BlackoutADR: An Adversarial Attack on Adaptive Data Rate in LoRaWAN for FANETs

We target the Adaptive Data Rate (ADR) feature of LoRaWAN protocol used in IoT that was later deployed in drones. Our adversarial attack on a FANET could be an effective way to fool the IDS and cause a power drain on the drones, leading to a DDoS attack if the attack persists. A detailed breakdown of the protocol and the approach we are going with :

LoRaWAN :

- LoRaWAN (Long Range Wide Area Network) is a communication protocol specifically designed for long-range, low-power IoT (Internet of Things) networks. It operates on top of the LoRa (Long Range) modulation technique, which is known for its ability to transmit data over large distances with minimal power consumption.

Ref : https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9566707

F. Mason, M. Capuzzo, D. Magrin, F. Chiariotti, A. Zanella and M. Zorzi, "Remote Tracking of UAV Swarms via 3D Mobility Models and LoRaWAN Communications," in IEEE Transactions on Wireless Communications, vol. 21, no. 5, pp. 2953-2968, May 2022, doi: 10.1109/TWC.2021.3117142.

Key Features of LoRaWAN:

LoRaWAN can cover distances up to 15-20 kilometers in rural areas and several kilometers in urban environments, making it most used for applications that require wide-area coverage.

Understanding Adaptive Data Rate (ADR) in LoRaWAN

Adaptive Data Rate (ADR) is a mechanism in LoRaWAN that optimizes data rates, transmission power, and communication efficiency based on the network conditions. ADR allows devices to dynamically adjust their data rate and transmission power to maintain reliable communication while minimizing energy consumption.

ref: https://ieeexplore.ieee.org/document/9656523

A. Ilarizky, A. Kurniawan, E. P. Subagyo, R. Harwahyu and R. F. Sari, "Performance Analysis of Adaptive Data Rate Scheme at Network-Server Side in LoRaWAN Network," 2021 2nd International

Conference on ICT for Rural Development (IC-ICTRuDev), Jogjakarta, Indonesia, 2021, pp. 1-5, doi: 10.1109/IC-ICTRuDev50538.2021.9656523.

ref : https://ieeexplore.ieee.org/document/10400320

S. Chen, H. Zhao, Z. Zhang, Y. Gong, R. Li and L. Wang, "Improved ADR and Initial SF Allocation in LoRaWAN Network and their Simulation on NS3," 2023 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC), ZHENGZHOU, China, 2023, pp. 1-5, doi: 10.1109/ICSPCC59353.2023.10400320.

How ADR Works:

- Data Rate Adjustment: If a device has a strong link (close to a gateway), it can increase its data rate (spreading factor, SF) and reduce transmission power, which saves energy.

- Power Adjustment: Conversely, if the link is weak, the device lowers its data rate and increases transmission power to ensure the message is delivered reliably.

Targeting ADR to Cause Power Drain

In a FANET, where drones are typically constrained by battery life, manipulating ADR to cause the drones to transmit at higher power levels can lead to rapid battery depletion. How we can target ADR:

1. Manipulating Signal Strength (RSSI/SNR):

ADR relies on Received Signal Strength Indicator (RSSI) and Signal-to-Noise Ratio (SNR) to adjust transmission parameters. By artificially lowering these values, we can trick the drones into believing they are in a poor signal environment, unnecessarily causing them to increase their transmission power.

- Implementation:

- In ns-3, we can simulate environmental noise or interference that affects RSSI/SNR values, causing the ADR algorithm to respond by increasing the transmission power.

2. Faking Network Conditions

- Concept: Another way to fool ADR is by faking network conditions that trigger the drones to lower their data rate and increase power usage.

- Implementation:

- We inject packets that appear to come from distant gateways or modify the payloads to simulate a need for increased power. For instance, we can simulate a larger network range , making the drones believe they are further away from the gateway than they really are.

- Using Kali Linux to create scripts that send crafted LoRaWAN messages with altered ADR parameters, instructing the drone to use a higher power setting.

In LoRaWAN, if a drone doesn't receive an acknowledgment (ACK) in time, it may retry the transmission at a higher power level, assuming the message wasn't delivered due to poor signal conditions.

- Using a tool like `Scapy` lib or custom ns-3 scripts to intercept and delay ACK packets. This forces the drones to retransmit their data at a higher power, draining the battery faster.

Fooling the IDS Model Deployed on FANET in NS-3:

To fool the IDS while performing the above ADR-targeted attacks:

- We ensure that the manipulated packets or interference patterns mimic legitimate fluctuations in network conditions. The IDS should perceive these as normal adjustments by the ADR mechanism.

- NS-3 Implementation:

- Using variations in timing and signal strength that are within expected ranges for normal operations but still effective enough to trigger the desired power increase, We Implement small, random delays and packet loss in the network traffic to simulate natural disruptions, ensuring the IDS does not flag the behavior as anomalous.

- Simulation : we Implement these attacks in our ns-3 simulation, testing how the drones respond to the manipulated ADR and how the IDS reacts.

- Evaluate Impact: Measure the power consumption of the drones and the effectiveness of the IDS under these conditions. Adjust the attack vectors based on the IDS's response to ensure that it does not detect the manipulation.

Architecture and Setup:

The current simulation architecture consists of a FANET network with 20 nodes, configured using ns-3, with Node 0 designated as the leader node (backbone). This leader node is equipped with Snort for IDS capabilities. Here is a breakdown of the components and configurations:

1. ***Node Configuration:***

- Leader Node (Node 0):

- Purpose: Acts as the central node for the FANET, responsible for receiving all communication from other nodes.

- Implementation: Node 0 is configured with a `UdpEchoServer` to simulate receiving packets from other nodes. This node is also equipped with Snort for packet analysis.

- Adaptive Data Rate (ADR): The leader node uses an ADR from LoraWAN mechanism to dynamically adjust transmission power and data rate based on simulated RSSI values. These adjustments are made using `WifiRemoteStationManager` and are scheduled every second using `Simulator::Schedule`.

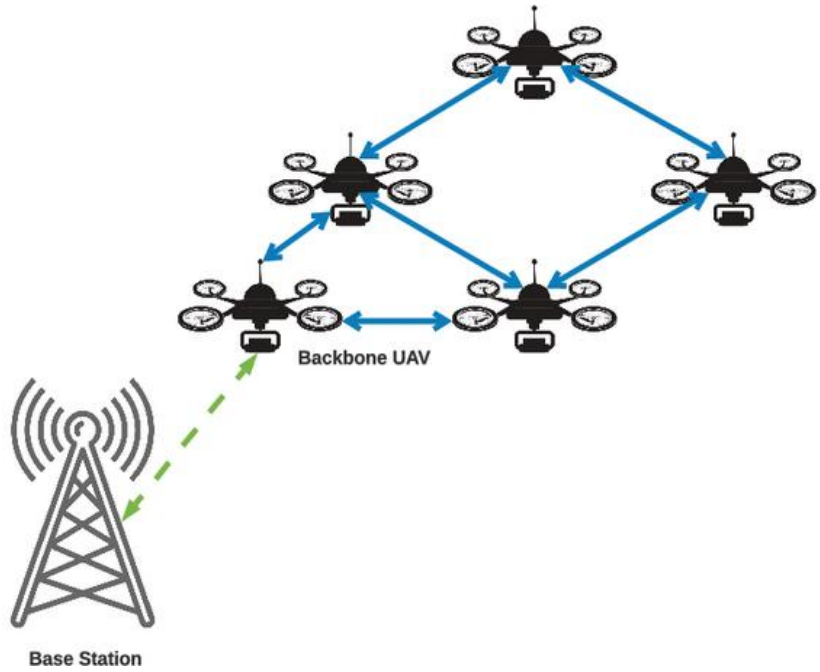
```
    NS_LOG_UNCOND("Leader Node 0 sniffed a packet of size: " << packet->GetSize() << " bytes");
}

// Install IDS on the leader node
void InstallIds(NodeContainer nodes) {
    Ptr<Node> leaderNode = nodes.Get(0); // Leader Node 0
    leaderNode->GetDevice(0)->TraceConnectWithoutContext("PhyRxEnd", MakeCallback(&PacketSniffer));
}

int main (int argc, char *argv[]) {
    CommandLine cmd;
    cmd.Parse(argc, argv);

    NodeContainer nodes;
    nodes.Create(20);
```

- Other Nodes:

- Purpose: Standard FANET nodes that communicate directly with the leader node.

- Implementation: These nodes are configured with `UdpEchoClient` applications to send periodic packets to Node 0. Each node adjusts its data rate and power settings based on RSSI values, simulating real-time adjustments in response to network conditions.

tested topology diagram from the literture and indusrty :

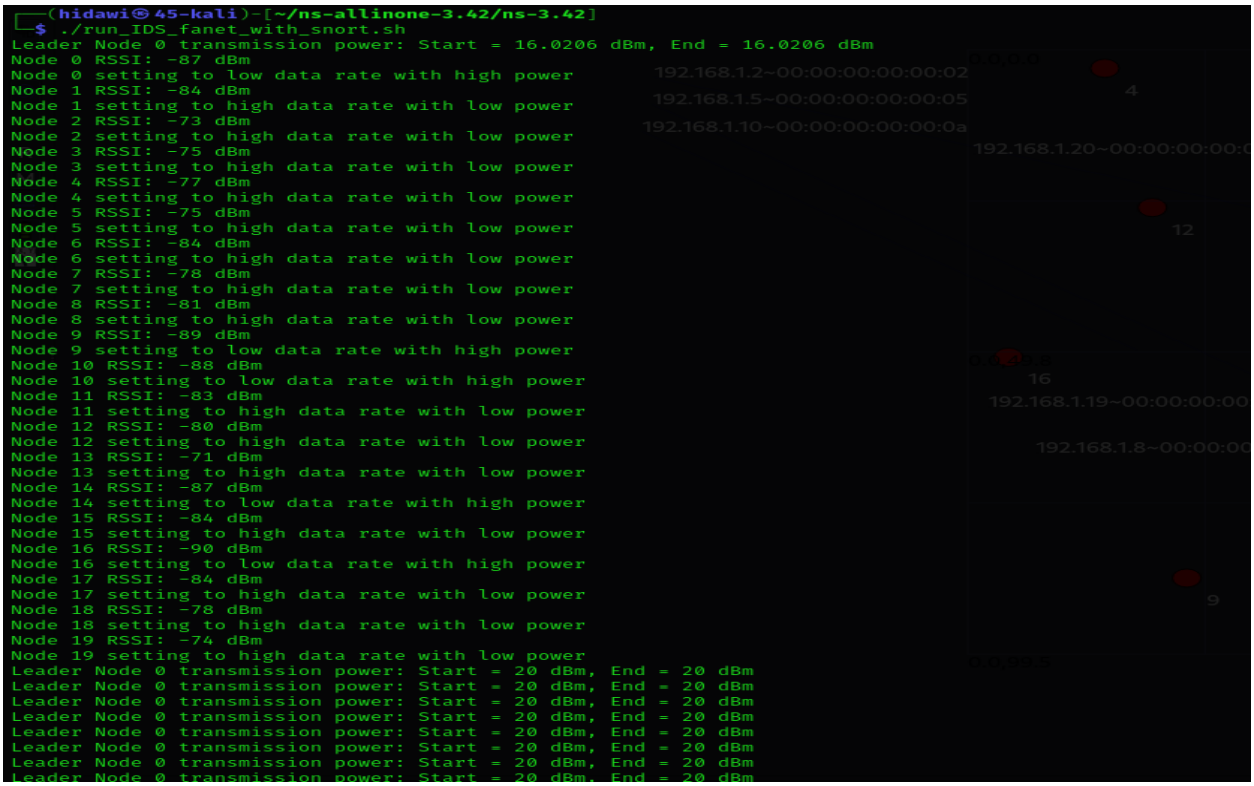


with the Backbone UAV (leader) in green (node 0) in the NetAnim simulation of our architecture:

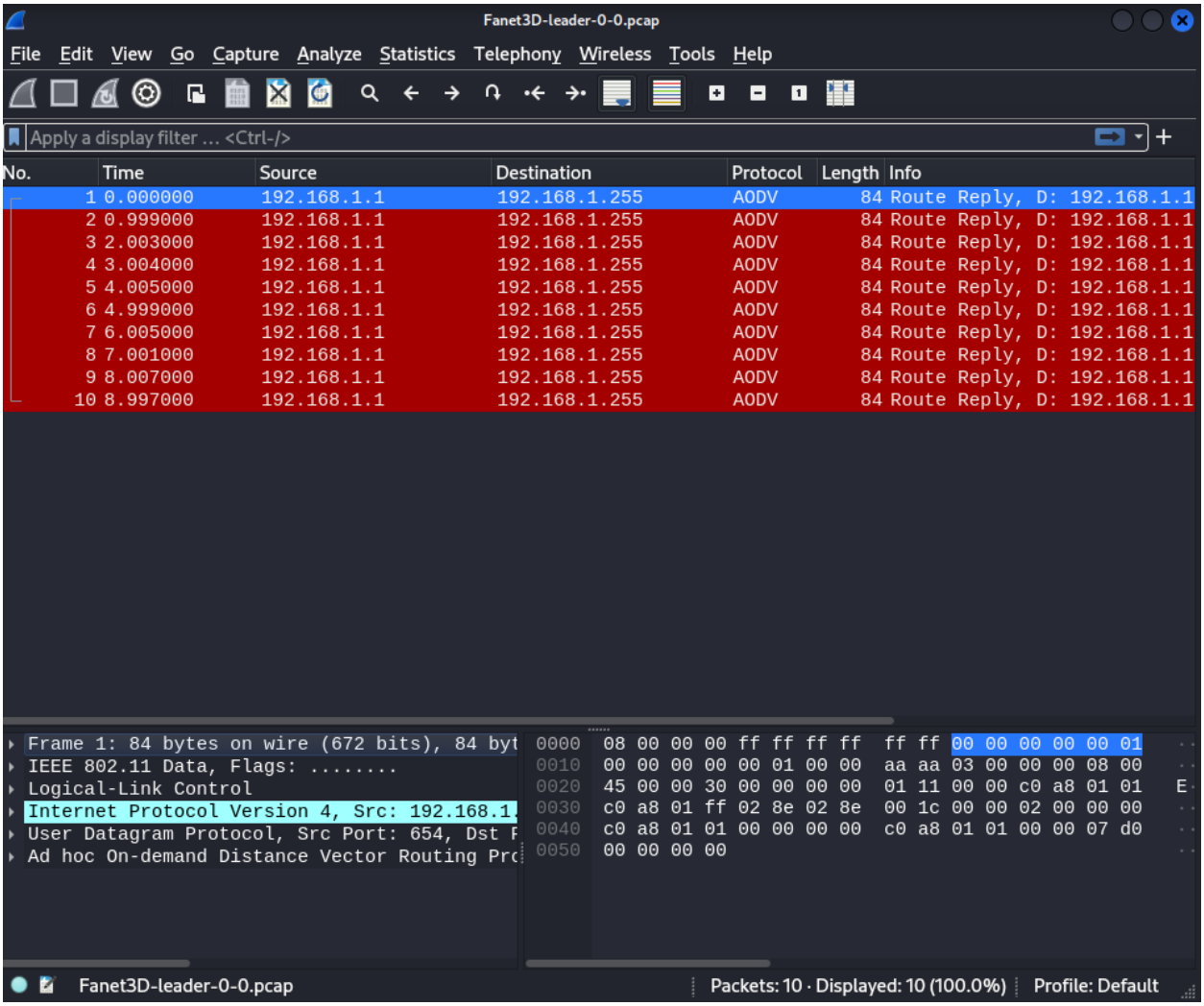
2. Snort IDS Setup:

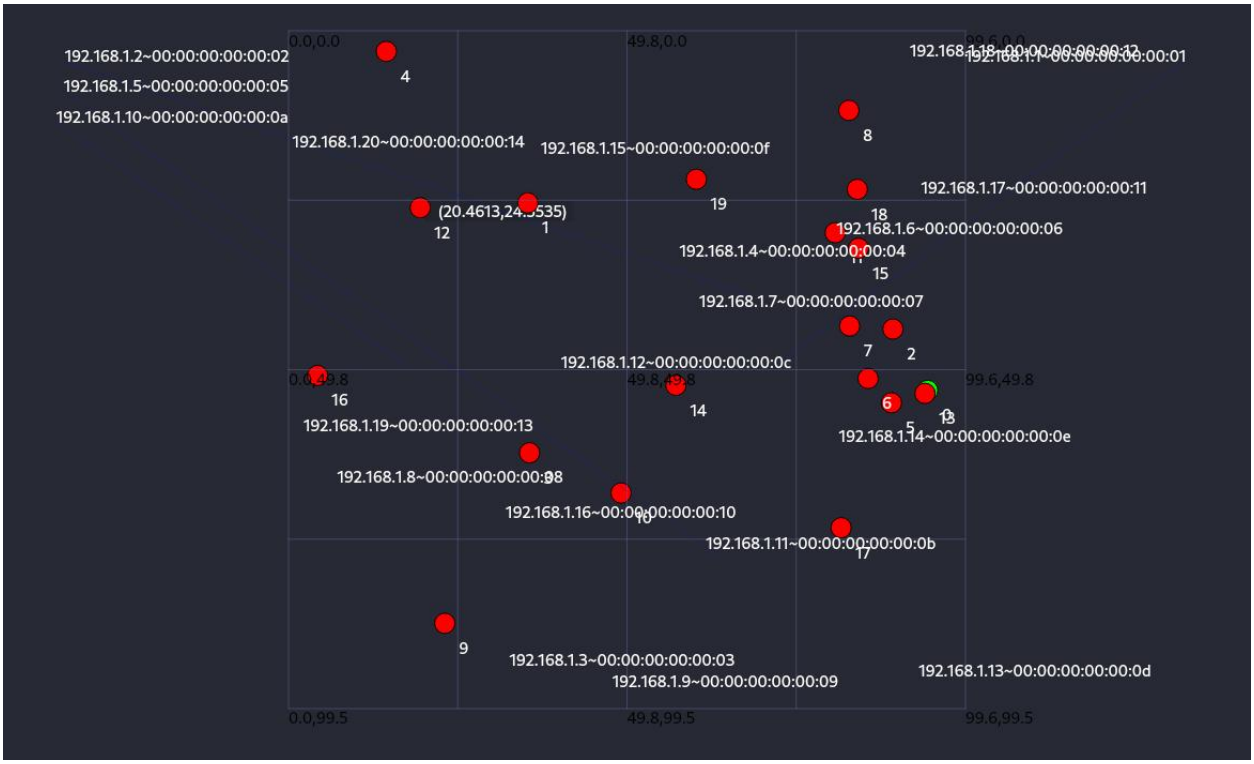
- Configuration: Snort is deployed on the leader node to monitor incoming packets. The Snort setup is automated using a `.sh` script (`run_IDS_fanet_with_snort.sh`). This script initiates the ns-3 simulation and launches Snort with customized configurations. Snort captures all incoming packets and saves them to a PCAP file, which is later analyzed.

- Packet Processing: Snort is configured to analyze packets in real-time, detecting potential anomalies. The `snort.lua` file defines the rules and parameters for packet inspection. In our initial setup, Snort is configured to recognize standard network attacks; however, we have expanded this with custom rules specific to FANET traffic patterns.



the PCAP file of the packets communicated between the UAVs in the FANET network:





code explain the sniffer IDS deployed on the leader node 0:

```
    // Simulate RSSI and adjust ADR based on it
    double simulatedRssi = -90 + (std::rand() % 20);
    if (simulatedRssi < -85) {
        stationManager->SetAttribute("DataMode", StringValue("DsssRate1Mbps"));
        phy->SetTxPowerStart(20.0);
        phy->SetTxPowerEnd(20.0);
        NS_LOG_UNCOND("Node " << node->GetId() << " setting to low data rate with high power");
    } else {
        stationManager->SetAttribute("DataMode", StringValue("DsssRate11Mbps"));
        phy->SetTxPowerStart(10.0);
        phy->SetTxPowerEnd(10.0);
        NS_LOG_UNCOND("Node " << node->GetId() << " setting to high data rate with low power");
    }

    // Log data to CSV
    LogData(node, phy, simulatedRssi);

    // Schedule next logging and ADR adjustment
    Simulator::Schedule(Seconds(1.0), &PeriodicLogAndAdr, node, dev);
}

// IDS Packet Sniffer for packet count
void PacketSniffer(Ptr<const Packet> packet) {
    packetCount++;
    NS_LOG_UNCOND("Leader Node 0 sniffed a packet of size: " << packet->GetSize() << " bytes");
}

// Install IDS on the leader node
void InstallIds(NodeContainer nodes) {
    Ptr<Node> leaderNode = nodes.Get(0); // Leader Node 0
    leaderNode->GetDevice(0)->TraceConnectWithoutContext("PhyRxEnd", MakeCallback(&PacketSniffer));
}

int main (int argc, char *argv[]) {
    CommandLine cmd;
```

the output of the automated snort IDS deployed on the node 0 (backbone based on the PCAP file :

In our simulation setup of a FANET using LoRaWAN within ns-3, a vulnerability assessment was conducted like the following. This section outlines a systematic approach to identifying potential attack vectors and security gaps using ns-3 simulation tools and Kali Linux.

1. **Simulating Network Traffic Capture**

Given the simulation of FANET over LoRaWAN in ns-3, traffic can be captured through ns-3's `PcapHelper` or ASCII tracing for post-analysis.

- Enabling Packet Capture in ns-3: The `PcapHelper` can be configured to log transmissions at each node.

- Analyzing Packets in Kali Linux: Once the capture is exported as `.pcap`, load it into Wireshark to detect patterns and gain insights into FANET communication details.

We used this command:

```bash
wireshark simulated_network.pcap
```

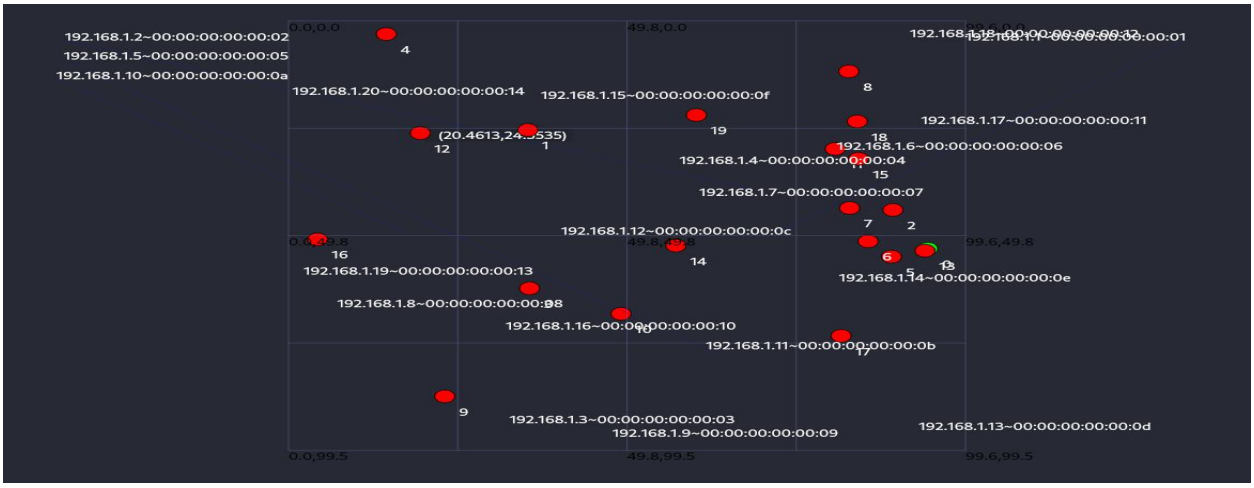
In Wireshark, we observed:

- Device Addressing: Identify device addresses (similar to LoRaWAN's `DevAddr`).

- Transmission Frequency: Assessed periodic packet intervals to infer key nodes, such as relay or leader nodes, in FANET's hierarchical setup.

2. **Analyzing Packet Structure and Protocol Behavior**

- Detecting Protocol Weaknesses: we also Analyzed MAC and network layer headers within captured packets for any vulnerabilities.



- Adaptive Data Rate (ADR) Analysis: Observed ADR-related fields, such as data rate and transmission power adjustments, and checked for predictability that could be exploited.

3. Simulating Adversarial ADR Attacks with `attacker.py`

The `attacker.py` script simulates ADR manipulation by introducing packet delays and varying payload sizes to degrade signal integrity, thereby targeting the FANET's adaptive mechanisms.

Example:

```python
from scapy.all import *
import random, time

# Target IP and Port of Leader Node
leader_ip = "192.168.1.1"
udp_port = 9

def spoof_fanet_traffic(target_ip, port, duration=60):
    end_time = time.time() + duration
    while time.time() < end_time:
        # Randomly vary signal quality by alternating packet characteristics
        packet_size = random.choice([50, 150, 300, 512, 1024])
        delay = random.uniform(0.5, 2.0)
        payload = random._urandom(packet_size)
        pkt = IP(dst=target_ip) / UDP(dport=port) / payload
        send(pkt, verbose=0)
        time.sleep(delay)
        print(f"Sent {packet_size}-byte packet with delay {delay:.2f}s")

if __name__ == "__main__":
    spoof_fanet_traffic(leader_ip, udp_port)
```

This code alternates between normal and degraded signal environments, creating irregular packet patterns that impact the ADR mechanism by influencing data rate changes, potentially leading to degraded FANET performance.

We also used as first hand this tools :

4. Vulnerability Assessment using Kali Linux Tools

Traditional Kali Linux tools support the vulnerability assessment of a simulated FANET:

- Network Mapping with Nmap: Scans for active IPs within the FANET simulation to establish node addresses and roles.

```bash
nmap -sP 192.168.1.0/24
```

- Packet Fuzzing with Scapy: Send unexpected or malformed packets to test the resilience of FANET nodes against abnormal inputs.

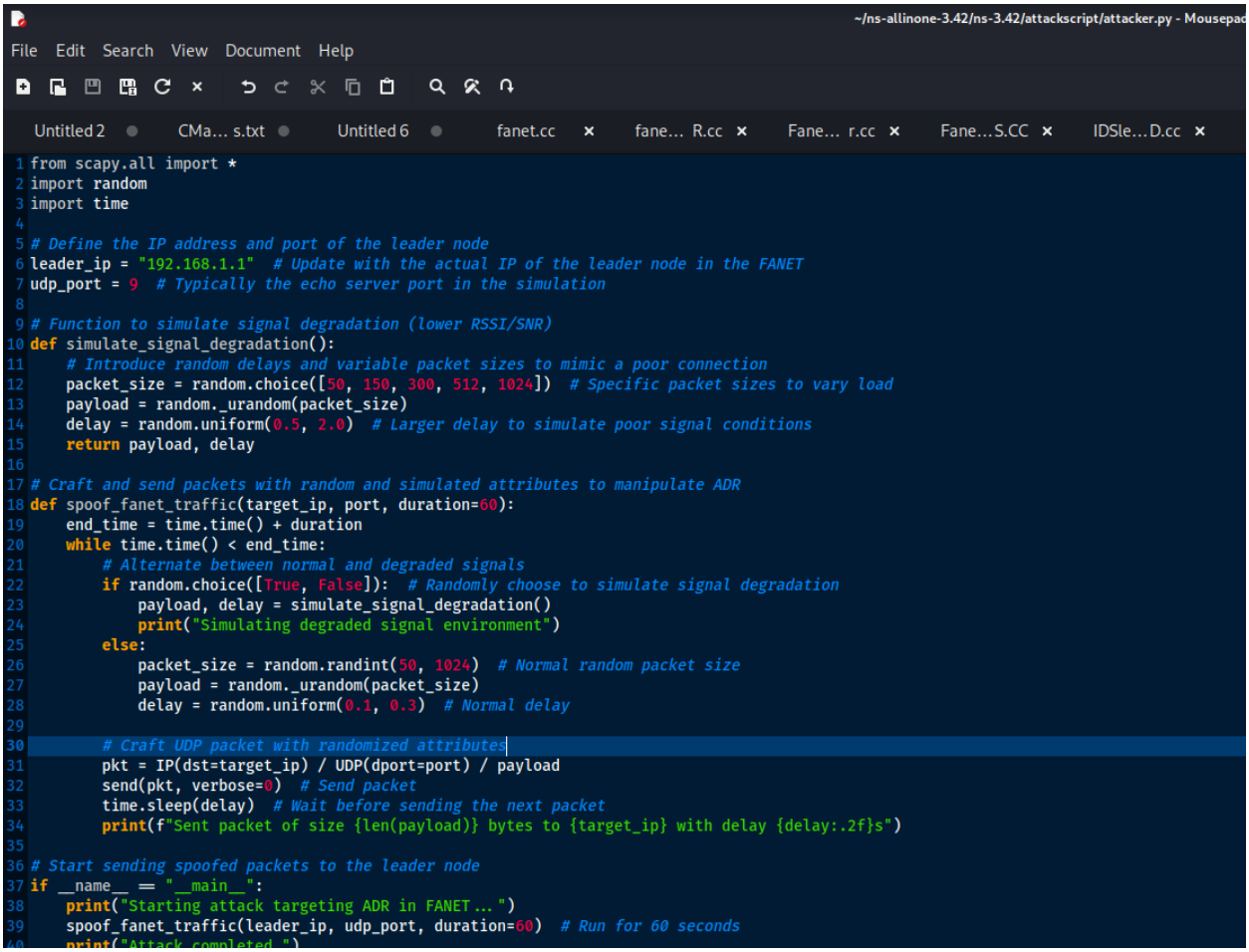
```python
from scapy.all import *
target_ip = "192.168.1.1"
for _ in range(100):
    packet = IP(dst=target_ip) / UDP(dport=9) / Raw(load="FuzzTest")
    send(packet, verbose=0)
```

---

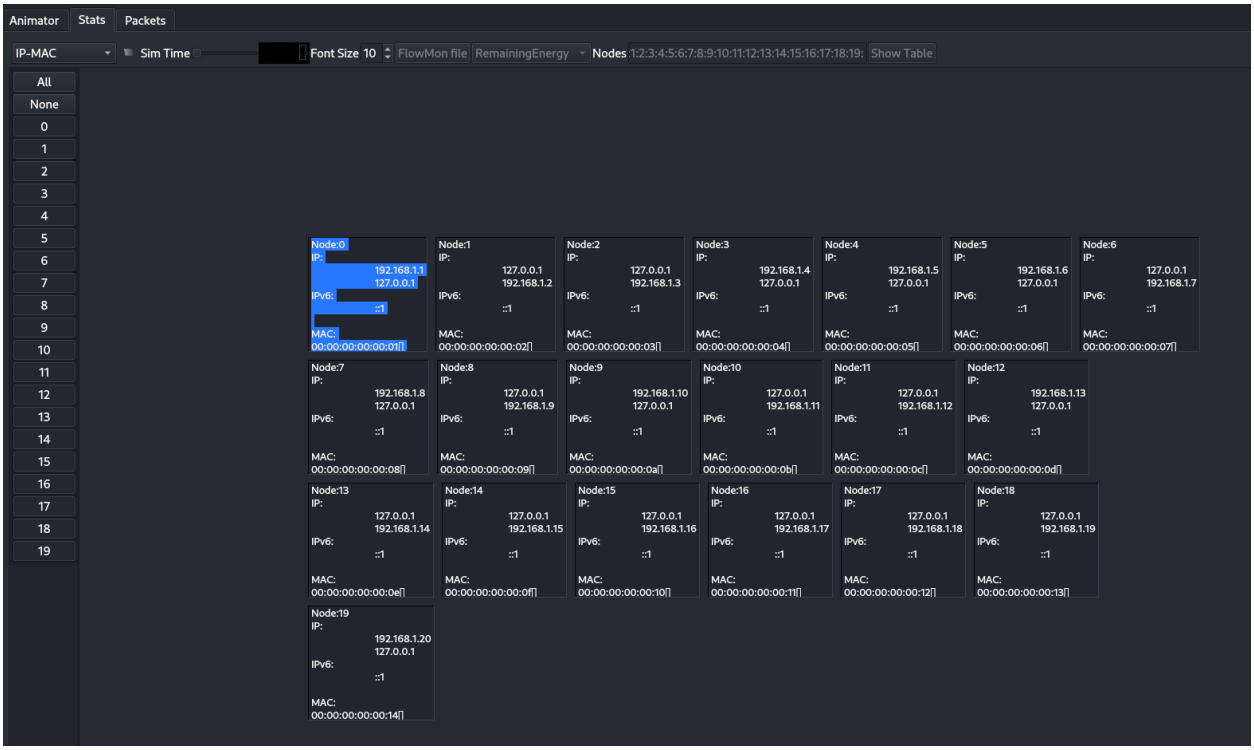
3. Attack Implementation:

- Attack Strategy: The attack targets the ADR mechanism by simulating a power drain attack. By sending frequent packets with varied sizes and delays, the goal is to trick the leader node into consistently increasing its transmission power, which could lead to faster battery depletion.

- Attacker Script (`attacker.py`):

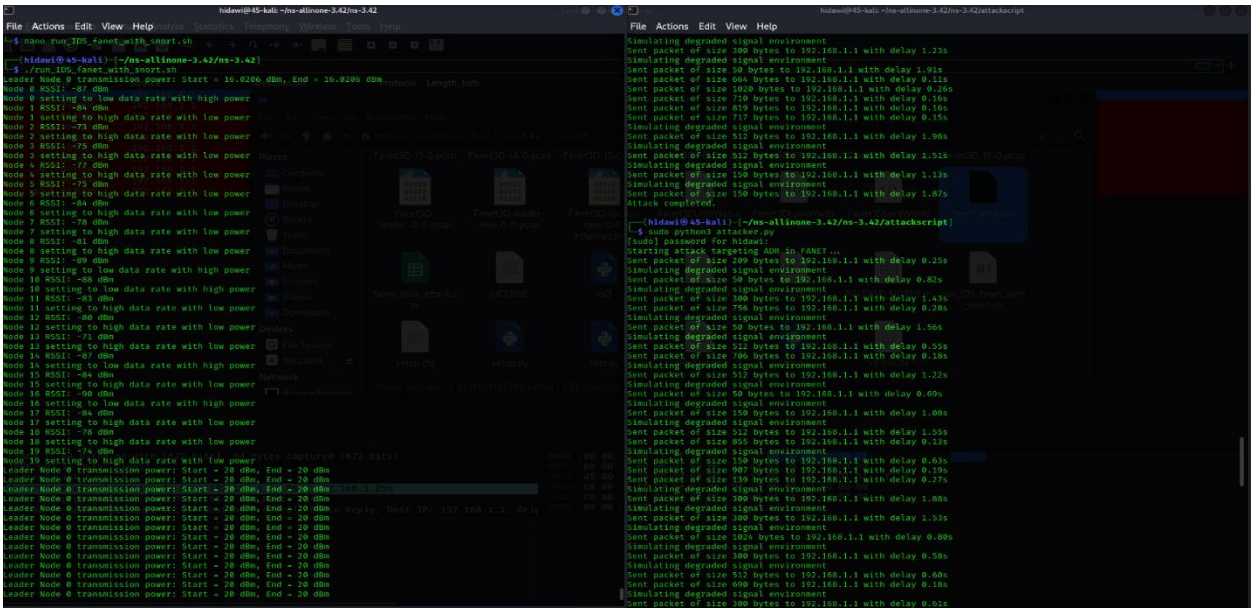


- Configuration: The script targets Node 0 by sending packets to its IP address (`192.168.1.1`). It randomly selects packet sizes between 50 and 1024 bytes and introduces delays between 0.1 and 0.5 seconds.



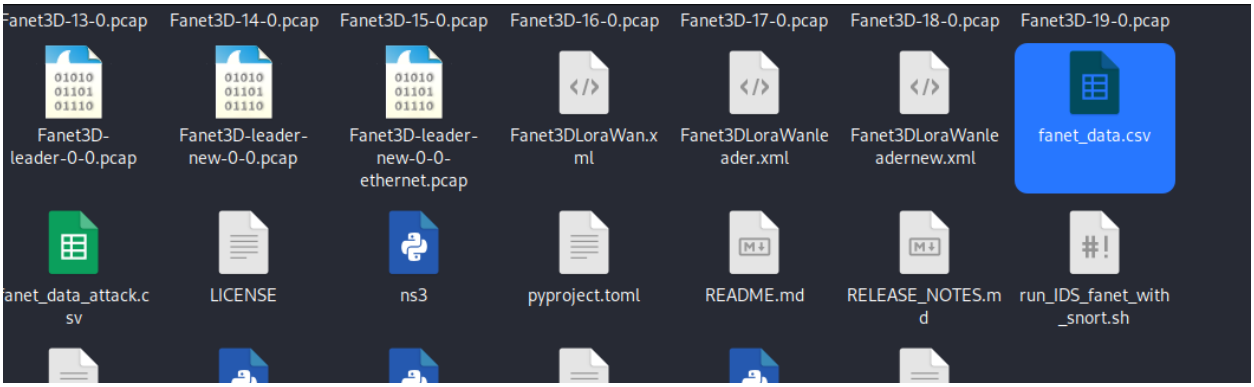
- Execution: This script runs for a specified duration (initially set to 60 seconds but extendable for prolonged attacks), continuously sending packets to simulate an environmental disturbance. This impacts the ADR and triggers power adjustments in Node 0.

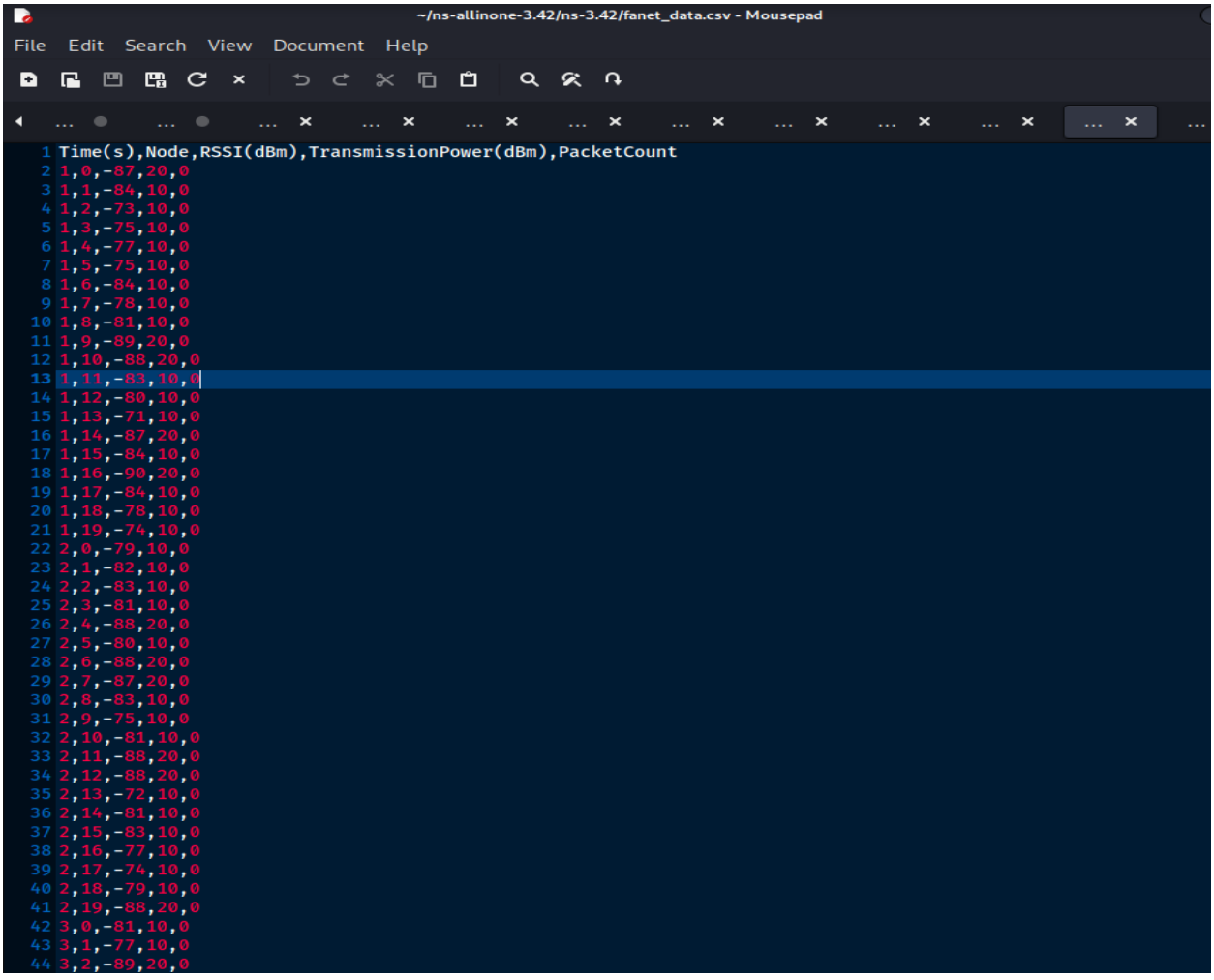
Experimentation :



1. Data Collection:

- Metric Logging: We log metrics including RSSI, transmission power, and packet count for each node. These logs are saved in CSV files for post-simulation analysis, allowing us to plot and compare behaviors between attack and no-attack scenarios.





- Visualization: Using Python, we generate graphs to visualize RSSI, transmission power, and packet counts over time. This provides insights into how the ADR and power settings respond to attack conditions.

2. Initial Findings:

- ADR Response: The simulation shows that Node 0 increases transmission power to 20 dBm when RSSI values drop below -85 dBm, which aligns with the expected ADR behavior.

- Snort Detection: So far, Snort processes packets and logs basic traffic statistics, but it does not flag the attack. The lack of alerts indicates that the current ruleset may not be sensitive to the specific traffic patterns created by the ADR-targeted attack.

Enhancements :

To fully evaluate the IDS's effectiveness, we considered several enhancements to the simulation and the attack strategy:

1. E**xtended Attack Duration:**

- Objective: Allow the attack to continue for a longer duration (e.g., 5 minutes) to observe a more significant impact on transmission power and Snort's detection capabilities.

- Implementation: Update the duration parameter in `attacker.py` to extend the script's runtime, providing a more prolonged and sustained disruption.

2. **Custom Snort Rules:**

- Objective: Detect traffic patterns indicative of power-drain attacks.

- Implementation: we created rules that detect high-frequency packet bursts, abnormal ADR adjustments, and potentially repetitive packet loss or delayed ACKs. This will help tailor Snort's detection capabilities to the specifics of FANET traffic.

3. **Distributed Topology Testing:**

- Objective: Explore whether placing IDS instances on multiple nodes improves detection capabilities, particularly for decentralized FANET deployments.

- Implementation: Deploy Snort on multiple nodes, each with tailored rules, to detect traffic abnormalities at various points in the network.

to provide detailed insights into the Snort rule customization process and the modifications we made to the `.sh` script in the FANET simulation environment .

1. Snort Rules File Setup

In our scenario, Snort is set up to monitor the FANET traffic by applying a set of rules to detect suspicious patterns. The rules file (`snort.lua` and other rule files) defines how Snort inspects packets and identifies potential attacks. These rules are built in accordance with the nature of our FANET simulation and its traffic patterns.

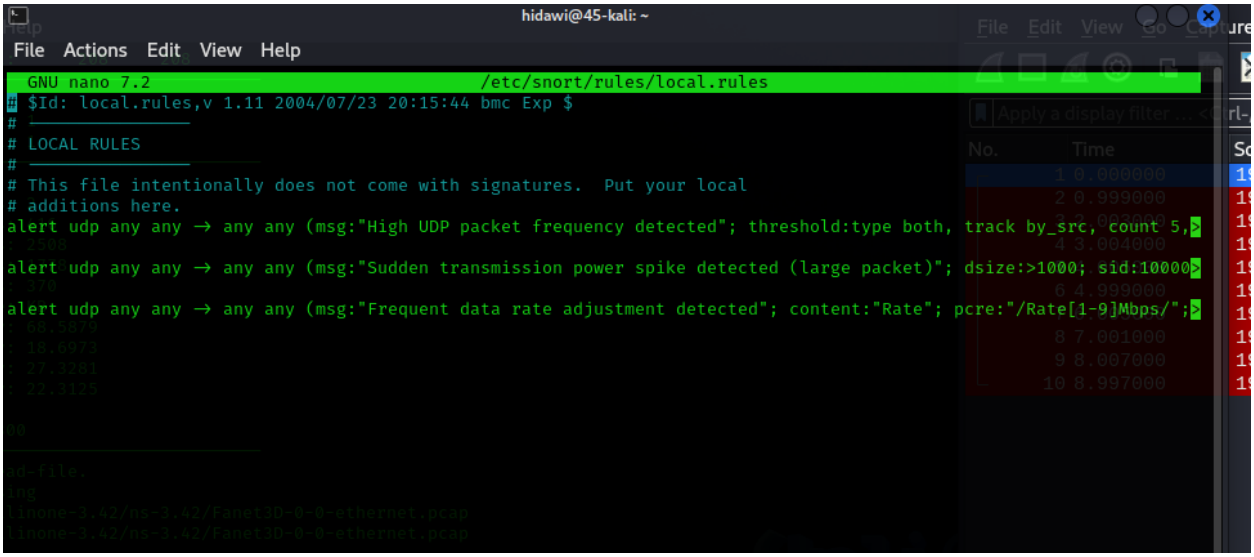
the steps for customizing Snort rules:

1. Identify Attack Signatures:

We first determined what type of attack traffic we aim to detect (e.g., packet flooding, malformed packets, or unusual traffic patterns targeting the leader node). For example, the traffic coming from the attacker script (`attacker.py`) sends randomized UDP packets targeting the leader node to simulate signal degradation and spoofing.

2. Customizing the Snort Rules file:

The researcher-playing the blue team defined specific Snort rules to detect this attack traffic. These rules typically match specific patterns in the packets, such as IP addresses, ports, payload contents, or abnormal behaviors.



- **UDP Flood Detection:**To identify if there's a large number of UDP packets being sent to a particular port (e.g., port 9 in our simulation):

```*bash*

*alert udp any any -> 192.168.1.1 9 (msg:"Potential UDP Flood Attack"; threshold:type threshold, track by_dst, count 100, seconds 10; sid:1000001;)*

```

This rule triggers an alert when more than 100 UDP packets are sent to the leader node's IP address (`192.168.1.1`) on port 9 within 10 seconds, indicative of a potential flood attack.

- **Packet Size and Payload Inspection:** To detect packets with unusual sizes or specific payload content, which the `attacker.py` script generates with randomized payload sizes.

```**bash**

**alert udp any any -> 192.168.1.1 9 (msg:"Suspicious UDP Packet Detected"; dsize:>500; sid:1000002;)**

```

This rule flags any UDP packet sent to port 9 with a size greater than 500 bytes.

3. A***djust Rule Parameters Based on Simulation Results:***

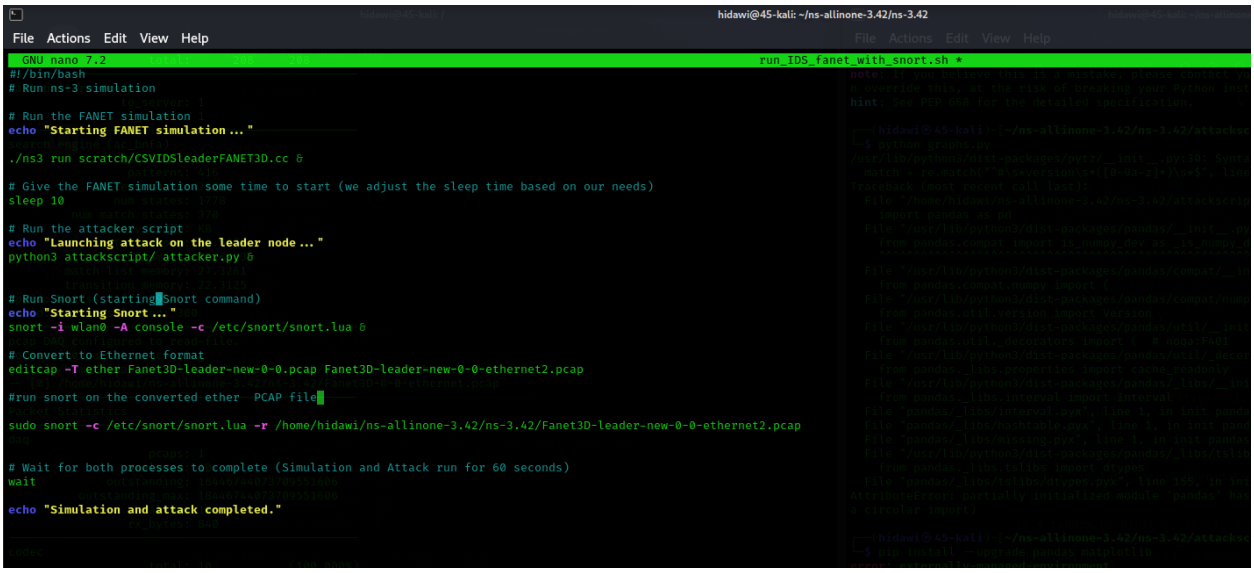
After running the simulation, the defesne team fine-tuned these rules based on the types of attacks we are simulating. The threshold for detecting a flood attack was adjusted based on traffic load from normal FANET communication, ensuring a balance between avoiding false positives and catching the real attack traffic.

Snort Configuration (`snort.lua`):

The configuration file (`snort.lua`) was customized to load the above rules and configure the necessary preprocessors and detection engines. Also we ensured that the correct interfaces (such as `wlan0` or pcap files) are monitored.

2. Changes in the `.sh` Script

The blue team made changes to The `.sh` script designed to automate the entire process of running the FANET simulation, when launching the attack by the red team, and monitoring traffic with Snort:



b) Running Snort:

Snort is executed in two phases:

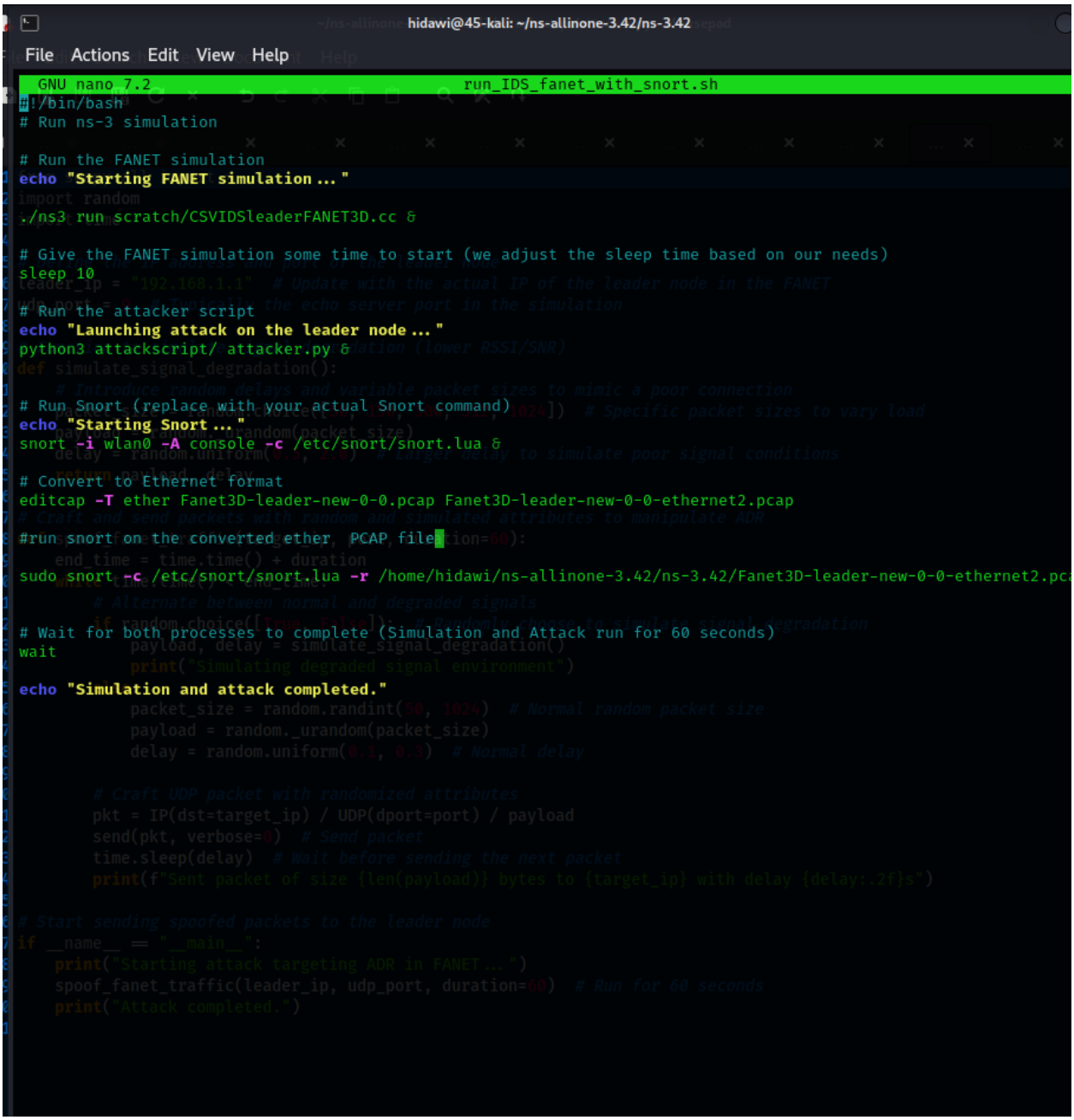
1. Real-Time Monitoring: Snort monitors the FANET traffic in real-time on the wireless interface (e.g., `wlan0`). This allows us to detect attacks as they occur during the simulation:

```bash
snort -i wlan0 -A console -c /etc/snort/snort.lua &
```

2. Offline Analysis Using Pcap: After the simulation, the wireless packets are converted to Ethernet format using `editcap`, and Snort is run on the resulting pcap file for offline analysis:

```bash
editcap -T ether Fanet3D-leader-new-0-0.pcap Fanet3D-leader-new-0-0-ethernet2.pcap
sudo snort -c /etc/snort/snort.lua -r /path/to/Fanet3D-leader-new-0-0-ethernet2.pcap
```

This step allows us to perform deeper packet analysis post-simulation, verifying whether the attack traffic was properly logged and detected.



Moving forward after finishing testing with Snort, we integrated a UAV-based ML-IDS that we have sourced from both GitHub repositories and journal papers into our same FANET simulation.

4. **Alternative IDS Models (other experiments) :**

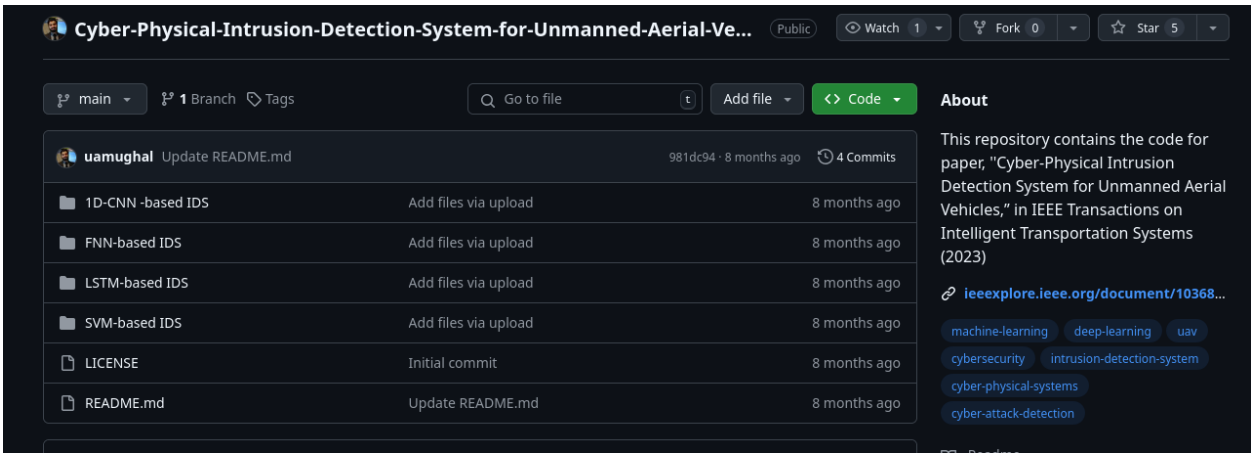
- Objective: Evaluate machine-learning-based IDS models that may be more sensitive to subtle patterns, such as those exhibited by ADR manipulations.

- Implementation: After gathering a comprehensive dataset from Snort and ns-3 and open sourced repositiores , we train ML-based IDS models to recognize the LoRaWAN-based attack. The plan includes testing with both centralized and decentralized ML-IDS configurations.

5. A**dditional Metrics:**

- Objective: Gain a more holistic view of network conditions and ADR adjustments.

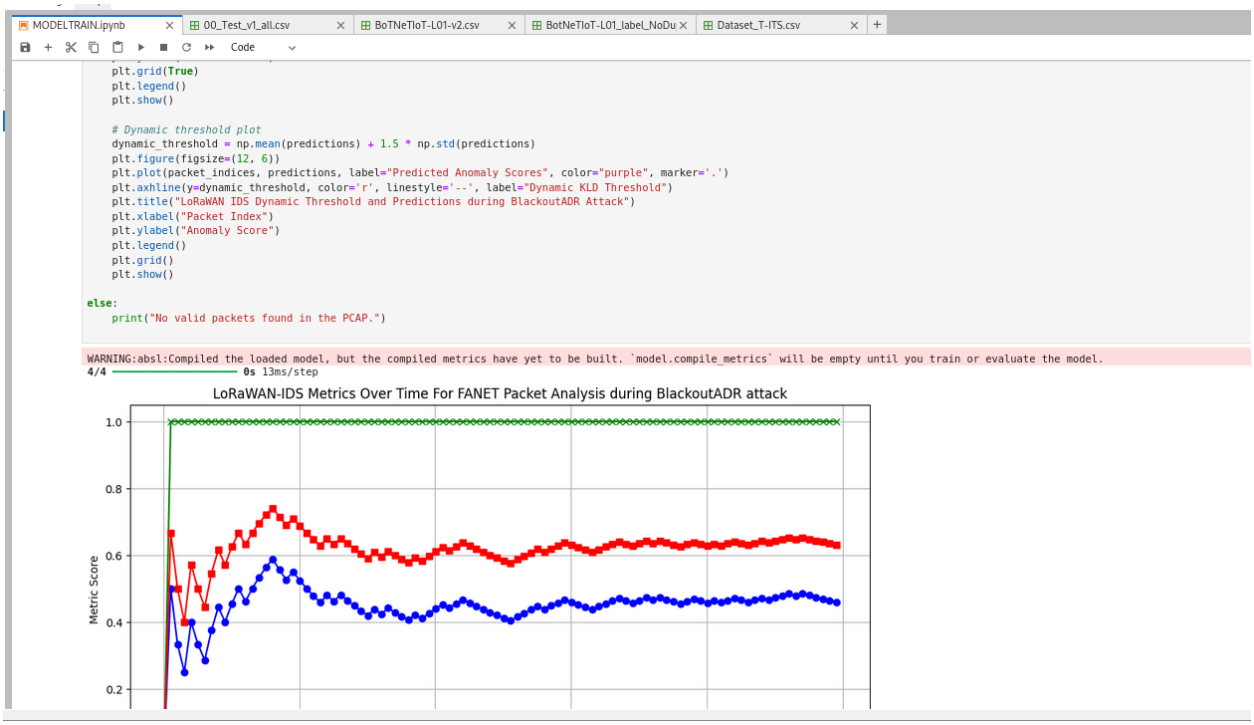
- Implementation: Incorporate metrics such as energy consumption, retransmissions, and round-trip time (RTT), which can serve as additional indicators of strain or network manipulation.



We adapt these into our existing NS3 environment(example of ML -IDS model files in attached as well the IEEE paper for the IDS proposed).

ref : S. C. Hassler, U. A. Mughal and M. Ismail, "Cyber-Physical Intrusion Detection System for Unmanned Aerial Vehicles," in IEEE Transactions on Intelligent Transportation Systems, vol. 25, no. 6, pp. 6106-6117, June 2024, doi: 10.1109/TITS.2023.3339728.

This involved capturing network traffic data (features like packet size, inter-arrival times, and RSSI) from the leader node, feeding it into the ML-based IDS module in real-time. Using NS3's Python bindings (PyNs3), we embed the pre-trained model directly into the simulation, allowing the IDS to classify traffic and detect potential threats dynamically. This process replace Snort as the detection mechanism, enabling us to assess the effectiveness of ML-based detection in identifying complex attack patterns like those generated by `attacker.py` (our attack targeting ADR in LOraWAN). The integration also involve real-time feedback, alerting, and the possibility of adaptive learning for continuous improvement in detection accuracy.



LoRaWAN-based IDS Implementation for Join Procedure Anomalies

The LoRaWAN-based IDS targets anomalies in the LoRaWAN join procedure using Kullback-Leibler Divergence (KLD) and Hamming Distance (HD) algorithms in our NS-3 FANET simulation with 20 nodes. KLD measures divergence between the baseline distribution of random numbers (DevNonce) in join requests and real-time distributions, flagging significant deviations as potential replay attacks. HD computes the Hamming distance between consecutive DevNonce values in binary, detecting abnormal reductions in randomness (e.g., from jamming). Both algorithms process join request packets captured via NS-3's PcapHelper, with thresholds set per [Danish et al., 2018]. KLD and HD run on the leader node (Node 0), analyzing traffic in real-time, with KLD showing higher accuracy.

Machine Learning-based IDS Implementation for Enhanced Detection

The ML-based IDS, adapted from [Hassler et al., 2024], is integrated into our NS-3 FANET simulation using PyNs3. It captures features (packet size, inter-arrival times, RSSI) from the leader

node's traffic via NS-3's tracing system. The pre-trained model, embedded in the simulation, classifies traffic in real-time, detecting subtle BlackoutADR patterns. The implementation extracts features every second, feeds them into the ML model (e.g., a neural network), and outputs anomaly scores. Alerts are generated for scores exceeding a dynamic threshold, computed via Markov-based estimation. The system supports adaptive learning by retraining on new data, improving detection over time.