Realtime Detection of DDoS Attacks in Cloud Networks Using Online Random Forest ML Model

Table of Content

1. Introduction

In order to achieve the objectives of this study, to develop a machine learning model for the detection of DDoS attacks in cloud networks, the following steps are implemented; The Online Random Forest (ORF) model will be built and trained with datasets of traffic streams comprising both normal and DDoS traffic using CAIDAand CICIDS traffic training datasets (CIC-DDoS2019). In addition to the pretrained model, Apache Kafka would also be used to simulate realtime traffic streams, with Apache Flink processing the data and performing feature preprocessing/extraction before the preprocessed traffic stream is finally fed into the ORF for classification.

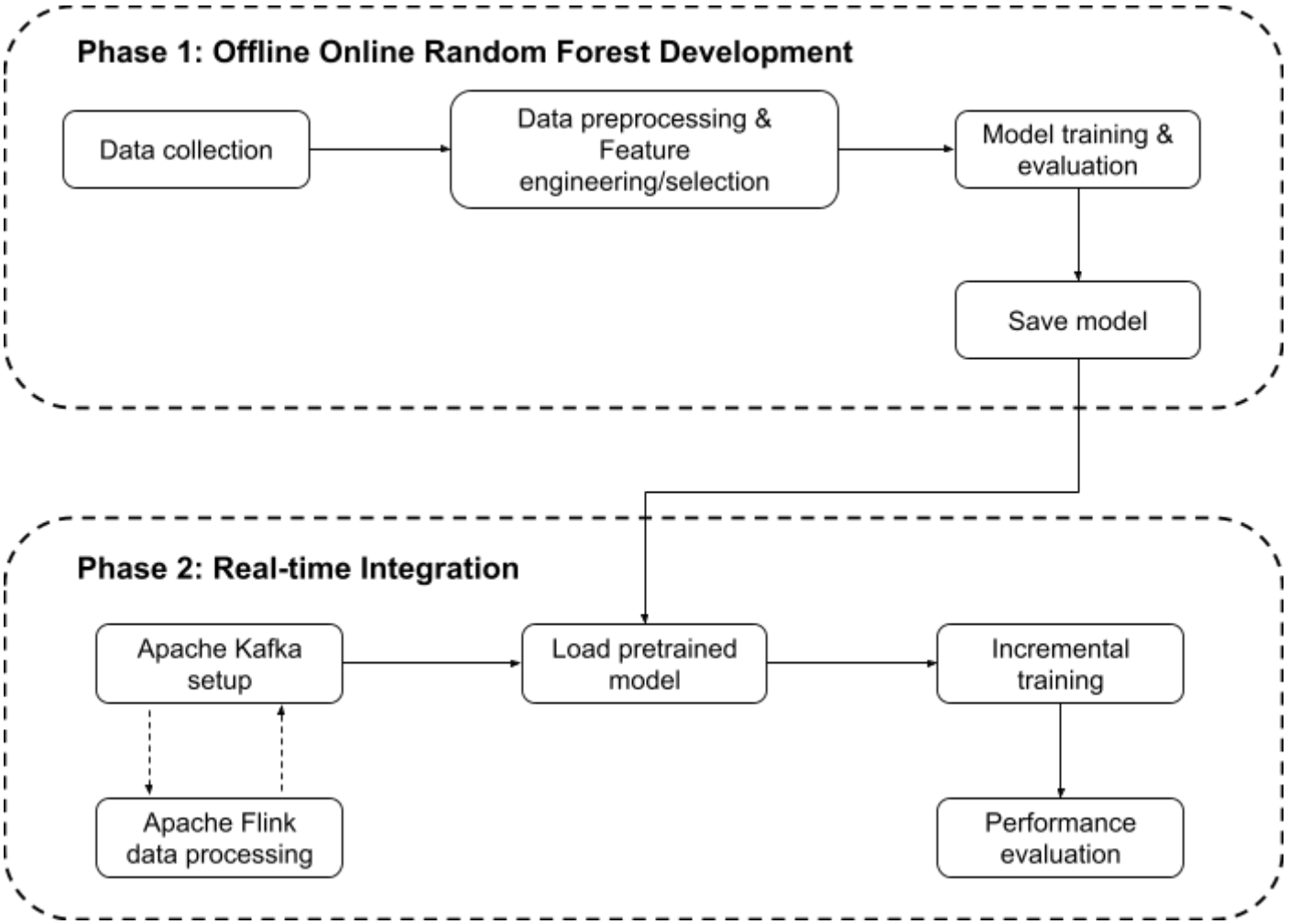

Diagram 1.1: Model implementation flow diagram

After the model development and implementation, a realistic realtime simulation of the implementation will include deploying the ORF model to the cloud, Openstack in this case, using FastAPI where incremental learning will continue, ensuring the model is flexible and constantly adapting to changing patterns in attack traffic.

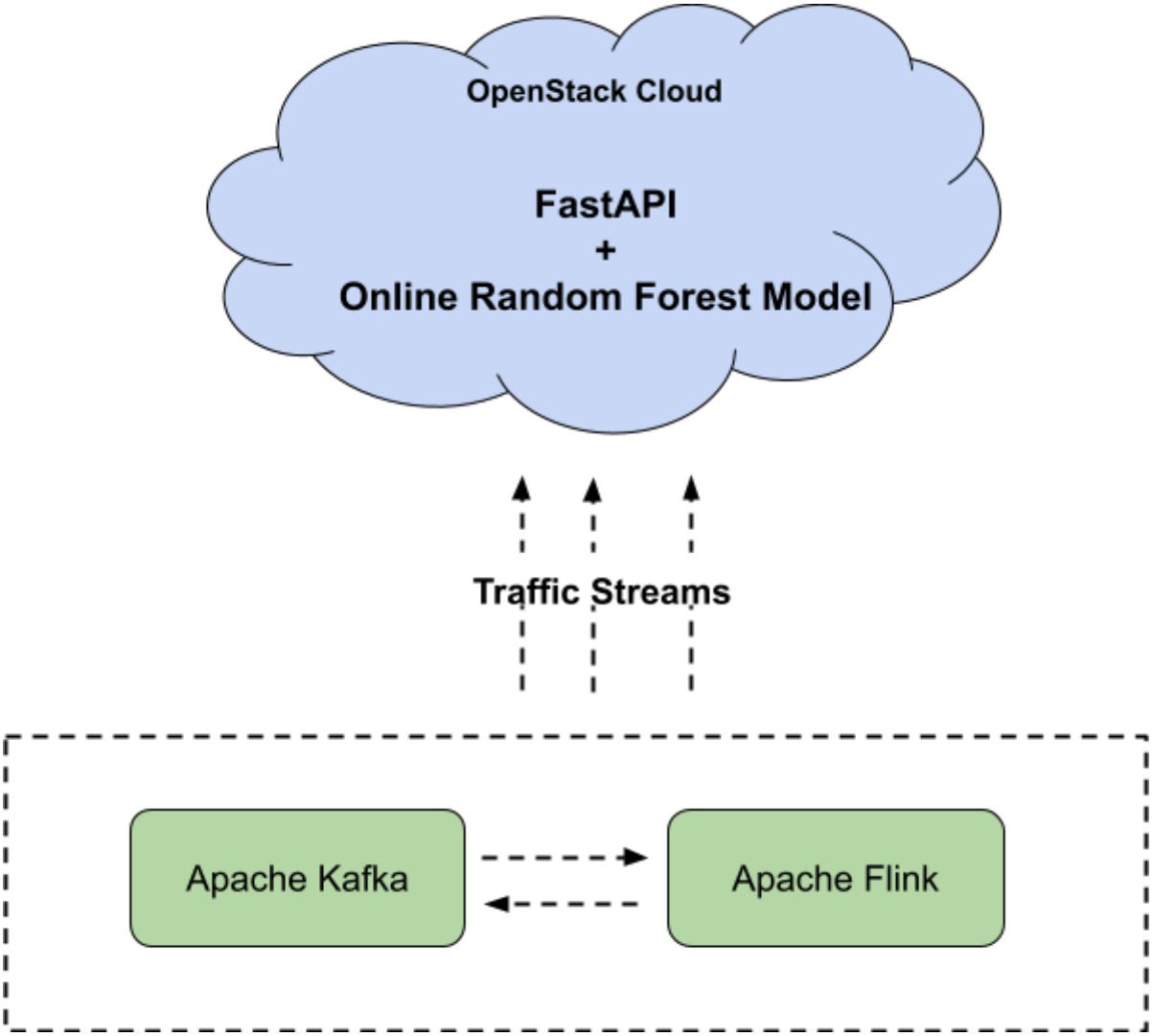

Image 1.1: Architectural diagram of the realtime system

Before we begin the implementation of the objectives of this research, we are going to install several Python library dependencies in a Python virtual environment to ensure the smooth running of the processes we will build.

Project Structure and Setup

Create a new folder called **Project**. Inside the Project folder, we will create 2 more folders; **datasets** and **models** to hold the datasets and saved trained model respectively later.

The folder structure should look like this:

Project - datasets

- models

Then navigate into the Project directory you just created.

Create a Python Virtual Environment

We will create a venv and name it **project-venv.**

Command: **`Python3 -m venv project-venv`**

Next, we will activate the virtual environment we just created and begin the installation of the required libraries.

Command: **`source project-venv/bin/activate`**

Image 1.2: Terminal showing activate venv

Also, we will make sure the pip version in the venv is upto date by running the following command: **`pip install --upgrade pip setuptools`**

Install Python Libraries
The required python package dependencies include:

1. pandas
2. kafka-python
3. apache-flink
4. scikit-learn
5. numpy
6. matplotlib
7. seaborn
8. scipy

Execute the following command from the terminal window to download and install them:

```
pip install pandas kafka-python apache-flink scikit-learn numpy matplotlib seaborn scipy
```

2. Objective One (1): Implementing Online Random Forest

2.1. Introduction

In this section, we will first download the cicddos2019 dataset, then analyse the data to get insight from the data in order to ensure we clean and process the data in the appropriate manner suitable for the machine learning classifier model, Online Random Forest (ORF). Subsequently, we will perform data visualization, cleaning and preprocessing before using it to train the ORF model both in the offline (static) phase, and incremental learning phase.

2.2. Data collection

Navigate to the following URL to download the cicddos2019 dataset that has been made available through Mendeley Data. URL: https://data.mendeley.com/datasets/ssnc74xm6r/1
Move the downloaded cicddos2019.csv file into the **Project** directory.

With the dataset download and ready for use, we will begin by analysing and assessing the dataset. We will be utilizing the Jupyter notebook tool for this purpose. Install Jupyter notebook on your machine by running the following command: **`pip install jupyter`**
To run jupyter notebook, run the command from your terminal: **`jupyter notebook`**
This will open the notebook in a web browser as shown in the image below.

Image 2.1: Jupyter notebook interface

2.3. Data Visualization

In this step, we explore the collected dataset to uncover insights which will guide us along the way when building out the ML model itself, for example insights about the features will show us which ones have more impact in the model's decision, thereby guiding us in which features to discard etc.

2.3.1. Raw Dataset Information

We will load the dataset to get an overview of the data such as:

- ➔ Column names
- ➔ Data types
- ➔ Missing values
- ➔ Sample rows

From Jupyter notebook we will run the commands as follows:

```
import pandas as pd

# Load the dataset
file_path = "cicddos2019.csv"  # Replace with your dataset path
raw_data = pd.read_csv(file_path)

# Display basic info about the dataset
print(raw_data.info())
print(raw_data.head())
```

[image from jupyter notebook]

2.3.2. Visualizing Missing Data

In order to prevent errors and/or bias in the training of the model, it is important to check for missing values in the dataset. This will also provide insight on how to deal with the missing values, whether to drop the feature or fill them with the appropriate replacements.

```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
sns.heatmap(raw_data.isnull(), cbar=False, cmap="viridis")
plt.title("Missing Data Heatmap")
plt.show()
```

[Image from jupyter notebook]

2.3.3. Class Imbalance

In some cases, when the dataset classes (i.e DDoS vs Normal traffic samples) are imbalance, it can cause the model to favour the majority class while subsequently creating a poor generalization for the minority group especially in this kind of DDoS datasets where malicious groups are often the minority.
This will help us to decide if techniques like oversampling, undersampling, or class weighting will be needed.

```
plt.figure(figsize=(8, 5))
sns.countplot(x="Label", data=raw_data, palette="coolwarm")
plt.title("Class Distribution")
plt.xticks(rotation=45)
plt.show()
```

[jupyter notebook image]

2.3.4. Correlation Heatmap

Sometimes in the dataset, some numerical features may have very high correlations which can result in redundancy and lead to overfitting of the model, it is necessary to explore this possibility and determine if it is appropriate to exclude such features.

```
# Compute correlation matrix for numerical features
corr_matrix = raw_data.corr()

plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=False, cmap="coolwarm")
plt.title("Feature Correlation Heatmap")
plt.show()
```

[jupyter image]

2.3.5. Detecting Outliers

Outliers are features that have very significant difference from the rest of the features or values in its category or group. These can skew the model and lead to over- or underfitting of the model.

```
plt.figure(figsize=(10, 6))
sns.boxplot(x=raw_data["Flow Bytes/s"])
plt.title("Boxplot of Flow Bytes/s (Outliers Detection)")
plt.show()
```

[jupyter image]

2.3.6. Feature Distributions

We will plot histograms for the features to detect how they are distributed and the nature of their skewness, this will inform us on whether to apply scalling or transformations to the dataset features.

```
selected_columns = [
    'Flow Duration', 'Total Fwd Packets', 'Total Backward Packets',
    'Flow Bytes/s', 'Flow Packets/s', 'Fwd Packet Length Mean',
    'Bwd Packet Length Mean', 'Packet Length Mean', 'Packet Length Std'
]

raw_data[selected_columns].hist(bins=30, figsize=(15, 10), color="blue")
plt.suptitle("Feature Distributions", fontsize=16)
plt.show()
```

[jupyter image]

2.3.7. Scatter Plot Matrix

The scatter plot matrix will help us gain insight into how pairs of features may be related to indicate and identify clusters, linearity, or multicollinearity.

```
from pandas.plotting import scatter_matrix

scatter_matrix(raw_data[selected_columns[:5]], figsize=(12, 8),
diagonal="kde")
plt.suptitle("Scatter Plot Matrix")
plt.show()
```

[jupyter image]

2.3.8. Observations and Conclusion

2.4. Data Preprocessing

3. Objective Two (2): Implementing Apache Kafka

3.1. Introduction

In this section, we will need to generate custom traffic/load to simulate both malicious and normal traffic for model training alongside the dataset that will be obtained from public repositories mentioned in previous sections. We will use Apache Kafka for this purpose.
After assessing these outliers, we can then decide if we want to handle them by capping, transformation, or removal.