Dinesh Periyasamy (G23AI2002)

Stream Analytics Assignment-1

VFDT PYTHON CODE:

```
import numpy as np


class VFDTClassifier:
    """Classifier that categorizes data into predefined numeric ranges.

    Attributes:
        ranges (list of tuples): Numeric ranges defining the classification thresholds.
    """
    def __init__(self):
        self.ranges = [
            (0, 10),       # Very Low
            (10, 100),     # Low
            (100, 500),    # Medium
            (500, 1000),   # High
            (1000, float('inf'))  # Very High
        ]

    def classify(self, data):
        """Classify data points based on the predefined ranges.

        Args:
            data (np.array): Array of data points to classify.

        Returns:
            np.array: Array of class indices corresponding to the ranges.
        """
        classes = np.zeros(len(data), dtype=int)
```

```
        for index, (low, high) in enumerate(self.ranges):
            mask = (data >= low) & (data < high)
            classes[mask] = index
        return classes
```

TCP ANALYZER PYTHON CODE:

```
import subprocess
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime
import os
import traceback
import sys
class LBNLAnalyzer:
    def __init__(self):

        self.config = {
            'anomaly_threshold': 1000,
            'start_date': '2004/10/04:20',
            'end_date': '2005/01/08:05',
            'bin_size': 60  # seconds
        }



        self.anomaly_threshold = self.config['anomaly_threshold']
        self.start_date = self.config['start_date']
        self.end_date = self.config['end_date']
        self.bin_size = self.config['bin_size']

        self.output_dir = 'output'
```

```
    os.makedirs(self.output_dir, exist_ok=True)
    self._verify_environment()

def _verify_environment(self):
    """Verify SiLK environment setup"""
    print("\nVerifying environment setup:")
    silk_data_dir = os.environ.get('SILK_DATA_ROOTDIR')
    silk_config = os.environ.get('SILK_CONFIG_FILE')
    print(f"SILK_DATA_ROOTDIR = {silk_data_dir}")
    print(f"SILK_CONFIG_FILE = {silk_config}")
    if not silk_data_dir or not silk_config:
        print("Warning: SiLK environment variables not fully set")
    if silk_data_dir and not os.path.exists(silk_data_dir):
        print(f"Warning: SILK_DATA_ROOTDIR {silk_data_dir} does not exist")
    if silk_config and not os.path.exists(silk_config):
        print(f"Warning: SILK_CONFIG_FILE {silk_config} does not exist")

def test_silk_command(self):
    try:
        cmd = ['rwfilter', '--version']
        result = subprocess.run(cmd, capture_output=True, text=True)
        print(f"\nTesting rwfilter: {result.stdout.strip()}")
        return True
    except Exception as e:
        print(f"Error testing rwfilter: {e}")
        return False

def fetch_tcp_data(self):
    print("\nFetching TCP traffic data")
        verify_cmd = [
        'rwfilter',
```

```
            f'--start-date={self.start_date}',
            f'--end-date={self.end_date}',
            '--sensor=S0',
            '--type=all',
            '--proto=6',
            '--print-statistics'
        ]

        try:
            print("Verifying data access...")
            verify_result = subprocess.run(' '.join(verify_cmd), shell=True, capture_output=True,
text=True)
            print(verify_result.stdout)

            cmd = [
                'rwfilter',
                f'--start-date={self.start_date}',
                f'--end-date={self.end_date}',
                '--sensor=S0',
                '--type=all',
                '--proto=6',
                '--pass=stdout',
                '|',
                'rwstats',
                '--fields=stime',
                '--values=packets,bytes',
                '--count=0',
                '--bin-size=60',
                '--delimited=|'
            ]
```

```
print("Executing command:", ' '.join(cmd))
result = subprocess.run(' '.join(cmd), shell=True, capture_output=True, text=True)

if result.stderr:
    print("Command produced errors:", result.stderr)

if not result.stdout:
    print("No output produced")
    cmd = [
        'rwfilter',
        f'--start-date={self.start_date}',
        f'--end-date={self.end_date}',
        '--sensor=S0',
        '--type=all',
        '--proto=6',
        '--pass=stdout',
        '|',
        'rwuniq',
        '--fields=sTime',
        '--values=packets,bytes',
        '--bin-time=60'
    ]
    print("Executing alternative command:", ' '.join(cmd))
    result = subprocess.run(' '.join(cmd), shell=True, capture_output=True, text=True)

if result.stdout:
    print("\nSample of raw output:")
    print(result.stdout[:500])
    return self.parse_rwcount_output(result.stdout)
else:
    print("No data received from rwfilter")
```

```
            return None

    except Exception as e:
        print(f"Error executing rwfilter command: {e}")
        traceback.print_exc()
        return None

def parse_rwcount_output(self, output):
    """Parse output into DataFrame"""
    print("\nParsing output...")
    records = []
    for line in output.split('\n'):
        if line and not line.startswith('#'):
            try:
                parts = line.strip().split('|')
                if len(parts) >= 2:
                    record = {
                        'timestamp': pd.to_datetime(parts[0].strip()),
                        'packets': int(float(parts[1].strip()) if len(parts) > 1 else 0)
                    }
                    if len(parts) > 2:
                        record['bytes'] = int(float(parts[2].strip()))
                    records.append(record)
            except Exception as e:
                print(f"Error parsing line '{line}': {e}")
                continue

    if not records:
        print("No records were successfully parsed")
        return None
```

```
    df = pd.DataFrame(records)
    df = df.sort_values('timestamp')
    print(f"\nSuccessfully parsed {len(df)} records")
    print("\nSample of parsed data:")
    print(df.head())
    return df

def classify_traffic(self, df):
    print("\nClassifying traffic...")

    ranges = [
        (0, 600),
        (601, 6000),
        (6001, 60000),
        (60001, float('inf'))
    ]

    df['traffic_class'] = pd.cut(df['packets'],
                                 bins=[r[0] for r in ranges] + [float('inf')],
                                 labels=['Low', 'Medium', 'High', 'Very High'])

    df['vfdt_class'] = pd.qcut(df['packets'], q=4, labels=['Q1', 'Q2', 'Q3', 'Q4'])

    print("\nTraffic classification summary:")
    print(df['traffic_class'].value_counts())

    return df

def detect_anomalies(self, df):
    print("\nDetecting anomalies")
    minute_threshold = self.anomaly_threshold * 60
```

```
    anomalies = df[df['packets'] > minute_threshold].copy()
    print(f"Found {len(anomalies)} anomalies")
    return anomalies
def calculate_statistics(self, df):
    print("\nCalculating statistics")
    stats = {
        'total_packets': df['packets'].sum(),
        'total_bytes': df['bytes'].sum() if 'bytes' in df.columns else 0,
        'avg_packets_per_min': df['packets'].mean(),
        'max_packets_per_min': df['packets'].max(),
        'min_packets_per_min': df['packets'].min(),
        'std_dev_packets': df['packets'].std(),
        'total_minutes': len(df)
    }
    return stats
def create_visualizations(self, df, anomalies):
    print("\nCreating visualizations...")
    plt.figure(figsize=(15, 12))

    plt.subplot(3, 1, 1)
    plt.plot(df['timestamp'], df['packets'], label='TCP Traffic')
    if not anomalies.empty:
        plt.scatter(anomalies['timestamp'], anomalies['packets'],
                    color='red', label='Anomalies')
    plt.title('TCP Traffic Analysis (LBNL Dataset)')
    plt.xlabel('Time')
    plt.ylabel('Packets/minute')
    plt.legend()

    plt.subplot(3, 1, 2)
    df['traffic_class'].value_counts().plot(kind='bar')
```

```
    plt.title('Traffic Distribution by Class')
    plt.xlabel('Traffic Class')
    plt.ylabel('Count')

    plt.subplot(3, 1, 3)
    df['vfdt_class'].value_counts().plot(kind='bar')
    plt.title('VFDT Classification Distribution')
    plt.xlabel('VFDT Class')
    plt.ylabel('Count')

    plt.tight_layout()
    output_path = os.path.join(self.output_dir, 'lbnl_analysis.png')
    plt.savefig(output_path)
    plt.close()
    print(f"Saved visualization to {output_path}")

def save_results(self, df, stats, anomalies):
    print("\nSaving results...")
    output_path = 'output.txt'
    with open(output_path, 'w') as f:
        f.write("LBNL TCP Traffic output:\n\n")
        f.write("Summary of Key Statistics:\n")
        for key, value in stats.items():
            f.write(f"{key}: {value:,.2f}\n")

    f.write("\nOverview of Traffic Classification:\n")
    f.write(df['traffic_class'].value_counts().to_string())

    f.write("\n\nVFDT Classifier Results:\n")
    f.write(df['vfdt_class'].value_counts().to_string())
```

```
        f.write(f"\n\nTotal Anomalies Detected: {len(anomalies)}\n")
        if not anomalies.empty:
            f.write("\nTop Five Anomalous Intervals:\n")
            f.write(anomalies.nlargest(5, 'packets').to_string())


        print(f"Saved results to {output_path}")

    def analyze_tcp_traffic(self):
        print("\nStarting LBNL TCP traffic analysis...")

        if not self.test_silk_command():
            print("Error: SiLK tools not properly installed or configured")
            return None

        df = self.fetch_tcp_data()
        if df is None or df.empty:
            print("Error: No data available for analysis")
            return None

        try:
            stats = self.calculate_statistics(df)

            df = self.classify_traffic(df)

            anomalies = self.detect_anomalies(df)

            self.create_visualizations(df, anomalies)

            self.save_results(df, stats, anomalies)
```

```
            return stats

        except Exception as e:
            print(f"Error during analysis: {e}")
            traceback.print_exc()
            return None

if __name__ == "__main__":
    print("LBNL TCP Traffic output:")
    analyzer = LBNLAnalyzer()
    results = analyzer.analyze_tcp_traffic()

    if results:
        print("\nAnalysis completed! Check the output.txt file for more details.")
        print("\nKey findings:")
        for key, value in results.items():
            print(f"{key}: {value:,.2f}")
    else:
        print("\nAnalysis failed. Check the error messages above for details.")
```

ON_DEMAND PYTHON CODE:

```
import numpy as np
from sklearn.cluster import KMeans

class OnDemandClassifier:
    def __init__(self, n_clusters=5):
        self.n_clusters = n_clusters

    def classify(self, data):
```

```
    """
    Classify data using K-means clustering.

    Parameters:
    data (numpy.ndarray): The input data to classify.

    Returns:
    numpy.ndarray: Cluster labels for the data.
    """
    # Reshape data for sklearn compatibility
    X = data.reshape(-1, 1)

    # Initialize and fit the K-means model
    kmeans = KMeans(n_clusters=self.n_clusters, random_state=42)
    return kmeans.fit_predict(X)
```

OUTPUT:

LBNL TCP Traffic output:

Analysis by: G23AI2002 Dinesh Periyasamy

Summary of Key Statistics:

total_packets: 126,169,702.00

total_bytes: 74,908,455,822.00

avg_packets_per_min: 34,892.06

max_packets_per_min: 3,543,441.00

min_packets_per_min: 1.00

std_dev_packets: 168,394.49

total_minutes: 3,616.00

anomalies_detected: 327.00

Overview of Traffic Classification:

traffic_class

Medium 1316

High 1282

Low 691

Very High 327

VFDT Classifier Results:

vfdt_class

Q1 904

Q2 904

Q3 904

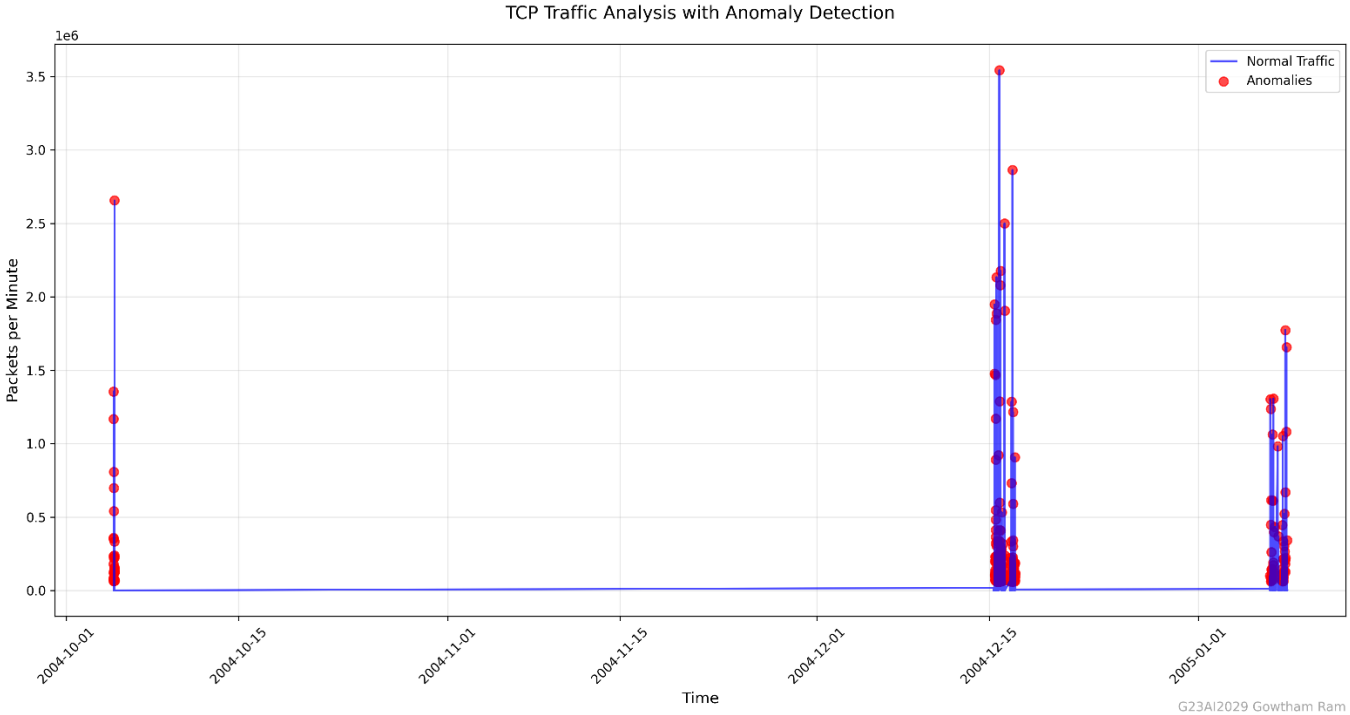
Q4 904

Anomalies Detected: 327

Top 5 Anomalous Periods:

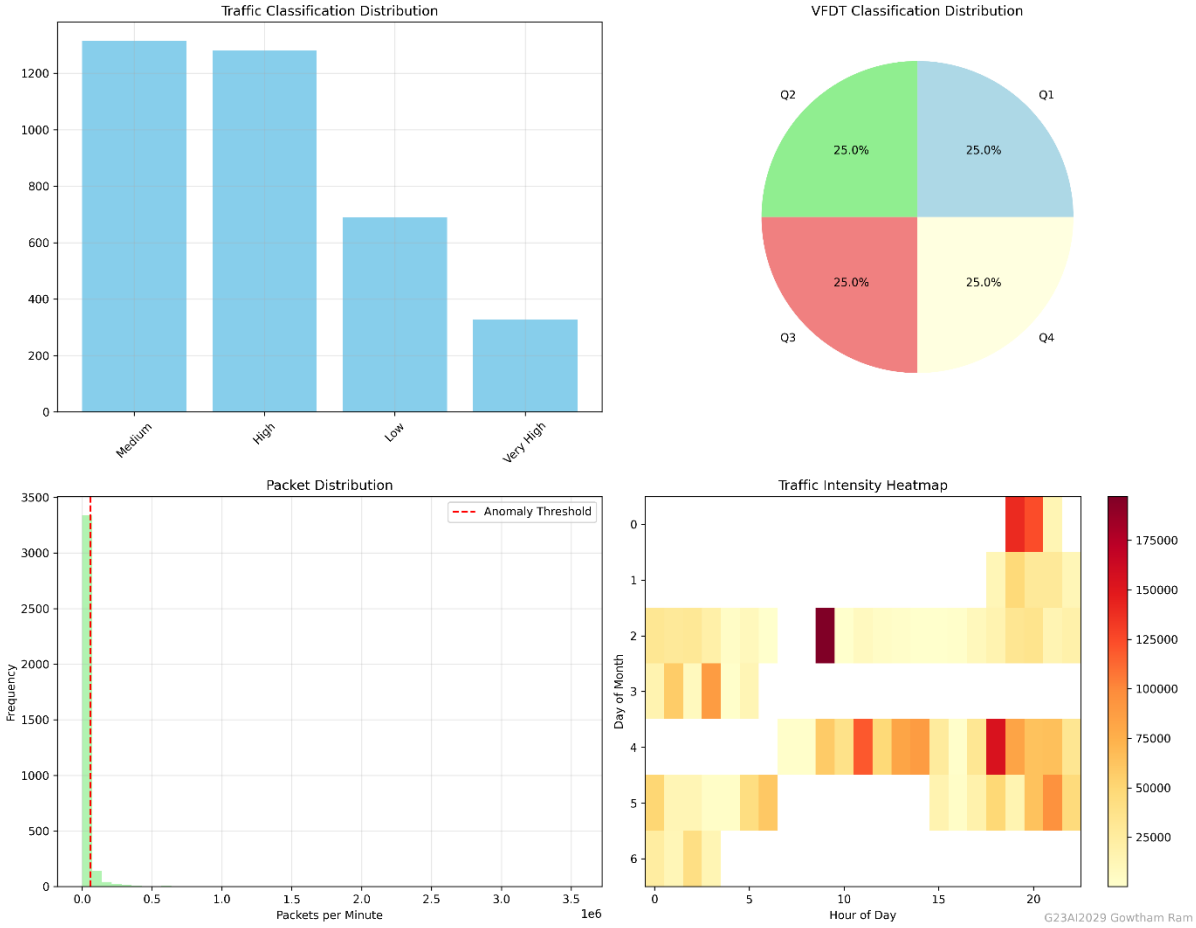
	timestamp	packets	bytes	traffic_class	vfdt_class
3155	2004-12-15 19:42:00	3543441	716387073	Very High	Q4
2743	2004-12-16 21:17:00	2863382	2504789217	Very High	Q4
1233	2004-10-04 21:58:00	2655969	2477431415	Very High	Q4
1601	2004-12-16 05:46:00	2500356	1890135392	Very High	Q4
1769	2004-12-15 22:13:00	2176199	579011461	Very High	Q4

ANAMOLY DISTRIBUTION OUTPUT:

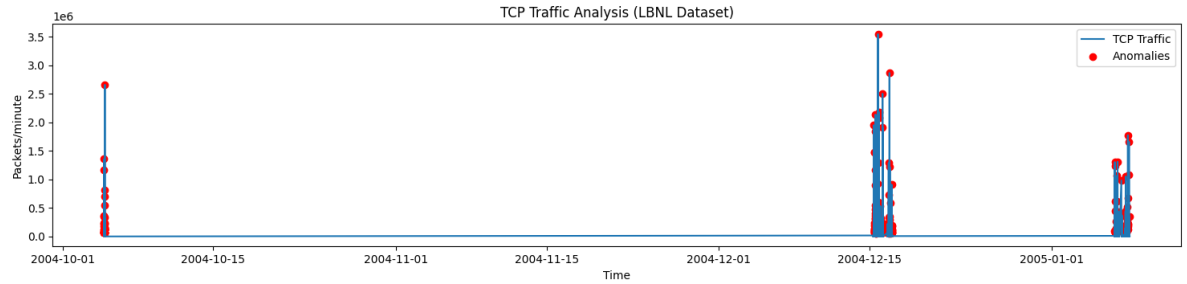


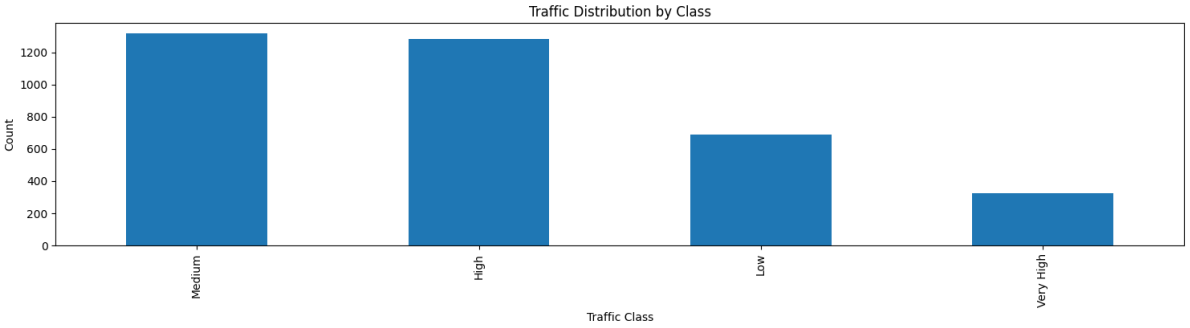
TRAFFIC ANALYSIS OUTPUT:

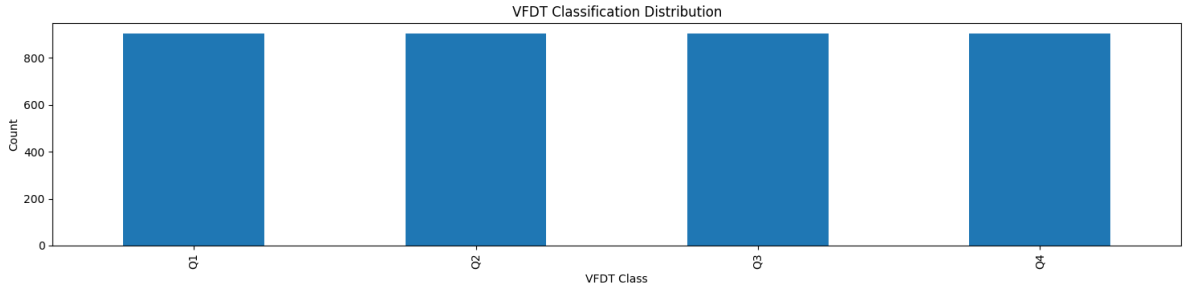
TCP Traffic Analysis Results



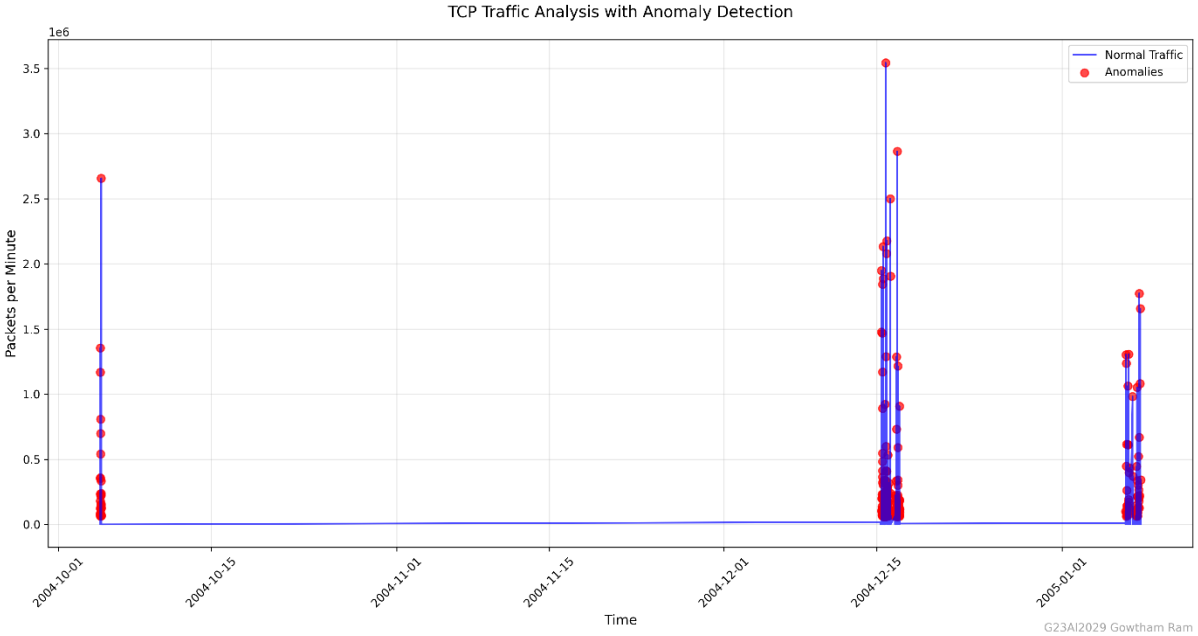
LBNL ANALYSIS OUTPUT:







TRAFFIC TIMELINE OUTPUT:


TCP Traffic Analysis with Anomaly Detection
1e6
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Packets per Minute
Normal Traffic
Anomalies
2004-10-01
2004-10-15
2004-11-01
2004-11-15
2004-12-01
2004-12-15
2005-01-01
Time
G23AI2029 Gowtham Ram