BotNet Detection Using Apache Spark GraphFrames

Professor: **Dr. Zeinab Zali**

Created by: **Arash Azhand**

Table of Contents

Introduction:

With the rapid growth of network infrastructures and Internet-connected devices, network security has become one of the most critical challenges in computer networks. Detecting malicious activities, such as botnet attacks, plays a crucial role in protecting network resources and preventing data breaches. Traditional intrusion detection systems often rely on flow-based or signature-based approaches; however, analyzing the structure and behavior of network traffic using graph models offers a new perspective for identifying abnormal communication patterns.

In this project, I focus on analyzing network traffic as a graph in order to detect central or potentially suspicious nodes (e.g., botmasters or infected hosts). I leverage Apache Spark GraphFrames, a powerful distributed graph analytics library, to handle large-scale data efficiently and perform advanced graph-based computations such as degree centrality and PageRank.

Dataset selection and download process:

In the original project proposal, I planned to use the CSE-CIC-IDS2018 dataset. This dataset is widely used for intrusion detection and includes a variety of modern attack scenarios. However, after a careful inspection of the provided CSV files, I found that source and destination IP addresses were not explicitly included in the processed CSV files (CICFlowMeter outputs). Instead, they mainly contained aggregated flow-level statistics such as packet lengths, flow durations, and protocol types, without clear identifiers for individual hosts.

Since our main objective is to model communication as a graph based on source and destination IPs, these columns are essential for creating nodes and edges. Without them, it is impossible to construct an accurate network communication graph.

To address this limitation, I decided to use the CTU-13 dataset, specifically Scenario 42 (Neris botnet), which provides detailed bidirectional flow summaries, including source and destination IP addresses. The CTU-13 dataset was developed by the Stratosphere Laboratory at Czech Technical University and is commonly used for botnet detection research.

I downloaded Scenario 42 from the official Stratosphere Lab repository. Instead of using the raw PCAP files (which require heavy parsing and are large in size), I chose the pre-processed bidirectional flow file (capture20110810.binetflow.2format) available in the detailed-bidirectional-flow-labels folder. This file contains all necessary flow-level attributes, including SrcAddr, DstAddr, total packets, total bytes, and a Label column that specifies whether the flow is normal, botnet, or background traffic.

Setting up the distributed Spark environment:

After choosing and preparing the CTU-13 dataset, I needed to set up a distributed processing environment to use Spark and the GraphFrames library for graph-based analysis of network traffic data.

Issue with running Spark locally on Windows

At first, I tried to install and run Spark locally on my Windows machine. However, due to common configuration issues like missing winutils.exe and environment variable setup problems, I couldn't get Spark to run properly.

Solution: Using Docker

To solve this, I decided to use **Docker**. I pulled a pre-built image that includes Jupyter Notebook and PySpark (jupyter/pyspark-notebook) which gave me a clean and reliable environment without complex local setup.

Steps I followed

1. Downloaded the Docker image with:

   ```
   docker pull jupyter/pyspark-notebook
   ```

2. Ran the container and mapped port 8888 with:

   ```
   docker run -p 8899:8888 jupyter/pyspark-notebook
   ```

3. Used the token provided in the Docker logs to access Jupyter Lab in my browser.

Uploading dataset and initializing Spark

Inside Jupyter Lab, I uploaded the CTU-13 dataset file (capture20110810.csv) and created a new Python notebook. Then, I started a SparkSession to load and process the dataset as a Spark DataFrame.

Environment Setup

At the start of the project, I created a Spark session and configured it to include the GraphFrames package. GraphFrames is an extra library that allows me to work with graph structures (nodes and edges) inside Spark, which is necessary for modeling network flows as a graph. I also set the driver and executor memory to 8 GB to avoid memory errors, since the CTU-13 dataset is large and graph algorithms like PageRank and degree calculations are memory-intensive. This setup let me analyze large-scale network traffic in a distributed and efficient way.

Data Loading and Cleaning

I loaded the CTU-13 dataset and explored its columns, including source IP (SrcAddr), destination IP (DstAddr), direction (Dir), and detailed labels (Label). I simplified the label column into three main classes: Background, Normal, and Botnet. This was important because the original dataset had many detailed labels, but for anomaly detection and classification, a simpler class definition helps make the analysis and model training clearer.

Graph Construction

I modeled the network data as a graph where IP addresses are vertices (nodes) and connections (flows) between them are edges. I extracted source and destination addresses to define the edges, and included edge features like total packets, total bytes, and duration. This graph-based approach helped me understand communication patterns and node importance in the network.

Graph Feature Engineering

I calculated graph metrics such as degree (number of connections per node) and PageRank (importance score) to characterize each IP. Nodes with high degree or high PageRank often indicate central or suspicious roles in the network. These features were later merged back into the edge-level data so I could use them as input features for classification.

Node and Label Analysis

I analyzed the distribution of the labels within high-degree and high-PageRank nodes. I computed how many connections for each IP belonged to each label and calculated a Botnet ratio, which helped identify IPs that are mostly involved in botnet activities. This gave me a first idea of which IPs might be anomalous or potentially compromised.

Edge Feature Creation for Machine Learning

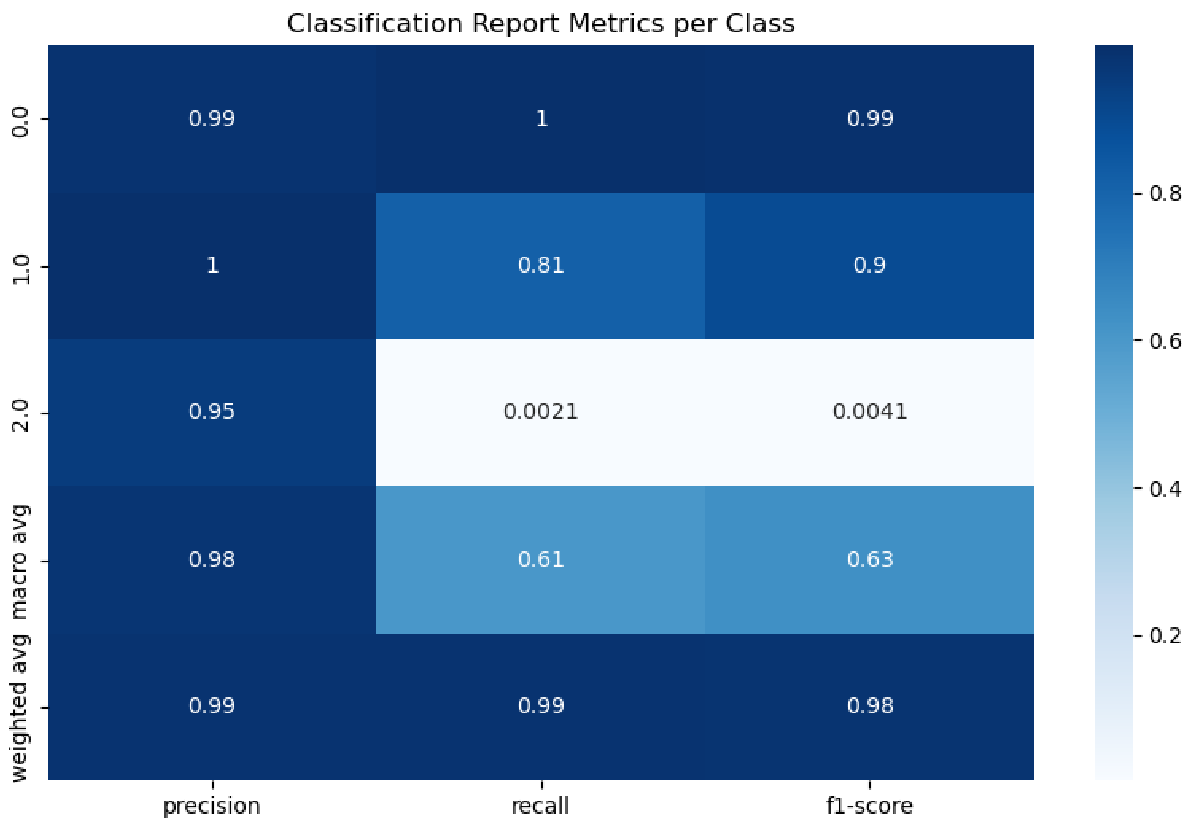
I created a new dataframe for machine learning, using both graph-based features (like source node degree and PageRank) and connection-level features (total packets, bytes, duration). I also encoded categorical features such as protocol and direction into numerical indices to prepare the data for model training.

Train-Test Split and Model Training

I split the data into training and testing sets (70% training, 30% testing) and used a Random Forest classifier to predict the simplified label. I chose Random Forest because it can handle mixed types of features and works well even when some features are not perfectly scaled or linear.

Evaluation and Results

After training, I evaluated the model using F1-score instead of just accuracy, since the classes were imbalanced (Background was much more frequent). I also generated a classification report to better understand precision, recall, and F1-score for each class. I visualized this using a heatmap to easily spot how well the model identified each class. The results showed that the model performed very well on Background and Botnet classes, but had difficulty detecting Normal connections, mainly because of their low representation and possible feature similarity to Background traffic.



Conclusion

Despite the huge scale of the network graph and the computational complexity involved, using Spark GraphFrames enabled me to perform advanced graph analytics and feature extraction in a very efficient and scalable way. Tasks that would be prohibitively slow or impossible on a single machine were handled smoothly in a distributed environment, significantly speeding up the analysis and making it feasible to work with large real-world network traffic data. This demonstrates the power of combining big data processing frameworks with graph analytics for cybersecurity applications like botnet detection.