# Prototype Report: NetFlow Data Processing and Analysis

## Overview

The goal of this prototype is to preprocess, classify, and analyze network traffic data from `.pcap` files using NetFlow conversion techniques, machine learning models, and a robust data pipeline. The prototype integrates tools like CICFlowMeter for NetFlow conversion, RabbitMQ for distributed message handling, and PostgreSQL for result storage, with a focus on scalability and real-time data processing.

## Components

### 1. NetFlow Conversion

#### Description

The prototype converts `.pcap` files into NetFlow `.csv` format using CICFlowMeter (`cfm`). This tool is executed programmatically in Python, ensuring seamless integration into the pipeline.

#### Challenges

- CICFlowMeter requires execution from its `bin` directory.  
- Dependencies on Java (`JAVA\_HOME`) and shared libraries (`LD\_LIBRARY\_PATH`) must be correctly set for successful execution.

#### Solutions

- The Python code dynamically sets the working directory and environment variables before executing `cfm`.  
- Absolute paths are used to handle `.pcap` files reliably.

### 2. Preprocessing

#### Description

Preprocessing ensures that the NetFlow data is clean and structured for machine learning classification. This involves:  
- Validating required columns.  
- Dropping unused or irrelevant columns.  
- Handling missing, infinite, and invalid data values.  
- Normalizing feature data for better model performance.

#### Key Features

- Missing columns are added with default values.  
- Invalid rows (e.g., with labels like `NeedManualLabel`) are removed.  
- Normalization is applied to numerical columns using `MinMaxScaler`.

#### Challenges and Solutions

- Empty datasets after preprocessing:  
 - Added logic to handle empty datasets gracefully by skipping to the next file.  
 - Debugging output provides insights into data issues.  
- Row validation:  
 - Ensured a minimum of 10 rows remains after preprocessing; otherwise, the dataset is skipped.

### 3. Machine Learning Classification

#### Description

A pre-trained deep neural network model (`dnn-model.hdf5`) is used to classify network traffic into categories like 'Normal' or 'Threat'.

#### Workflow

- The preprocessed data is split into features (`X`) and labels (`Y`).  
- Predictions are mapped back to labels and saved into the database.

#### Challenges

- Ensuring feature compatibility with the trained model.  
- Handling edge cases where the input dataset is insufficient for model inference.

### 4. Distributed Messaging with RabbitMQ

#### Description

RabbitMQ handles `.pcap` file ingestion, ensuring distributed and asynchronous processing. The callback function processes each received file, including conversion, preprocessing, and classification.

#### Challenges

- Maintaining robust error handling to avoid pipeline crashes.  
- Efficient message acknowledgment for processed files.

### 5. Database Integration

#### Description

Classified results are stored in a PostgreSQL database for later analysis and reporting.

#### Implementation

- Used `psycopg2` for database interactions.  
- Results are batch-inserted into the `dash\_trafficlog` table.

#### Challenges

- Handling schema mismatches and ensuring column names are SQL-compliant.

## Current Progress

### Completed Tasks

1. Integrated CICFlowMeter for `.pcap` to `.csv` conversion.  
2. Built a preprocessing pipeline for cleaning and normalizing NetFlow data.  
3. Incorporated RabbitMQ for distributed message handling.  
4. Established PostgreSQL integration for result storage.  
5. Implemented logic to handle edge cases like empty datasets and missing columns.

### Pending Tasks

1. Enhance real-time performance through further optimization.  
2. Expand the machine learning model to support multi-class classification.  
3. Implement detailed reporting and visualization of results.  
4. Conduct end-to-end testing with large-scale datasets.

## Challenges Encountered

1. Dependency Issues:  
 - CICFlowMeter's dependency on Java and library paths.  
 - Resolved by dynamically setting environment variables.  
2. Data Quality:  
 - Handling incomplete or corrupted `.pcap` files.  
 - Introduced robust validation and fallback mechanisms.

## Future Enhancements

1. Automate dataset quality checks before processing.  
2. Add support for additional input formats beyond `.pcap`.  
3. Implement a web-based dashboard for real-time monitoring and visualization.  
4. Scale the system to handle thousands of `.pcap` files concurrently.

## Dependencies

1. CICFlowMeter: A tool for converting `.pcap` files into NetFlow `.csv` format.  
2. Java Runtime Environment: Required for CICFlowMeter execution. Ensure `JAVA\_HOME` is set to `/usr/lib/jvm/jdk-23.0.1-oracle-x64`.  
3. Python Libraries:  
 - `pandas` for data manipulation.  
 - `numpy` for numerical computations.  
 - `scikit-learn` for data normalization.  
 - `psycopg2` for PostgreSQL database interaction.  
 - `pika` for RabbitMQ messaging.  
4. RabbitMQ: For distributed message handling.  
5. PostgreSQL: For storing classified results.

## Running Sequence

1. Set Up Environment:  
 - Install all dependencies (e.g., Python libraries, RabbitMQ, PostgreSQL, CICFlowMeter).  
 - Configure environment variables:  
 - `JAVA\_HOME` to `/usr/lib/jvm/jdk-23.0.1-oracle-x64`.  
 - `LD\_LIBRARY\_PATH` to include the required libraries.  
2. Start RabbitMQ:  
 - Ensure RabbitMQ is running to handle `.pcap` file ingestion.  
3. Execute Prototype:  
 - Run the main Python script to start processing `.pcap` files received via RabbitMQ.  
4. Monitor Logs:  
 - Check logs for processing status and errors.  
5. Analyze Results:  
 - Access classified results stored in the PostgreSQL database.

## Conclusion

This prototype demonstrates a robust pipeline for processing and analyzing network traffic data. By addressing real-world challenges and implementing scalable solutions, it lays a strong foundation for further enhancements and applications in network security and traffic analysis.