ms	src2dst_mean_piat_ms	src2dst_stddev_piat_ms	\
0	0.577350	0.000000	0.000000	
1	6.890297	7.000000	12.124356	
2	487.795105	183.538462	587.939001	
3	0.577350	0.000000	0.000000	
4	21.388693	21.000000	36.373067	

	dst2src_mean_piat_ms	dst2src_stddev_piat_ms	fan_out_src	fan_in_dst	\
0	0.000000	0.000000	6	7	
1	7.333333	12.701706	7	8	
2	474.200000	1053.091259	2	2	
3	0.000000	0.000000	7	3	

```
4             24.333333              42.146570            7           8

   fan_in_src  fan_out_dst  label
0           0            0      0
1           0            0      0
2           0            0      1
3           0            0      0
4           0            0      0

[5 rows x 27 columns]
```

[12]:
```python
# Save labeled DataFrame
df.to_csv('csv_files/labeled_flows.csv', index=False)
print(f"Labeled flows saved to {output_path}")
```

```
Labeled flows saved to csv_files/raw_flows.csv
```

[13]:
```python
def get_first_octet(ip):
    """Extract first octet from IP address"""
    return int(ip.split('.')[0])

def get_ip_class(ip):
    """
    Determine IP address class based on first octet
    Returns: Class A, B, C, D (multicast), or E (reserved)
    """
    first_octet = get_first_octet(ip)

    if 0 <= first_octet <= 127:  # Class A
        return 'Class A'
    elif 128 <= first_octet <= 191:  # Class B
        return 'Class B'
    elif 192 <= first_octet <= 223:  # Class C
        return 'Class C'
    elif 224 <= first_octet <= 239:  # Class D (multicast)
        return 'Class D (multicast)'
    elif 240 <= first_octet <= 255:  # Class E (reserved)
        return 'Class E (reserved)'
    else:
        return 'Unknown'

print("\nEnriching data with IP classification...")
# Apply IP classification
df['src_ip_class'] = df['src_ip'].apply(get_ip_class)
df['dst_ip_class'] = df['dst_ip'].apply(get_ip_class)
df = pd.get_dummies(df, columns=['src_ip_class', 'dst_ip_class'])
```

Enriching data with IP classification…

[14]:
```
# Calculate flow duration
df['bidirectional_duration_ms'] = df['bidirectional_last_seen_ms'] -␣
↪df['bidirectional_first_seen_ms']
```

[15]:
```
print("Standardizing numerical features...")
# Select numerical columns for standardization
numeric_cols = ['bidirectional_bytes', 'bidirectional_duration_ms',
       'bidirectional_mean_piat_ms', 'bidirectional_mean_ps',
       'bidirectional_packets', 'bidirectional_stddev_piat_ms',
       'bidirectional_stddev_ps', 'dst2src_mean_piat_ms', 'dst2src_mean_ps',
       'dst2src_stddev_piat_ms', 'dst2src_stddev_ps', 'fan_in_dst',
       'fan_in_src', 'fan_out_dst', 'fan_out_src', 'src2dst_mean_piat_ms',
       'src2dst_mean_ps', 'src2dst_stddev_piat_ms', 'src2dst_stddev_ps']

# Apply StandardScaler
scaler = StandardScaler()
df[numeric_cols] = scaler.fit_transform(df[numeric_cols])
```

Standardizing numerical features…

[16]:
```
print("Processing protocol information...")
# Handle protocol encoding
most_frequent_values = [6, 17, 89, 1, 132]  # Most common protocol numbers
df['protocol'] = df['protocol'].where(df['protocol'].
↪isin(most_frequent_values), 0)
df = pd.get_dummies(df, columns=['protocol'])
```

Processing protocol information…

[17]:
```
def get_port_cat(port):
    """
    Categorize ports based on IANA assignments:
    - Well-known ports: 0-1023
    - Registered ports: 1024-49151
    - Dynamic ports: 49152-65535
    """
    if 0 <= port <= 1023:
        return 'WellKnown'
    elif 1024 <= port <= 49151:
        return 'Registered'
    elif 49152 <= port <= 65535:
        return 'Dynamic'
    else:
```

```
        return 'Unknown'

print("Categorizing ports...")
# Apply port categorization
df['src_port_class'] = df['src_port'].apply(get_port_cat)
df['dst_port_class'] = df['dst_port'].apply(get_port_cat)
df = pd.get_dummies(df, columns=['src_port_class', 'dst_port_class'])
```

```
Categorizing ports…
```

[18]:
```
# Save encoded DataFrame
csv_filename = 'csv_files/encoded_flows.csv'
df.to_csv(csv_filename, index=False)
print(f'Encoded DataFrame saved to {csv_filename}')
```

```
Encoded DataFrame saved to csv_files/encoded_flows.csv
```

[19]:
```
def create_connection_pattern_features(df):
    """
    Create features based on connection patterns using fan-in/fan-out metrics
    and IP class relationships
    """
    print("Creating connection pattern features...")
    features = df.copy()

    # Fan-in/fan-out ratios
    features['fan_ratio_src'] = features['fan_out_src'] /␣
 ↪(features['fan_in_src'] + 1e-6)
    features['fan_ratio_dst'] = features['fan_out_dst'] /␣
 ↪(features['fan_in_dst'] + 1e-6)
    features['fan_total_src'] = features['fan_in_src'] + features['fan_out_src']
    features['fan_total_dst'] = features['fan_in_dst'] + features['fan_out_dst']

    # Connectivity features
    features['connection_asymmetry'] = np.abs(features['fan_total_src'] -␣
 ↪features['fan_total_dst'])
    features['connection_intensity'] = features['fan_total_src'] *␣
 ↪features['fan_total_dst']

    # IP class-based features
    features['ip_class_mismatch'] = (
        (features['src_ip_class_Class A'] & features['dst_ip_class_Class C']) |
        (features['src_ip_class_Class C'] & features['dst_ip_class_Class A'])
    ).astype(int)

    # Suspicious behavior detection based on IP classes
```

```
    features['potential_broadcast_attack'] = (
        features['dst_ip_class_Class D (multicast)'] &
        (features['bidirectional_packets'] > features['bidirectional_packets'].
↪quantile(0.95))
    ).astype(int)

    return features

def create_timing_features(df):
    """
    Create features based on timing characteristics
    """
    print("Creating timing features...")
    features = df.copy()

    # Timing ratios
    features['piat_ratio_src2dst'] = features['src2dst_mean_piat_ms'] /␣
↪(features['dst2src_mean_piat_ms'] + 1e-6)
    features['piat_ratio_stddev'] = features['bidirectional_stddev_piat_ms'] /␣
↪(features['bidirectional_mean_piat_ms'] + 1e-6)

    # Regularity features
    features['timing_regularity'] = 1 -␣
↪(features['bidirectional_stddev_piat_ms'] /
                                          ␣
↪(features['bidirectional_mean_piat_ms'] + 1e-6))

    # Burst detection
    features['burst_factor'] = (features['bidirectional_packets'] /
                                (features['bidirectional_duration_ms'] + 1e-6))

    return features

def create_protocol_features(df):
    """
    Create features based on protocols and ports
    """
    print("Creating protocol features...")
    features = df.copy()

    # Suspicious protocol/port combinations
    features['suspicious_port_protocol'] = (
        (features['protocol_17'] & features['dst_port_class_WellKnown'] &
         (features['bidirectional_packets'] <␣
↪features['bidirectional_packets'].quantile(0.1)))
    ).astype(int)
```

```
    # Protocol anomalies
    features['protocol_anomaly'] = (
        (features['protocol_0'] | features['protocol_1'] |␣
↪features['protocol_89'] | features['protocol_132'])
    ).astype(int)

    return features

def create_packet_features(df):
    """
    Create features based on packet characteristics
    """
    print("Creating packet features...")
    features = df.copy()

    # Packet size ratios
    features['ps_ratio_src2dst'] = features['src2dst_mean_ps'] /␣
↪(features['dst2src_mean_ps'] + 1e-6)
    features['ps_variation_ratio'] = features['bidirectional_stddev_ps'] /␣
↪(features['bidirectional_mean_ps'] + 1e-6)

    # Flow characteristics
    features['flow_efficiency'] = features['bidirectional_bytes'] /␣
↪(features['bidirectional_packets'] + 1e-6)
    features['flow_regularity'] = 1 - (features['bidirectional_stddev_ps'] /␣
↪(features['bidirectional_mean_ps'] + 1e-6))

    return features

def create_behavioral_features(df):
    """
    Create features for detecting specific behaviors
    """
    print("Creating behavioral features...")
    features = df.copy()

    # Scan detection
    features['potential_scan'] = (
        (features['fan_out_dst'] > features['fan_out_dst'].quantile(0.95)) &
        (features['bidirectional_packets'] < features['bidirectional_packets'].
↪quantile(0.05)) &
        (features['bidirectional_duration_ms'] <␣
↪features['bidirectional_duration_ms'].quantile(0.05))
    ).astype(int)
```

```
    # DDoS detection
    features['potential_ddos'] = (
        (features['fan_in_dst'] > features['fan_in_dst'].quantile(0.95)) &
        (features['bidirectional_packets'] > features['bidirectional_packets'].
↪quantile(0.95)) &
        (features['bidirectional_mean_piat_ms'] <␣
↪features['bidirectional_mean_piat_ms'].quantile(0.05))
    ).astype(int)

    # Data exfiltration detection
    features['potential_exfiltration'] = (
        (features['bidirectional_bytes'] > features['bidirectional_bytes'].
↪quantile(0.95)) &
        (features['dst2src_mean_ps'] < features['src2dst_mean_ps'] * 0.1)
    ).astype(int)

    return features

def create_final_features(df):
    """
    Combine all features into a single DataFrame
    """
    print("Combining all feature sets...")
    connection_features = create_connection_pattern_features(df)
    timing_features = create_timing_features(df)
    protocol_features = create_protocol_features(df)
    packet_features = create_packet_features(df)
    behavioral_features = create_behavioral_features(df)

    final_features = pd.concat([
        connection_features,
        timing_features,
        protocol_features,
        packet_features,
        behavioral_features
    ], axis=1)

    # Remove duplicate columns
    final_features = final_features.loc[:, ~final_features.columns.duplicated()]

    return final_features

print("Generating enriched feature DataFrame...")
enriched_features_df = create_final_features(df)
```

```
Generating enriched feature DataFrame…
Combining all feature sets…
```

```
Creating connection pattern features…

Combining all feature sets…
Creating connection pattern features…
Creating timing features…
Creating protocol features…
Creating packet features…
Creating behavioral features…
```

[20]:
```python
# Save the DataFrame with labels to a CSV file
output_path = 'csv_files/final_features_flows.csv'
enriched_features_df.to_csv(output_path, index=False)
print(f"Final DataFrame saved to {output_path}")
```

```
Final DataFrame saved to csv_files/final_features_flows.csv
```