CHAPTER 5
CODING AND TESTING

This means the project was taken on around critical milestones that would ensure DDoS attacks on an IoT network are detected and mitigated. Advanced AI techniques, particularly Graph Neural Networks (GNNs), made this testbed: a real-world within an NS3 simulation environment that aimed to improve real-time DDoS detection and mitigation systems using GNN-based models. This chapter details the simulation setup data production, as well as training, integrating the trained AI model in the NS3 environment, which then provides real-time attack identification and mitigation.

5.1 Simulation Setup in NS3

The first critical step in the process of testing was modeling the IoT simulation environment in NS3, which indeed could basically mimic real IoT device behaviors and network attack patterns. This simulated benign and malicious traffic pattern was essential in the generation of traffic that would assist in the validation of the ability of the system to detect and mitigate DDoS attacks.

5.1.1 Network Topology Design

The network topology design was therefore essential to the success of the project. The network was a natural or typical example of an IoT environment where central routers mediate traffic between a set of IoT devices and a server.

- IoT Devices: The simulation included many IoT devices. Some of them include smart home appliances, like smart speakers, thermostats, and security cameras. In this way, such devices would send benign traffic to the central server as legitimate data packets. In this experiment, a total of 20 IoT devices were simulated to mimic close-to-reality cases.
- Bot Nodes. In the network, 5 bot nodes mimicked a DDoS attack through their malicious traffic. Bot nodes are compromised IoT devices taken over by a botnet. They overwhelm the server due to heavy data traffic.
- Router and Server: In the architecture, the router would be working as a central hub on which benign and malicious traffic flow. All the intensity of DDoS attack fell on the server, with this architecture especially filtering the traffic reaching the server, and that is also with the help of routers.

5.1.2 Traffic Generation

Realistic traffic patterns accurately represent the behavior of an IoT network.

- Benign Traffic: This was created using the feature of TCP BulkSend, simulating typical data uploads from IoT devices, such as sensor readings or status updates. The benign traffic sent was low volume and periodic. The NS3 pseudocode is as follows:

```
BEGIN Simulation
DEFINE experimental parameters: UDP_PORT = 9001
    TCP_PORT = 9000 TCP_RATE = "10Gbps"
    MAX_SIMULATION_TIME = 1200 seconds
    SEND_SIZE = 64 bytes NUM_TCP_CONNECTIONS = 10
INITIALIZE network components:
    Create nodes: iotDevices (1 device), router (1 device), server (1 device) DEFINE
Point-To-Point link attributes:
    Data Rate = TCP_RATE Channel Delay = 0.1 ms
INSTALL network devices:
    Connect iotDevices to router and router to server via Point-To-Point links ASSIGN
IP addresses:
    Set base IP address for network with specified subnet mask Assign IP addresses to
    all devices
CONFIGURE multiple TCP connections between IoT Device and Server: FOR each TCP
    connection (up to NUM_TCP_CONNECTIONS):
        SET bulk send application on IoT device to send unlimited traffic SET send
        size to SEND_SIZE for packet size
        INSTALL bulk send application on IoT device START bulk send application at
        time 0
        STOP bulk send application at MAX_SIMULATION_TIME END FOR
CREATE TCP sinks on Server: FOR each TCP connection:
        CONFIGURE TCP sink application on server to receive packets on specific
        TCP_PORT START TCP sink application at time 0
STOP TCP sink application at MAX_SIMULATION_TIME END FOR
POPULATE routing tables
CONFIGURE node mobility and positions for visualization: SET router, server, and
    iotDevices to fixed positions SET layout parameters for network visualization
CREATE animation output file "BenignTraffic.xml" and enable packet metadata RUN
Simulation
END Simulation
```

- Malicious Traffic: Bot nodes using On-Off traffic generators generated malicious traffic, which produced bursty, high-rate UDP traffic designed to flood the server. The traffic was like that shown in real-world DDoS attacks-traffic spikes, which are hard to predict and overwhelming of the server. The NS3 pseudocode is as follows:

```
BEGIN DDoS Simulation
DEFINE experimental parameters: UDP_SINK_PORT = 9001
    TCP_SINK_PORT = 9000 DDOS_RATE = "100Mbps"
    MAX_SIMULATION_TIME = 100 seconds NUMBER_OF_BOTS = 5
INITIALIZE network components:
    Create nodes: iotDevices (1 legitimate device), router (1 device), server (1
    device) Create botNodes for attack traffic with NUMBER_OF_BOTS devices
DEFINE Point-To-Point link attributes: Data Rate = DDOS_RATE
    Channel Delay = 1 ms INSTALL network devices:
    Connect iotDevices to router and router to server via Point-To-Point links Connect
    each botNode to router with individual Point-To-Point links
ASSIGN IP addresses:
    Set base IP address for network with specified subnet mask Assign IP addresses to
    each bot on a unique subnet
    Assign IP addresses to IoT device, router, and server CONFIGURE DDoS Application on
botNodes:
    FOR each botNode (up to NUMBER_OF_BOTS):
        SET OnOff application to UDP traffic with constant rate and on/off times
        INSTALL OnOff application on each botNode
START OnOff application at time 0
        STOP OnOff application at MAX_SIMULATION_TIME END FOR
CONFIGURE legitimate TCP traffic from IoT Device to Server:
    SET BulkSend application to TCP traffic with unlimited data size and packet size
1024 bytes
    INSTALL BulkSend application on IoT device START BulkSend application at time 0
    STOP BulkSend application at MAX_SIMULATION_TIME SET UDP Sink on Server:
    CONFIGURE UDP sink to receive packets on specified UDP_SINK_PORT START UDP Sink
    application at time 0
    STOP UDP Sink application at MAX_SIMULATION_TIME SET TCP Sink on Server:
    CONFIGURE TCP sink to receive packets on specified TCP_SINK_PORT START TCP Sink
    application at time 0
    STOP TCP Sink application at MAX_SIMULATION_TIME POPULATE routing tables
```

```
CONFIGURE node mobility and positions for visualization: SET router, server, and
    botNodes to fixed positions SET layout parameters for network visualization
CREATE animation output file "DDoSTraffic.xml" and enable packet metadata RUN
Simulation
END Simulation
```

5.1.3 Mobility Model

The Constant-Position-Mobility-Model was utilized to simulate static nodes. Such static IoT devices, like household appliances, as represented by the model, would remain in fixed positions when they were in action. Though static, the system dynamically monitored their traffic patterns with a constant network topology at any point in time.

5.1.4 Setting up NetAnim

The visualization tool NetAnim was used to allow vivid animation of network traffic and behaviors under normal operation as well as DDoS attack conditions.

- Node Placement: Many nodes were placed around the central router and server at distances where the network interactions are visible. The IoT devices and bot nodes are positioned across from each other.
- Packet Flows and Effect of the Attack: NetAnim was an important thing for visualizing the packet flows and how the nodes interact with one another, as well as the effect caused by the DDoS attacks on the network. Clearly, it differentiated between benign and malicious traffic flows and even extracted insight into the effects of various mitigations.

5.2 Dataset Generation

Based on this simulation environment, a dataset containing benign and malicious traffic patterns was generated next. The size of the dataset is what made it crucial in the training of the AI model on how to detect and classify DDoS attacks.

5.2.1 Traffic Data Collection

For the simulation, traffic was captured by logging every packet passed to the router. This included benign and malicious traffic. The data logged in an XML file meant that through that format, I was able to capture critical details such as timestamp, source IP, destination IP, and packet size.

5.2.2 Data Conversion into CSV Format

For converting the XML logs into a format suitable for training the AI model, custom scripts were used. It has been chosen primarily due to direct compatibility with most of the machine learning frameworks and due to the efficiency of maintaining required network features in an organized manner.

- Feature Extraction: Important features, such as packet size, source/destination IP addresses, and traffic patterns, were extracted from the data for the training purpose.
- Labelling: Each packet of data was labeled as benign or DDoS according to whether it originated from a legitimate IoT device or bot node.
- XML to CSV conversion pseudocode for Benign Traffic is as follows:

```
IMPORT XML parsing and CSV writing libraries IMPORT regex library for IP and size
extraction
DEFINE FUNCTION parse_benign_xml_to_csv WITH parameters:
    - xml_file: XML input file containing packet data
    - csv_file: output CSV file
    - benign_ip_prefix: prefix indicating benign traffic sources BEGIN FUNCTION
parse_benign_xml_to_csv
    LOAD XML data from xml_file GET root of the XML document OPEN csv_file in write
    mode
    INITIALIZE CSV writer and write header row: ["Timestamp", "Source",
"Destination", "PacketSize", "Label"]
    SET packet_count = 0 // Total packets processed SET benign_count = 0 // Count of
    benign packets FOR each packet element in XML root:
        GET 'meta-info' attribute of packet IF 'meta-info' attribute exists THEN:
            EXTRACT timestamp from 'fbTx' attribute OR default to '0' if not present
            EXTRACT source and destination IP addresses using regex on 'meta-info'
            IF IP addresses are found THEN:
                EXTRACT packet size using regex on 'meta-info' OR default to '0' if
                not
present
                CHECK if source IP starts with benign_ip_prefix: IF TRUE, SET label
                    to "Benign"
                    WRITE row to CSV with [timestamp, source_ip, destination_ip,
packet_size, label]
                    INCREMENT benign_count
                END IF

            INCREMENT packet_count END IF
    DISPLAY total packets processed and total benign packets END FUNCTION
```

- XML to CSV conversion pseudocode for DDoS Traffic is as follows:

```
IMPORT XML parsing and CSV writing libraries IMPORT regex library for IP and size
extraction
DEFINE FUNCTION parse_ddos_xml_to_csv WITH parameters:
    - xml_file: XML input file containing packet data
    - csv_file: output CSV file
    - ddos_ips: list of IP prefixes identifying DDoS traffic sources BEGIN FUNCTION
parse_ddos_xml_to_csv
    LOAD XML data from xml_file GET root of the XML document OPEN csv_file in write
    mode
    INITIALIZE CSV writer and write header row: ["Timestamp", "Source",
"Destination", "PacketSize", "Label"]
    FOR each packet element in XML root: GET 'meta-info' attribute of packet
        IF 'meta-info' attribute exists THEN:
            EXTRACT timestamp from 'fbTx' attribute OR default to '0' if not present
            EXTRACT source and destination IP addresses using regex on 'meta-info'
            IF IP addresses are found THEN:
                EXTRACT packet size using regex on 'meta-info' OR default to '0' if
                not
present
                CHECK if source IP matches any prefix in ddos_ips: IF TRUE, SET label
                    to "DDoS"
                    ELSE, SET label to "Unknown"
                WRITE row to CSV with [timestamp, source_ip, destination_ip,
packet_size, label]
            END IF END IF
    DISPLAY message indicating successful DDoS traffic export to CSV END FUNCTION
```

- XML to CSV conversion pseudocode for Validation generation setup is as follows:

```
IMPORT XML parsing and CSV writing libraries IMPORT regex library for IP and size
extraction DEFINE FUNCTION parse_xml_to_csv WITH parameters:
    - xml_file: XML input file containing packet data
    - csv_file: output CSV file
    - ddos_ips: list of IP prefixes identifying DDoS traffic sources
    - benign_ips: list of IP prefixes identifying benign traffic sources BEGIN
FUNCTION parse_xml_to_csv
    LOAD XML data from xml_file GET root of the XML document
    INITIALIZE ddos_count and benign_count to 0 OPEN csv_file in write mode
    INITIALIZE CSV writer and write header row: ["Timestamp", "Source",
"Destination", "PacketSize", "Label"]
    FOR each packet element in XML root: GET 'meta-info' attribute of packet
        IF 'meta-info' attribute exists THEN:
            EXTRACT timestamp from 'fbTx' attribute OR default to '0' if not present
            EXTRACT source and destination IP addresses using regex on 'meta-info'
            IF IP addresses are found THEN:
                EXTRACT packet size using regex on 'meta-info' OR default to '0' if
                not
present
                DETERMINE traffic label based on source IP:
                    IF source IP matches any prefix in ddos_ips: SET label to "DDoS"
                        INCREMENT ddos_count
                    ELSE IF source IP matches any prefix in benign_ips: SET label to
                        "Benign"
                        INCREMENT benign_count ELSE:
                        SKIP packet (no matching IP ranges)
                WRITE row to CSV with [timestamp, source_ip, destination_ip,
packet_size, label]
            END IF END IF
    DISPLAY total DDoS packets, total benign packets, and total packets processed
END FUNCTION
```

- XML to CSV main driver conversion pseudocode for all the three is as follows:

```
DEFINE FUNCTION main
    SET xml_file to path of the XML file generated by NS-3 SET csv_file to output
    path for the CSV file
    DEFINE ddos_ips as a list of IP prefixes associated with DDoS attack sources
    DEFINE benign_ips as a list of IP prefixes associated with benign sources
    DISPLAY message indicating the start of XML processing
    CALL parse_xml_to_csv FUNCTION with arguments:
        - xml_file
        - csv_file
```

```
            - ddos_ips
            - benign_ips
        DISPLAY message confirming CSV file has been saved after conversion END FUNCTION
    IF script is run directly THEN CALL main FUNCTION
    END IF
```

5.2.3 Data Balancing

By balancing the dataset provided, which would have otherwise been biased due to an AI model's favor toward one class of data, the modeling aspect was taken care of to prevent biased weightings toward one class of data on the part of the AI model. This ensured that the model had equal numbers of both benign and malicious packets, so it may learn to correctly classify either type of traffic. The pseudocode to balance the dataset is as follows:

```
IMPORT pandas library for data manipulation DEFINE FUNCTION combine_csv_files WITH
parameters:
    - benign_csv: path to the benign traffic CSV file
    - ddos_csv: path to the DDoS traffic CSV file
    - output_csv: path for the combined output CSV file LOAD benign traffic CSV into
    benign_df
    LOAD DDoS traffic CSV into ddos_df
    DETERMINE minimum number of rows between benign_df and ddos_df
    SAMPLE both benign_df and ddos_df to contain min_rows rows each, with a fixed random
state for consistency
    CONCATENATE benign_df and ddos_df into a single DataFrame combined_df SHUFFLE
    combined_df to mix benign and DDoS rows, using a fixed random state SAVE combined_df
    to output_csv without row indices
    DISPLAY message confirming combined CSV file has been saved END FUNCTION
DEFINE FUNCTION main
    SET benign_csv to path of benign traffic CSV file SET ddos_csv to path of DDoS traffic
    CSV file
    SET output_csv to output path for combined CSV file
    CALL combine_csv_files FUNCTION with benign_csv, ddos_csv, and output_csv END FUNCTION
IF script is run directly THEN CALL main FUNCTION
END IF
```

5.3 AI Model Training

The dataset was then prepared, and training the GNN model on detecting DDoS attacks was the job done.

5.3.1 Model Architecture

The GNN model was designed to study the node interactions in the network. It was trained with the classification of traffic as either benign or malicious.

- Input Layer: This was a feed made up of the raw features from the CSV dataset.
- Hidden Layers: These layers took the data streams in and looked for patterns that distinguished benign from DDoS traffic.
- Output Layer: This output layer spewed out the entire classification, whether it was benign or malicious.
- GNN identification pseudocode is as follows:

```
IMPORT necessary libraries for data processing, neural networks, and visualization
DEFINE GCN MODEL CLASS:
    - INIT function:
        CREATE two GCNConv layers:
            1. First layer with input channels 1 and output channels 16
            2. Second layer with input channels 16 and output channels 2 (for two
classes: Benign, DDoS)
    - FORWARD function:
        APPLY first convolution layer on data with edge index APPLY ReLU activation
        APPLY second convolution layer RETURN output
DEFINE FUNCTION load_data WITH parameter:
    - csv_file: path to the CSV dataset LOAD CSV data into a pandas DataFrame
    EXTRACT PacketSize as node features and Label as target labels CONVERT
    PacketSize to tensor of floating-point values
    CONVERT Label to tensor of integer values (1 for DDoS, 0 for Benign) CREATE
    edge_index tensor with self-loops for each node
    RETURN Data object containing node features, edge index, and labels DEFINE
FUNCTION train_gnn_model WITH parameters:
    - data: dataset for training
    - model: GCN model instance
    - epochs: number of training epochs (default 150) INITIALIZE optimizer with
    model parameters and learning rate INITIALIZE loss criterion as cross-entropy
    loss
    SET model to training mode
    CREATE lists to store training loss and accuracy for each epoch FOR each epoch:
        ZERO gradients
```

```
            PERFORM forward pass to get model output CALCULATE loss and backpropagate
            UPDATE model parameters with optimizer
            COMPUTE accuracy by comparing predictions to actual labels STORE loss and
            accuracy for visualization
            DISPLAY epoch number, loss, and accuracy
        RETURN trained model, list of losses, and list of accuracies DEFINE FUNCTION
    validate_gnn_model WITH parameters:
        - model: trained GCN model
        - validation_data: dataset for validation SET model to evaluation mode
        PERFORM forward pass to get model output
        COMPUTE accuracy on validation set DISPLAY validation accuracy
        GENERATE classification report and confusion matrix DISPLAY confusion matrix as
        heatmap
    DEFINE FUNCTION plot_training_metrics WITH parameters:
        - losses: list of loss values
        - accuracies: list of accuracy values PLOT training loss over epochs
        PLOT training accuracy over epochs DEFINE FUNCTION main
        SET paths to training and validation datasets LOAD training and validation data
        INITIALIZE GCN model
        CALL train_gnn_model with training data, model, and specified epochs SAVE
        trained model to specified path
        PLOT training metrics
        CALL validate_gnn_model with trained model and validation data IF script is run
    directly THEN
        CALL main FUNCTION END IF
```

5.3.2 Training Process

This model was trained using this labeled dataset by minimizing errors and increasing accuracy.

- Loss Function: Utilize the loss function to track how much each class predicted was different from the actual class in the data set. With each iteration, it updated the model's parameters so that its errors were minimized.
- Hyperparameter Tunwas In relation to hyperparameters, the learning rate, hidden layers, and the size of the batch were all properly tuned for optimal model performance via iterative training.

5.3.3 Validation

After every training epoch, the model was validated on a separate dataset to ensure that it generalized very well to new, unseen data. Techniques for preventing overfitting were also put in place to ensure that the model did not perform well only on training data but could handle real-world scenarios.

5.4 Mitigation in NS3 – Real-time detection and Action

The final stage involved implementing the trained GNN model in the NS3 simulation environment to facilitate real-time detection and mitigation of DDoS attacks.

5.4.1 Detection of Malicious IP Address

The trained GNN was implemented within the NS3 environment through processing real-time network traffic. Each packet's features were analyzed by the algorithm, which asserted the malicious IP addresses involved in the DDoS attack and detected them in real time.

5.4.2 Packet Filtering

Once such malicious IP address identities were known, they were placed in a blocklist. A router level packet filtering mechanism prevented the packets from reaching the server by dropping those packets coming from identified malicious IP addresses, reducing the spread of this attack. Pseudocode for Packet Filtering is as follows:

```
IMPORT necessary NS-3 modules DEFINE CONSTANTS:
    - UDP_SINK_PORT, TCP_SINK_PORT
    - DDOS_RATE for attack traffic, TCP_RATE for legitimate traffic
    - MAX_SIMULATION_TIME
    - NUMBER_OF_BOTS, NUMBER_OF_IOT_DEVICES
DEFINE FUNCTION GetDeviceFromIp to retrieve device from IP address for packet dropping:
    FOR each node in nodes:
        CHECK if node has interface with given IP
        IF interface exists, RETURN the device associated with the IP RETURN nullptr if
    no match found
BEGIN main FUNCTION
    PARSE command-line arguments SET simulation time resolution
    ENABLE logging for UDP echo applications
    CREATE nodes for IoT devices, router, server, and bot nodes CONFIGURE Point-To-
    Point links:
        SET data rate and delay attributes
```

```
      INSTALL Point-To-Point connections:
           - BETWEEN router and server
           - BETWEEN each bot and router
           - BETWEEN each IoT device and router INSTALL Internet stack on all nodes
        ASSIGN IP addresses to bot nodes, server, and each IoT device with unique subnets
        DEFINE list of malicious IPs (detected by GNN model)
        FOR each malicious IP:
            FIND device associated with IP
            IF device found, SET device callback to drop packets CONFIGURE DDoS attack
        behavior:
            INITIALIZE OnOff application for each bot node with UDP traffic SET traffic
            rate, on-time, and off-time attributes
            INSTALL OnOff applications on bot nodes and set start/stop times CONFIGURE
        legitimate traffic from IoT devices:
            FOR each IoT device:
                INITIALIZE BulkSend application with TCP traffic to server SET unlimited
                packet transmission with specified packet size
                INSTALL BulkSend application on IoT device and set start/stop times SET UDP
        Sink application on server to receive UDP packets
        SET TCP Sink application on server to receive TCP packets POPULATE routing tables
        CONFIGURE node positions for visualization:
            - SET positions for router, server, bot nodes, and IoT devices with
              specified
spacing
        CREATE AnimationInterface object for XML output ENABLE packet metadata in the
        animation file SET custom positions for nodes
        RUN and DESTROY the simulation
    END main FUNCTION
```

5.5 Conclusion of Testing and Implementation

The deployment and testing of this system clearly showed how an AI-driven DDoS mitigation system can detect and mitigate attacks in real-time conditions. Comparing packet delivery ratio, latency, and server loads before and after the attacks being mitigated, the AI-driven system proved to maintain network performance even while under a DDoS attack.