DDOS-ATTACK-DETECTION-AND-MITIGATION-IN-SDN-USING-MACHINE-LEARNING

TEAM NAME:

The Team name is " FLY HIGH ".

COLLEGE NAME AND MENTOR:

College Name: J.N.N INSTITUTE OF ENGINEERING

Mentor: Swaminathan N - swaminathann@jnn.edu.in

TEAM MEMBERS:

Sony Jenith D - dsony15@jnn.edu.in (Leader)

Prabhakar R - rprabhakar40@jnn.edu.in

Kishori Sowmya G V - skishori15@jnn.edu.in

Kaviya Priya M - mkaviyapriya36@jnn.edu.in

Reshmashree N - mreshmasree30@jnn.edu.in

PROBLEM STATEMENT:

AI/ML for networking or Network Security.

DDOS ATTACK DETECTION:

Deep belief network feature detection.

And

LSTM MODEL

(Feature Extraction)

+

Random Forest ML Model.

(Machine Learning Algorithm)

||

DDOS ATTACK DETECTION AND MITIGATION.

DDOS-ATTACK-DETECTION-AND-MITIGATION-IN-SDN-USING-MACHINE-LEARNING

INTRODUCTION:

A Distributed Denial-of-Service (DDoS) attack is a malicious attempt to disrupt the normal functioning of a targeted server, service, or network by overwhelming it with a massive flood of internet traffic. These attacks leverage multiple compromised devices ranging from traditional computers to IoT devices acting in unison to generate traffic that exhausts the target's resources. At a high level, a DDoS attack resembles an unexpected traffic jam that clogs a highway, preventing legitimate users from reaching their destination. Prompt detection and response are critical, as delays can lead to server crashes and prolonged downtime. Mitigating such attacks is especially challenging because attackers disguise malicious traffic to appear legitimate, making it difficult to block harmful requests without affecting real users.

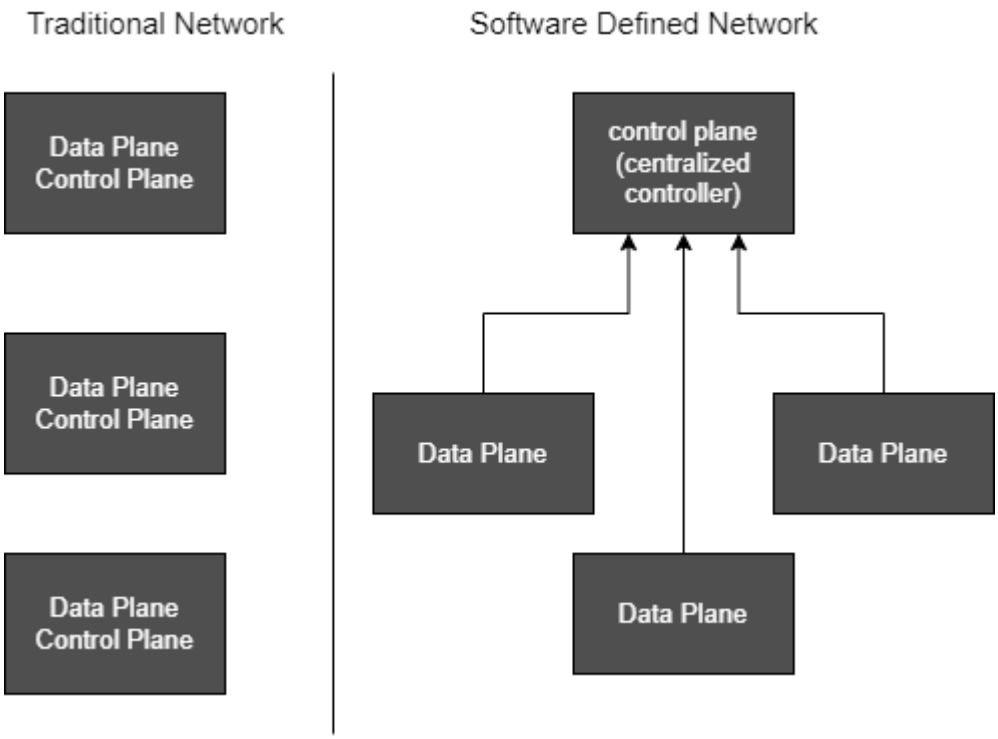
PROJECT OBJECTIVES:

- **Implement a Software-Defined Network (SDN) testbed** using Mininet for network emulation and the Ryu controller for centralized network management. This setup enables flexible experimentation with custom network topologies and real-time traffic control.
- **Generate a comprehensive network traffic dataset** by simulating both normal and DDoS attack scenarios within the Mininet environment. The dataset will include labeled traffic flows, which are essential for training and evaluating machine learning models.
- **Apply the Random Forest machine learning algorithm** to the collected dataset to accurately detect DDoS attacks. The model will be trained to distinguish between legitimate and malicious traffic based on extracted network features.
- **Integrate a mitigation module** within the Ryu controller to respond to detected DDoS attacks. This module will automate defensive actions, such as blocking malicious flows or rate-limiting suspicious hosts, to minimize service disruption and maintain network availability.

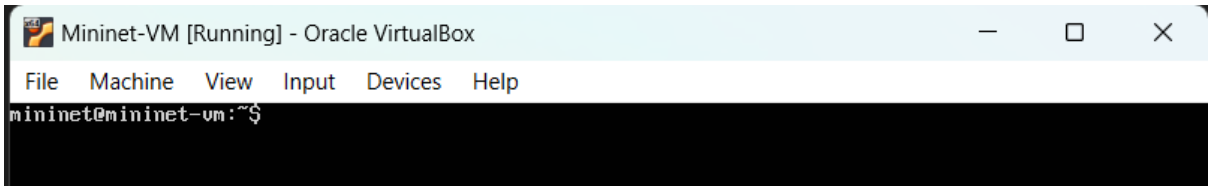
SOFTWARE DEFINED NETWORKING:

Software Defined Networking (SDN) is a modern approach to network management that separates the control of network traffic (the control plane) from the actual forwarding of traffic (the data plane). In SDN, a centralized software controller

manages the entire network, making decisions about how data should flow, while network devices like switches and routers simply forward packets based on those instructions.



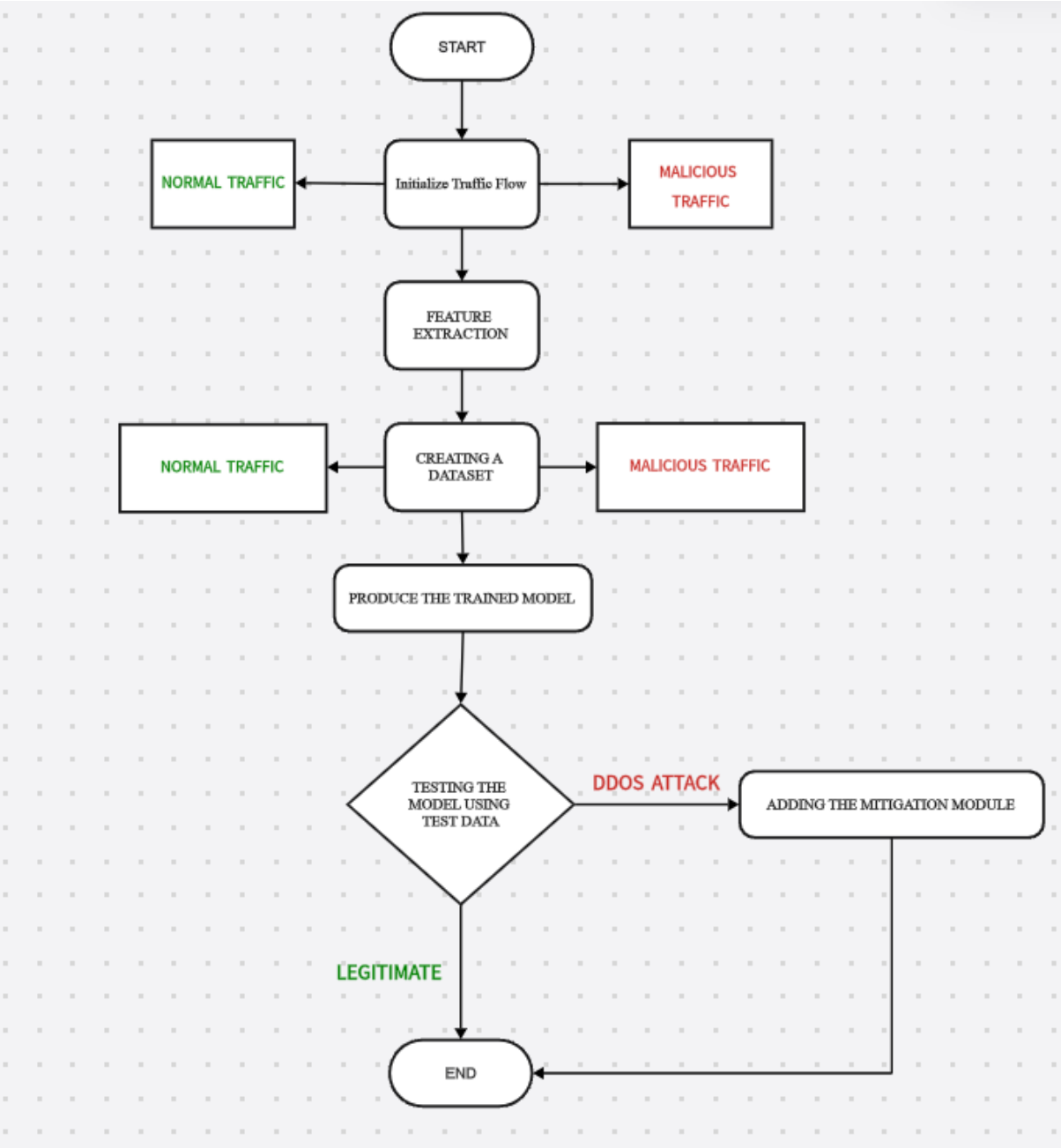
DATA PLANE AND CONTROL PLANE:



In this setup, the Mininet VM functions as the **DATA PLANE** within the Software Defined Networking (SDN) architecture. In SDN, the data plane is responsible for the actual transmission and forwarding of data packets across the network.

The Ryu controller serves as the **CONTROL PLANE** in a Software Defined Networking (SDN) environment. It centrally manages the network by making all routing and forwarding decisions, communicating these instructions to the underlying data plane devices such as virtual switches and hosts. By leveraging open protocols like OpenFlow, Ryu dynamically installs and updates flow rules, enabling real-time control and flexible policy enforcement across the network. This centralized intelligence allows for efficient traffic management, rapid adaptation to network changes, and streamlined implementation of advanced features, making Ryu an essential component in SDN-based architectures.

ARCHITECTURE FLOW:



ARCHITECTURE OF THE SYSTEM:

Software-Defined Networking (SDN) is a network architecture approach that enables the network to be intelligently and centrally controlled, or 'programmed,' using software applications.

Three Layer Architecture of SDN:

- **Application Layer:** This is where network applications (such as management, security, and optimization tools) reside. These applications communicate their requirements and policies to the control layer.
- **Control Layer:** This layer hosts the centralized SDN controller, which acts as the network's "brain." The controller receives instructions from the application layer and translates them into policies and rules for the network devices.
- **Infrastructure Layer:** This consists of the physical and virtual switches or routers that forward traffic according to the controller's instructions. These devices do not make independent routing decisions but execute commands from the control layer.

DATASET FEATURE INFORMATION:

The dataset for this study was generated by simulating both legitimate and DDoS network traffic within a Mininet virtual environment. Legitimate traffic and DDoS attack scenarios were created using custom Python scripts executed on Mininet hosts. All network activity was monitored and captured by the Ryu SDN controller, which collected detailed flow statistics and packet information in real time

- Dataset size = 22 columns x 1048576 rows
- Data description:
- Ip_source - source IP address
- Ip_dst - destination IP address
- Icmp_type
- Flow duration
- Idle time
- Packet_count
- Label - classify malicious or legitimate traffic(1=legitimate & 0=malicious)

IMPLEMENTATION METHODOLOGY:

The Implemented system consists of a detection module and a mitigation module. The model is trained on the dataset that is generated .To generate the dataset, a topology in SDN using mininet. simulated network traffic(normal traffic and malicious traffic) is created. Then feature's source IP, destination IP, port number, and timestamp from the packets using the RYU controller are extracted.

FILE ARCHITECTURE:

```
DDOS_ATTACK-main/
│
├── README.md
└── src/
    ├── mininet/
    │   ├── benign_traffic_generator.py
    │   ├── ddos_traffic_generator_1.py
    │   ├── ddos_traffic_generator_2.py
    │   └── topology.py
    │
    ├── Mitigation-KNN/
    │   ├── mitigation_module.py
    │   └── switchm.py
    │
    ├── ml-models/
    │   ├── DT.py
    │   ├── KNN.py
    │   ├── LR.py
    │   ├── ML_MAIN.py
    │   ├── NB.py
    │   ├── RF.py
    │   └── SVM.py
    │
    └── SDN-Controller/
        ├── controller.py
        ├── ddos_traffic_collector.py
        ├── mitigation_module.py
        ├── switch.py
        ├── switchm.py
        └── traffic_collector.py
```

MODULE DESCRIPTION:

ML-Models: (Worked by Prabhakar, Sowmya, Sony Jenith)

The detection module in this system is designed to evaluate and compare the performance of six distinct machine learning algorithms for network traffic classification. The implemented models include Decision Tree (DT), K-Nearest Neighbors (KNN), Logistic Regression (LR), Naive Bayes (NB), Random Forest (RF), and Support Vector Machine (SVM). Each algorithm is encapsulated in its own Python script (e.g., DT.py, KNN.py, etc.), ensuring a modular and organized approach to experimentation and evaluation. During development, all models were trained and tested on the same dataset to ensure a fair comparison. Among these, the Random Forest (RF) model demonstrated the highest accuracy and overall performance, making it the preferred choice for the final deployment in the detection pipeline.

Traffic-Control-Module: (Worked by Prabhakar, Kaviyapriya, Sowmya, Sony Jenith, Reshma)

ddos_traffic_collector.py -> Kaviya Priya.
traffic_collector.py -> Prabhakar
benign_traffic_generator.py -> Reshma
ddos_traffic_generator_1.py -> Sony Jenith
ddos_traffic_generator_2.py -> Sowmya

In order to obtain traffic data, the Flow Collector first contacts the controller. When the controller receives a request for traffic information, it employs To gather data from flow tables, 1use OpenFlow protocol. Every switch connected to the controller receives a flow-stats request from the controller, which asks the switch for flow statistics. As a result, each switch responds with a flow-stats reply message that includes all flow entries from all flow tables, each entry of which includes the flow description and any associated counters. The controller then answers this component after compiling the traffic data from all the switches. The Flow Collector parses the flow information after receiving it and discards any unnecessary data. The flow identifier (flow-id) and the flow counters are the pertinent data that are kept. The source IP address, destination IP address, source MAC address, destination MAC address, TCP/UDP source port, TCP/UDP destination port, and transport protocol identification make up the seven-tuple that is referred to as the flow-id (protocol). It's crucial to remember that this tuple can be modified to meet the requirements of the network infrastructure. Only three counters—the packet count, byte count, and duration—are included in the OpenFlow flow counters.

Detection-Module: (Worked by Sowmya, Sony Jenith)

controller.py -> Sony Jenith.
topology.py -> Sowmya.

Depending on the source of the data to be studied or the method used to identify anomalous occurrences, there are several categories that might be used to classify intrusion detection. Flow-based or packet-based detection is employed depending on the data to be processed, and depending on the detection method, signature-based or anomaly-based detection can be utilized. Flow-based detection was selected based on the source of the data to be evaluated since it would be more suited for high-speed networks and more effective than packet-based detection in terms of processing and memory overhead. Anomaly-based intrusion detection is implemented as a detection method, and more specifically, detection with ML.

Mitigation-Module: (Worked by Prabhakar, Sowmya, Sony Jenith, Reshma)

switch.py -> Prabhakar
switchm.py -> Reshma
mitigation_module.py -> Sony Jenith
mitigation_module.py (KNN) -> Sowmya

Once a malicious flow has been verified by the anomaly detection module, the Anomaly Mitigation module is in charge of implementing mitigation measures to prevent network disruption or performance degradation. In the framework, attack at its source is stopped. The preventing attacks caused by IP spoofing does not necessarily involve banning the attackers' IP addresses. For proof-of-concept purposes, in this case, attacker's Ethernet address is blocked.

RESULT AND DISCUSSION:

The proposed attack detection and mitigation framework was evaluated by testing various machine learning models for online traffic classification and DDoS attack detection.The Python-based prototype dynamically installed firewall rules to block Ethernet addresses of detected malicious flows. Results showed that once attacks were detected, all subsequent malicious traffic from those sources was successfully blocked without impacting normal traffic. Among the models tested, some algorithms demonstrated higher accuracy and better detection performance. Overall, the framework effectively identified and mitigated attacks in real time while preserving legitimate network activity.

CONCLUSION:

In this work, a random forest machine learning algorithm is used to develop a model that can automatically identify and mitigate DDoS assaults in SDN networks. All of the traffic flow entries are regularly collected by the model, which then extracts the native flow features and expands them by including additional features. A detection module uses five criteria to categorize each flow as normal or anomalous. When an attack is discovered, its source is prevented. Three ML algorithms were assessed with regard to the classification ML method utilized in the detection module, including random forest, SVM, and KNN. The outcomes of the experiment demonstrated that RF is the best classifier for the generated network. Without disrupting regular traffic, the implemented methodology proved effective at swiftly and correctly identifying and thwarting threats.

TOOLS AND INSTALLATION GUIDE:

- install [Virtual box](https://www.virtualbox.org/wiki/Downloads) or VM-ware workstation.

- install [Mininet-VM](https://github.com/mininet/mininet/releases/).

- install [Ubuntu](https://ubuntu.com/download/desktop) in virtual box.

- install[ryucontroller](https://github.com/faucetsdn/ryu) in ubuntu VM. Read this article (https://heltale.com/sdn/setting_up_ryu/) Before Setup RYU.

- install MOBAXTERM or XMING or PUTTY for mininet VM.

IN UBUNTU/RYU CONTROLLER:

```
#check your IP address

ifcongif

# it should be something 198.162.XX.XX copy it

# change working directory to controller folder

cd controller

# switch on the ryu-controller

ryu-manager controller.py
```

IN MININET-VM:

```
# change working directory to mininet folder
cd mininet
# change controller ip address that you copied from ryu controller ip
nano topology.py
# run topology
sudo python topology.py
```

HPING COMMANDS IN XTERM HOST:

```
# icmp flood
hping3 -1 -V -d 120 -w 64 -p 80 --rand-source –flood <target-IP>
# syn flood
hping3 -S -V -d 120 -w 64 -p 80 --rand-source –flood <target-IP>
# udp flood
hping3 -2 -V -d 120 -w 64 -p 80 --rand-source –flood <target-IP>
```

THE END