NetSim IoT

ML based classifier to detect attacks in RPL based IoT networks

Applicable Release: NetSim v14.3 or higher
Applicable Version(s): NetSim Standard
Project download link: https://github.com/NetSim-TETCOS/ML-Classifier-based-Attack-detection-in-IoT-v14.3/archive/refs/heads/main.zip

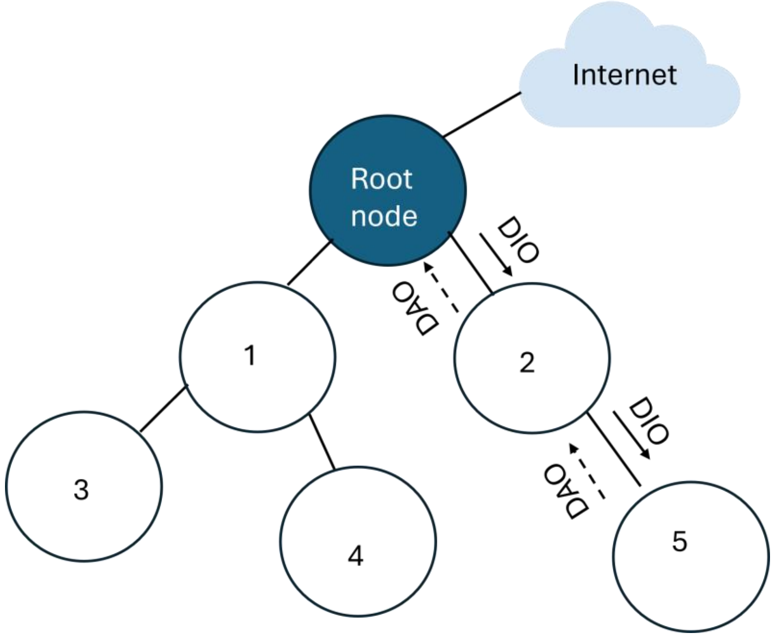
The URL has the exported NetSim scenario for the examples used in this document and the python scripts to run classifiers.

Outline

- Introduction to RPL protocol
  - Objective function and Link quality
  - Rank Calculations in NetSim
- Rank attack in RPL using NetSim
- Attack scenarios with malicious nodes - Training data
  - Attack scenarios with 2, 4, 5, 6, 8, 10, 12 and 14 malicious nodes
  - Data processing
  - Feature visualization
- Attack scenarios with malicious nodes - Test data
  - Attack scenarios with 3, 7, 9, 11, 13, and 15 malicious nodes
  - Data processing
  - Feature visualization
  - Classification
- Detection of malicious nodes using ML based classifiers
  - Confusion Matrix: Accuracy, Precision, F1 Score, Recall
  - Comparison between different classifiers: Logistic Regression, Naïve Bayes, KNN, Support Vector Machine

Introduction to RPL Protocol

- RPL: Routing Protocol for Low-Power and Lossy Networks.
- Purpose: Designed for IPv6-based routing in Low-Power and Lossy Networks (LLNs)
- Key concept: Constructs a Directed Acyclic Graph (DAG) rooted at the sink
- Goal: Minimize the cost of reaching the sink from any node based on the Objective Function (OF)
- Key Terminology:
    - DAG (Directed Acyclic Graph): A directed graph without cycles
    - DAG root: Node with o outgoing edges
    - DODAG ID: Unique IPv6 ID assigned to the root
    - Rank: Defines node positions relative to the DODAG root
- RPL implementation in NetSim is based on RFC 6550.



Control messages in RPL

RFC Reference: https://www.rfc-editor.org/rfc/rfc6550

Objective function and Link quality

- Objective Function (OF): Determines route prioritization
- NetSim implementation: OF prioritizes routes with the best link quality
- Link quality depends on:
    - Received power
    - Receiver sensitivity of nodes
- Link Quality Calculation in NetSim
    - Calculate in both direction $\left(1 - \frac{p}{r_s}\right)$
        - p is the received power (dBm)
        - r_s is the receiver sensitivity (dBm)
    - Denote as Transmit link quality, TLq and receive link quality RLq.
    - Final link quality: $Lq\ = \frac{TLq+RLq}{2}$

Rank calculation in NetSim

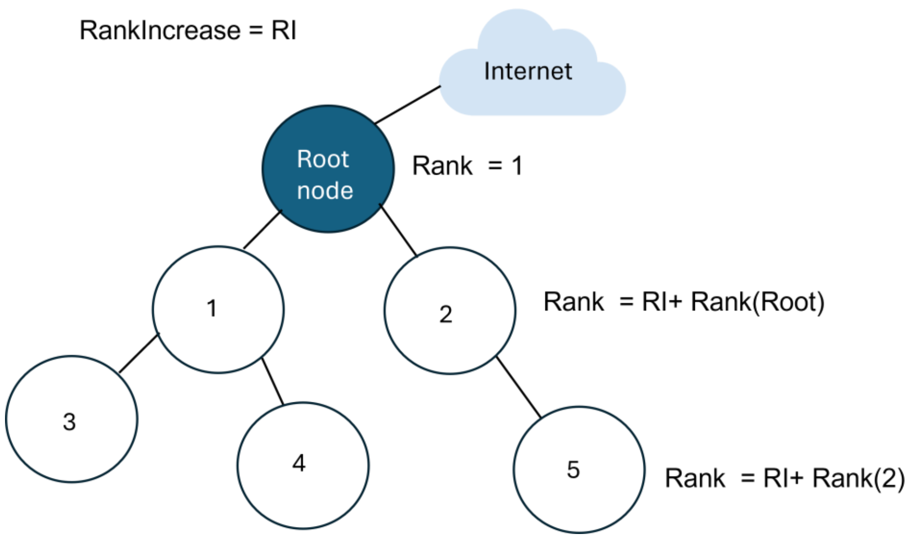
- Rank: Scalar representation of node location within DODAG
- Purpose:
  - Measure distance from root
  - Avoid and detect loops
- Root node always has Rank 1 (also the border router in IoT)
- The rank calculation is based on the objective function defined.
- Rank Increase Formula:

$$RI = (MaxIncrement - MinIncrement) \times (1 - Lq)^2 + MinIncrement$$

$$Rank = RI + Rank(Parent)$$

Where:

- RI is the Rank Increase
- MaxIncrement = 16
- MinIncrement = 1 as per RFC 6550
- Lq is the Link quality



DODAG with Node Ranks in an IoT Network

Example calculation for better understanding

- Rank of the root node: 1,
- Parent node of S2 is the root node.
- The received power at S2 can be calculated using the following formula,

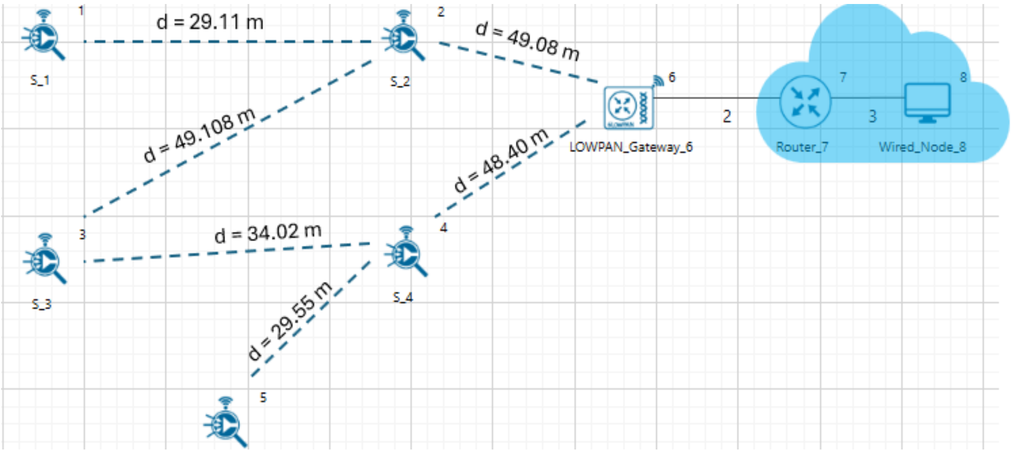
$$P_r(dBm) = P_t + G_t + G_r + 20\ \log_{10}\left(\frac{\lambda}{4\pi d0}\right) + 10\ \times \eta\ \times \log_{10}\left(\frac{d_0}{d}\right)$$

$$P_r(dBm) = 1 + 0 + 0 + 20\log_{10}\left(\frac{0.125}{4 \times\ 3.14\ \times\ 8}\right) + 10\ \times\ 3\ \times\ \log_{10}\left(\frac{8}{49.08}\right)$$

$$P_r(dBm) = -80.74$$

where, $P_t = 1mW$, d is the distance between s2 and root node, and is equal to $37.32m$, $d_0 = 8, G_t = 0,\ G_r = 0,\ \ \eta = 3$, $\lambda = \frac{c}{f} = 0.125m, f = 2400MHz$

- One way link quality $Lq = \left(1 - \frac{P_t}{r_s}\right) = 1 - \left(\frac{-80.74}{-85}\right) = 1 - 0.949 = 0.051$



The network topology in IoT using RPL Protocol,
Pathloss Model: Log Distance, Pathloss Exponent = 3
Transmit power = 1mW, Receiver Sensitivity = -85 dBm

Rank calculations in NetSim

$$Lq = \frac{TLq + RLq}{2} = \frac{0.051 + 0.051}{2} = 0.051$$

$$RankIncrease = Floor((MaxIncrement\ - MinIncrement) \times (1 - Lq)^2 + MinIncrement)$$

where, Lq = 0.051, MaxIncrement = 16, and MinIncrement = 1

$$RankIncrease = Floor((16 - 1) \times (1 - 0.051)^2 + 1) = ((15 \times 0.900) + 1) = \ floor(14.50) = \ 14$$

$$Rank = RankIncrease + Rank\ (Parent)$$

$$Rank = 14 + 1 = 15$$

- The rank of S2 is 15. We next calculate the rank for S1
- The received power at S1 can be calculated using the following formula,

$$P_r(dBm) = 1 + 0 + 0 + 20\log_{10}\left(\frac{0.125}{4 \times 3.14 \times 8}\right) + 10 \times 3 \times \log_{10}\left(\frac{8}{27.85}\right) = -73.35$$

where d is the distance between s2 and root node, $d_0 = 8, G_t, G_r = 0, \eta = 3, \lambda = \frac{c}{f} = 0.125m, f = 2400MHz$

$$Link\ quality = 1 - \left(\frac{-73.35}{-85}\right) = 1 - 0.862 = 0.137$$

Rank calculations in NetSim

$$Lq = \frac{TLq + RLq}{2} = \frac{0.137 + 0.137}{2} = 0.137$$

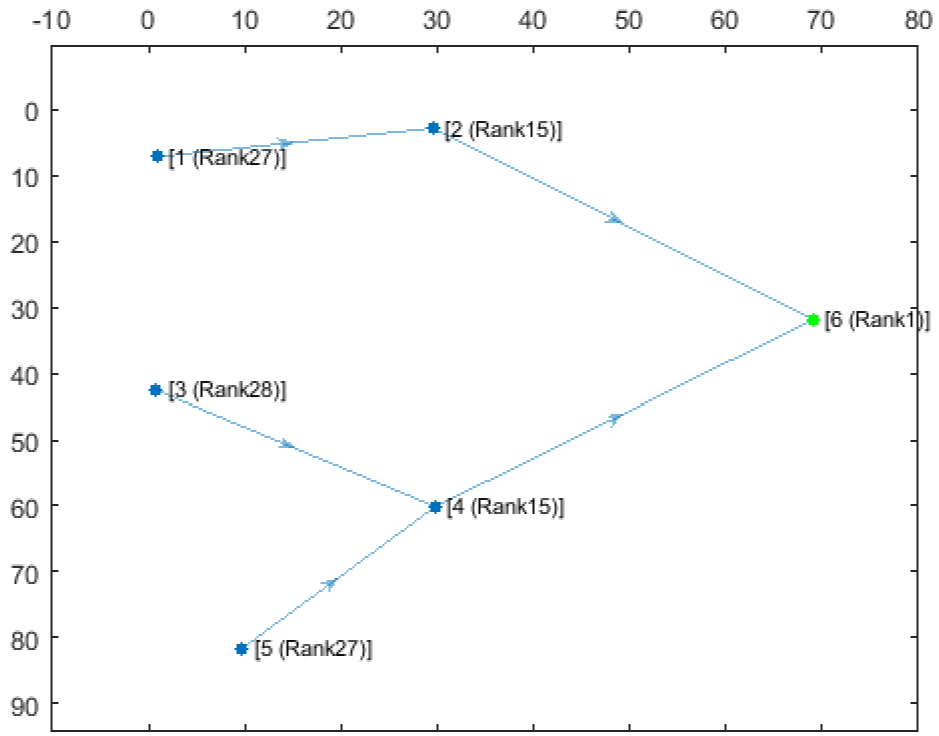
$$RankIncrease = Floor((16 - 1) \times (1 - 0.137)^2 + 1) = ((15 \times 0.744) + 1)$$

$$= floor(12.16) = 12$$

- In this case, the parent of S2 is S1, and the rank of S2 is 15

$$Rank = RankIncrease + Rank\ (Parent) = 12 + 15 = 27$$

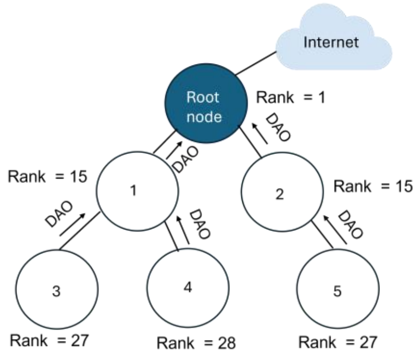
- The Rank of S1 is 27.
- Similarly, the rank for other nodes will be calculated. The rank of a node in NetSim can be observed through the DODAG Visualizer.
- We see that the Ranks of S4, S3, S5 are 15, 28, 27 respectively



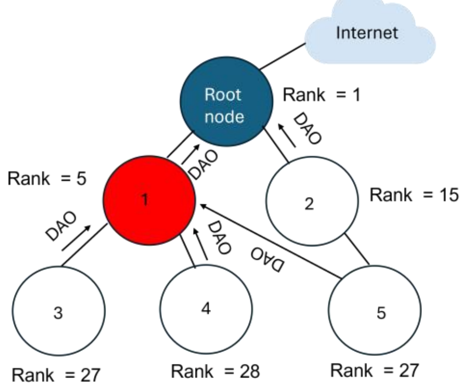
DODAG visualizer showing information about rank and parent relationships

Rank attack in RPL using NetSim

- Normal RPL process:
  - Transmitter broadcasts DIO during DODAG formation
  - Receiver updates parent list, sibling list, and rank
  - Receiver sends DAO message with route information
- Malicious node behavior:
  - Receives DIO but doesn't update its rank
  - Advertises a fake (lower) rank
  - Other nodes update their rank based on this fake information
- Attack impact:
  - Nodes choose malicious node as preferred parent due to lower rank
  - Malicious node drops packets instead of forwarding
  - Result: Zero network throughput



All nodes in the network choose their parent based on link quality

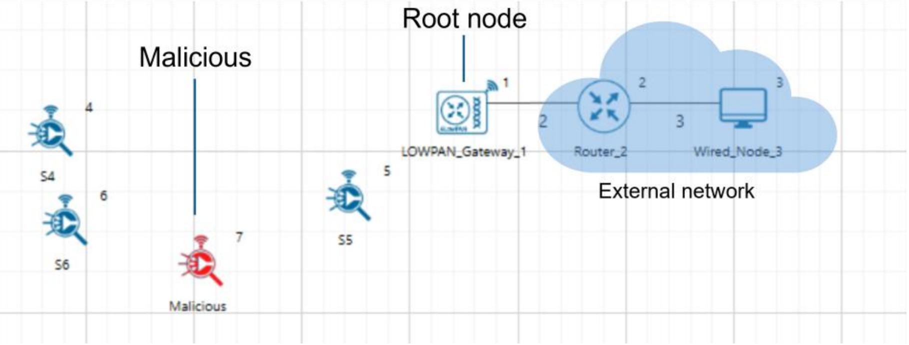


Nodes 3, 4, and 5 choose parent as node 1 due to its lower rank

Rehman et al., “Rank Attack using Objective Function in RPL for Low Power and Lossy Networks,” 2016, *IEEE International Conference on Industrial Informatics and Computer Systems (CIICS).*

Rank attack in RPL using NetSim

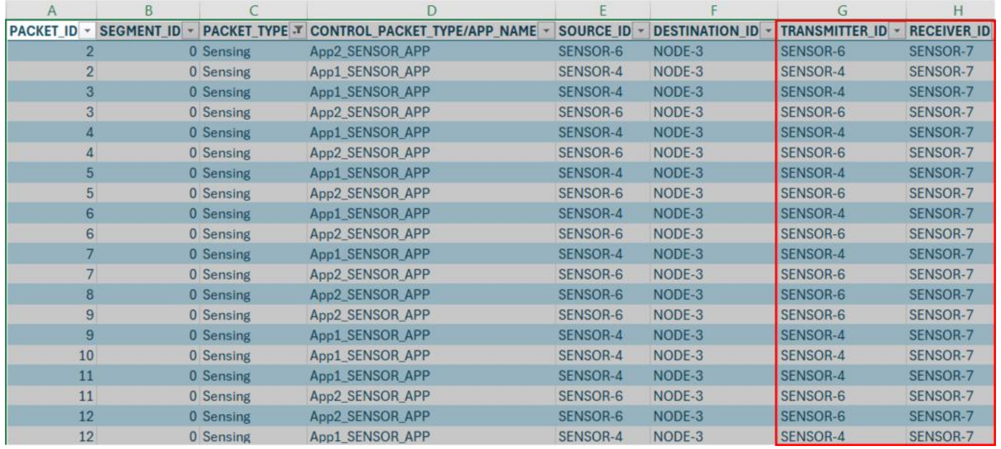
- Consider the scenario shown. The root node(LOWPAN Gateway) has rank 1. It sends DIO messages to Sensor 5 and Sensor 7, which are within its range.
- Both Sensor 5 and Sensor 7 recognize the DODAG ID of the root node. They identify the root node as their parent.
- After this, Sensor 5 and Sensor 7 transmit DAO messages to the root node. These DAO messages help to propagate destination information upward along the DODAG. Sensor 5 then updates its rank and broadcasts DIO messages.
- However, Sensor 7 is a malicious node. It also updates its rank but advertises a fake, lower rank after receiving the DIO message from the root node.
- Sensors 6 and 4 receive DIO messages from both Sensor 5 and Sensor 7. Due to Sensor 7's falsely advertised lower rank, Sensors 6 and 4 choose Sensor 7 as their preferred parent.
- After selecting Sensor 7 as their parent, Sensors 6 and 4 send DAO messages and data packets to Sensor 7.But instead of forwarding the data packets, Sensor 7 drops them.



The network topology in IoT using RPL Protocol,
Pathloss Model: Log Distance, Pathloss Exponent: 2

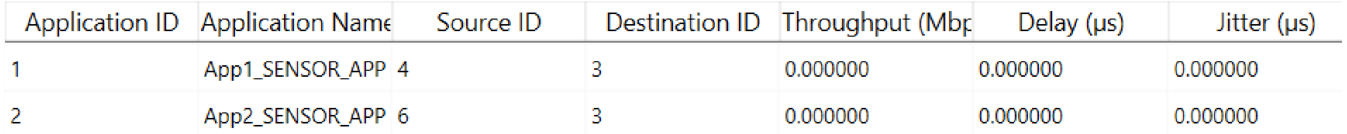
Rank attack in RPL using NetSim

- The results can be observed in the Results window, showing that the network has zero throughput.
- Users can also observe Packet trace that after Sensor 7 receives packets, it does not forward them, resulting in no data packet transmission from Sensor 7.
- Additionally, users can generate the DODAG visualizer using Python and MATLAB utilities. In the DODAG, it can be observed that Sensor 6 and Sensor 4 have chosen Sensor 7 as their parent.



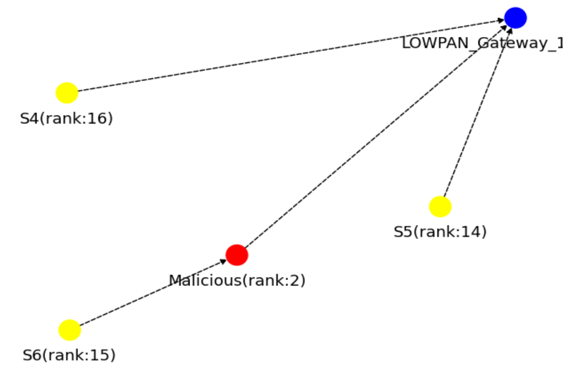
PACKET_ID	SEGMENT_ID	PACKET_TYPE	CONTROL_PACKET_TYPE/APP_NAME	SOURCE_ID	DESTINATION_ID	TRANSMITTER_ID	RECEIVER_ID
2	0	Sensing	App2_SENSOR_APP	SENSOR-6	NODE-3	SENSOR-6	SENSOR-7
2	0	Sensing	App1_SENSOR_APP	SENSOR-4	NODE-3	SENSOR-4	SENSOR-7
3	0	Sensing	App1_SENSOR_APP	SENSOR-4	NODE-3	SENSOR-4	SENSOR-7
3	0	Sensing	App2_SENSOR_APP	SENSOR-6	NODE-3	SENSOR-6	SENSOR-7
4	0	Sensing	App1_SENSOR_APP	SENSOR-4	NODE-3	SENSOR-4	SENSOR-7
4	0	Sensing	App2_SENSOR_APP	SENSOR-6	NODE-3	SENSOR-6	SENSOR-7
5	0	Sensing	App1_SENSOR_APP	SENSOR-4	NODE-3	SENSOR-4	SENSOR-7
5	0	Sensing	App2_SENSOR_APP	SENSOR-6	NODE-3	SENSOR-6	SENSOR-7
6	0	Sensing	App1_SENSOR_APP	SENSOR-4	NODE-3	SENSOR-4	SENSOR-7
6	0	Sensing	App2_SENSOR_APP	SENSOR-6	NODE-3	SENSOR-6	SENSOR-7
7	0	Sensing	App1_SENSOR_APP	SENSOR-4	NODE-3	SENSOR-4	SENSOR-7
7	0	Sensing	App2_SENSOR_APP	SENSOR-6	NODE-3	SENSOR-6	SENSOR-7
8	0	Sensing	App2_SENSOR_APP	SENSOR-6	NODE-3	SENSOR-6	SENSOR-7
9	0	Sensing	App2_SENSOR_APP	SENSOR-6	NODE-3	SENSOR-6	SENSOR-7
9	0	Sensing	App1_SENSOR_APP	SENSOR-4	NODE-3	SENSOR-4	SENSOR-7
10	0	Sensing	App1_SENSOR_APP	SENSOR-4	NODE-3	SENSOR-4	SENSOR-7
11	0	Sensing	App1_SENSOR_APP	SENSOR-4	NODE-3	SENSOR-4	SENSOR-7
11	0	Sensing	App2_SENSOR_APP	SENSOR-6	NODE-3	SENSOR-6	SENSOR-7
12	0	Sensing	App2_SENSOR_APP	SENSOR-6	NODE-3	SENSOR-6	SENSOR-7
12	0	Sensing	App1_SENSOR_APP	SENSOR-4	NODE-3	SENSOR-4	SENSOR-7

The packet trace shows that packets from Sensor-4 and Sensor-6 are received by Sensor-7, but Sensor-7 is not transmitting packets.



Application ID	Application Name	Source ID	Destination ID	Throughput (Mbp	Delay (µs)	Jitter (µs)
1	App1_SENSOR_APP	4	3	0.000000	0.000000	0.000000
2	App2_SENSOR_APP	6	3	0.000000	0.000000	0.000000

Throughput for the two applications is zero because the malicious sensor is collecting all the packets

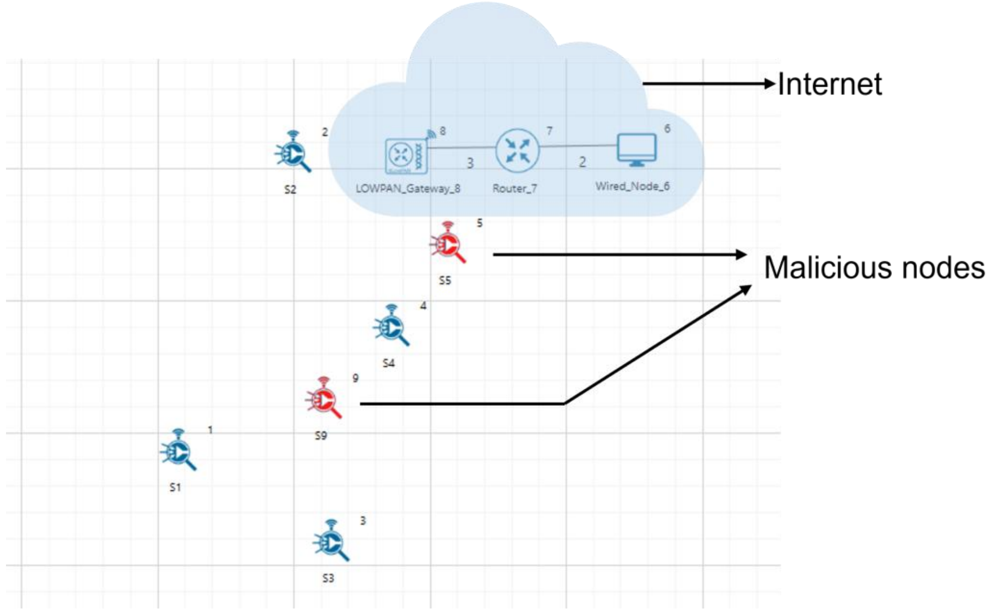


The DODAG visualizer shows that Sensor-6 and Sensor-4 are choosing Sensor-7 as a parent node.

Training

Attack scenarios - Training data generation

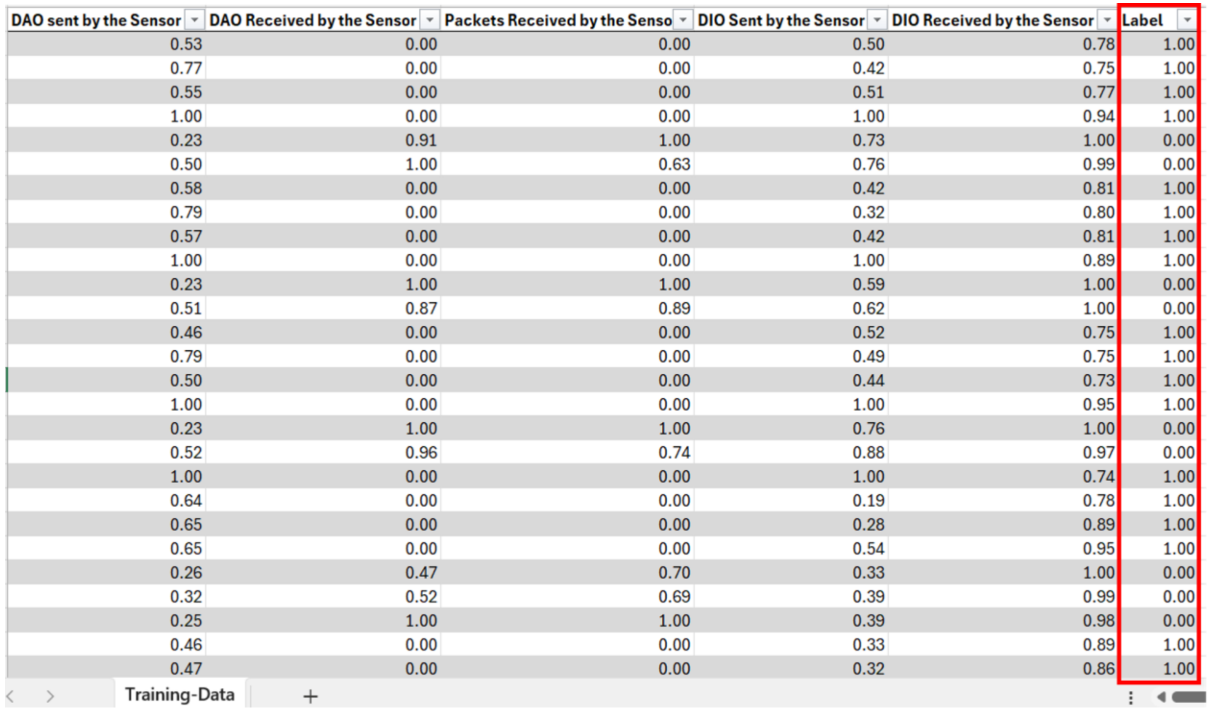
- Created 8 scenarios with varying node counts (6 to 39)
- Malicious node count: 2, 4, 5, 6, 8, 10, 12, and 14
- Simulations run with 3 random seeds for each scenario
- Enabled packet trace for all scenarios
- Used a python script to calculate the number of DAO, DIO, and data packets received by each sensor from packet trace.
- Feature Extraction
    1. DAO Sent
    2. DAO Received
    3. DIO Sent
    4. DIO Received
    5. Data Packets Received



The network topology in IoT using RPL Protocol with 2 malicious nodes

Data processing and Feature Visualization

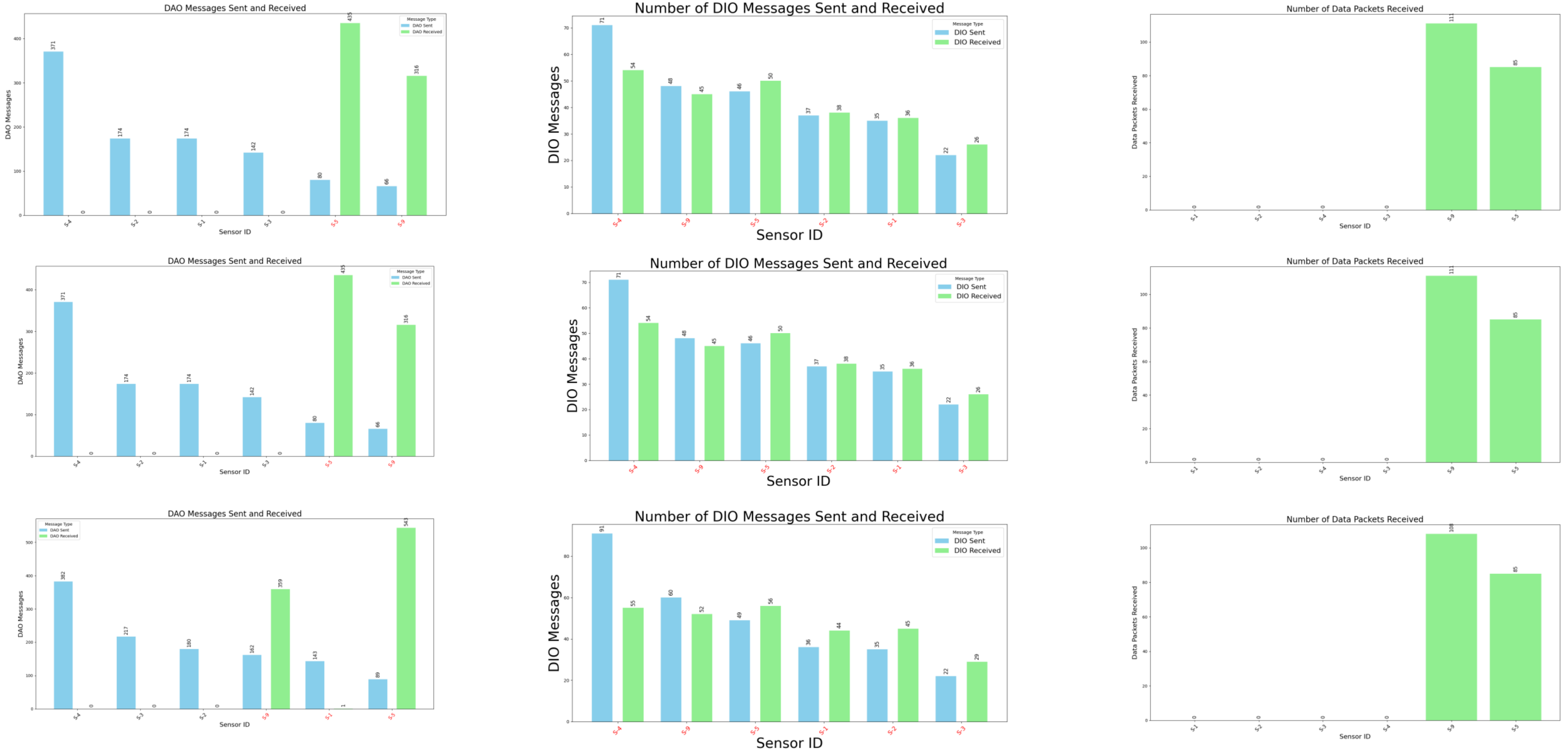
- Data extraction from packet trace to Excel using Python script
- Total dataset: 534 sensors, 5 features each
- Feature normalization process:
  - Calculate max value for each feature across all sensors
  - Divide each sensor's value by the max to get 0-1 range
- Manual labeling: 1 for non-malicious, 0 for malicious



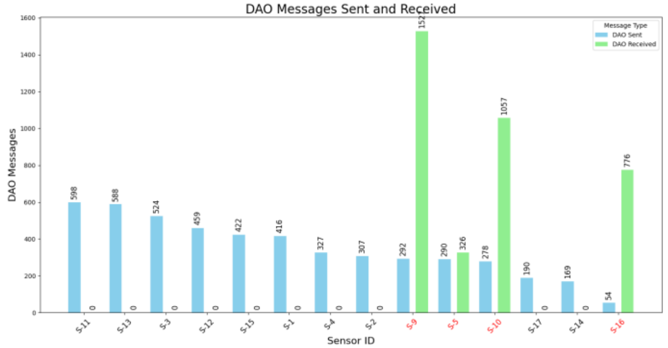
DAO sent by the Sensor	DAO Received by the Sensor	Packets Received by the Senso	DIO Sent by the Sensor	DIO Received by the Sensor	Label
0.53	0.00	0.00	0.50	0.78	1.00
0.77	0.00	0.00	0.42	0.75	1.00
0.55	0.00	0.00	0.51	0.77	1.00
1.00	0.00	0.00	1.00	0.94	1.00
0.23	0.91	1.00	0.73	1.00	0.00
0.50	1.00	0.63	0.76	0.99	0.00
0.58	0.00	0.00	0.42	0.81	1.00
0.79	0.00	0.00	0.32	0.80	1.00
0.57	0.00	0.00	0.42	0.81	1.00
1.00	0.00	0.00	1.00	0.89	1.00
0.23	1.00	1.00	0.59	1.00	0.00
0.51	0.87	0.89	0.62	1.00	0.00
0.46	0.00	0.00	0.52	0.75	1.00
0.79	0.00	0.00	0.49	0.75	1.00
0.50	0.00	0.00	0.44	0.73	1.00
1.00	0.00	0.00	1.00	0.95	1.00
0.23	1.00	1.00	0.76	1.00	0.00
0.52	0.96	0.74	0.88	0.97	0.00
1.00	0.00	0.00	1.00	0.74	1.00
0.64	0.00	0.00	0.19	0.78	1.00
0.65	0.00	0.00	0.28	0.89	1.00
0.65	0.00	0.00	0.54	0.95	1.00
0.26	0.47	0.70	0.33	1.00	0.00
0.32	0.52	0.69	0.39	0.99	0.00
0.25	1.00	1.00	0.39	0.98	0.00
0.46	0.00	0.00	0.33	0.89	1.00
0.47	0.00	0.00	0.32	0.86	1.00

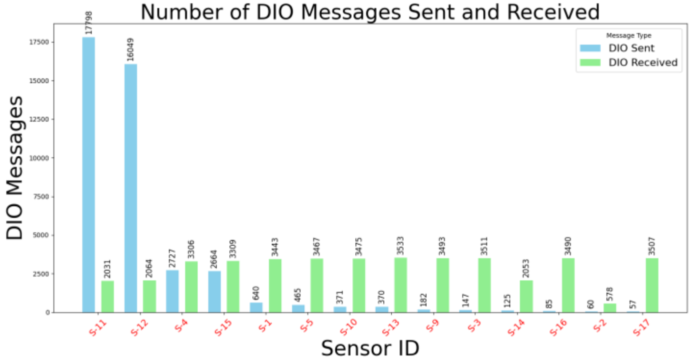
We label the sensors based on the features

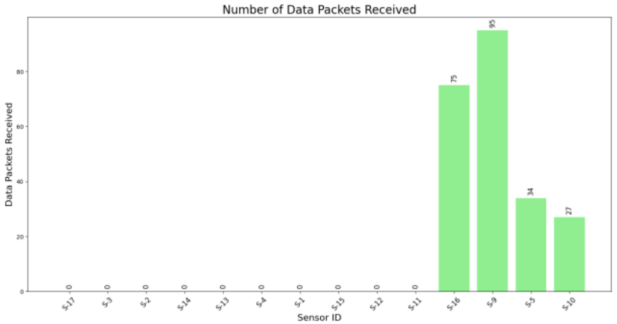
Feature visualization: 2 malicious nodes; 3 runs, each with a different random seed

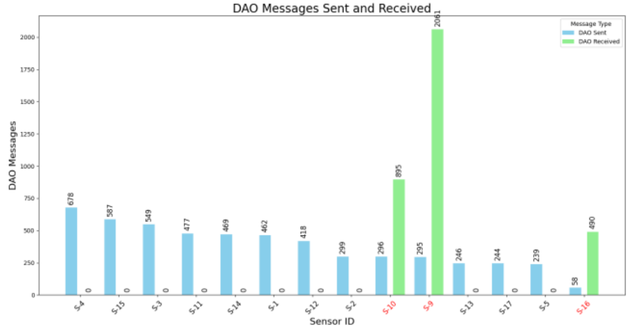


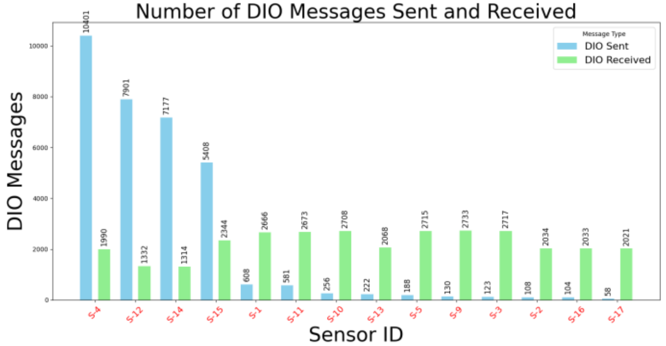
Feature visualization: 4 malicious nodes; 3 runs, each with a different random seed

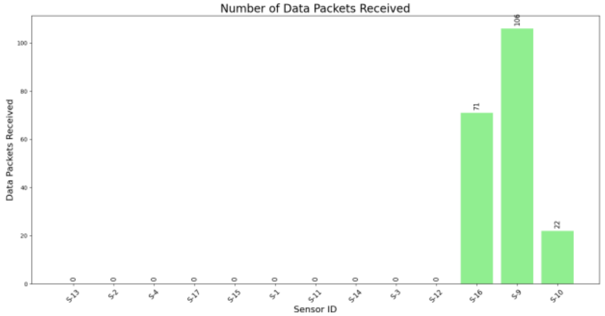


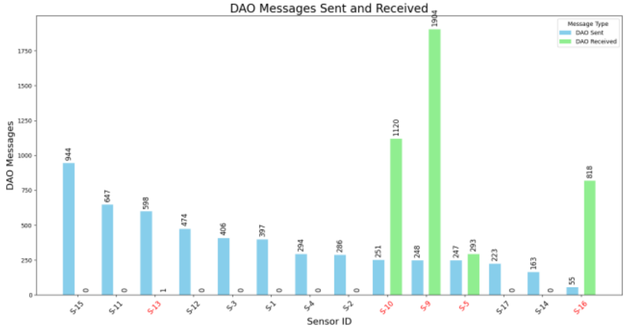


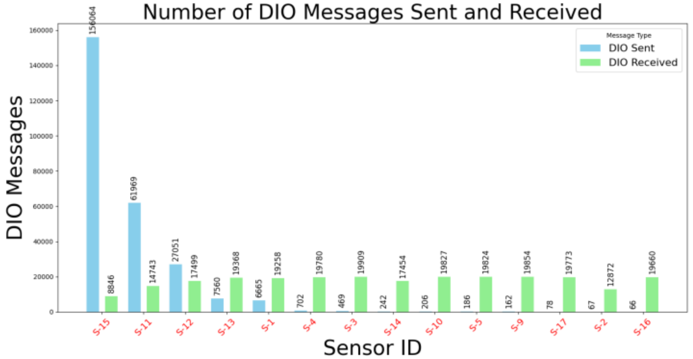


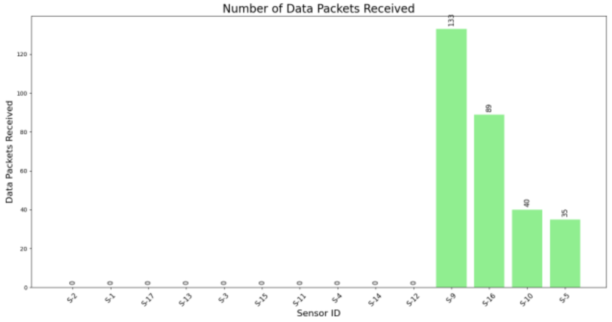




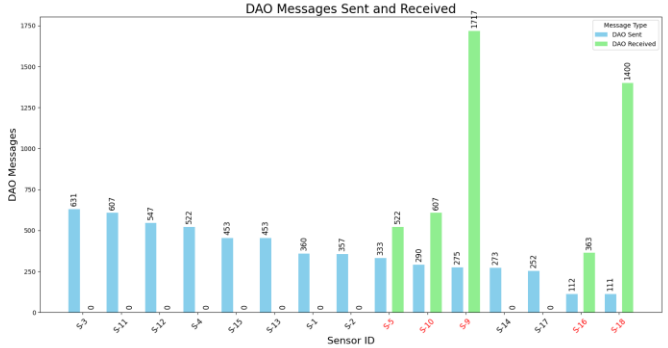


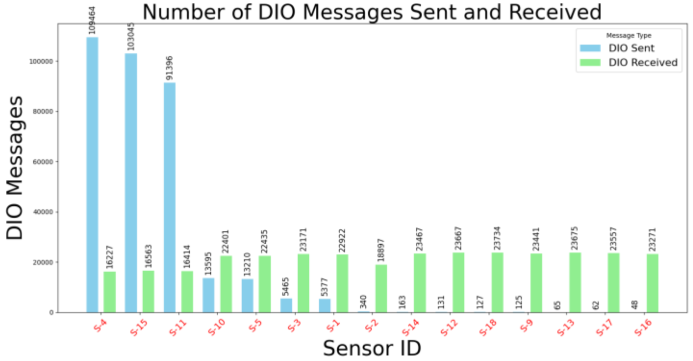


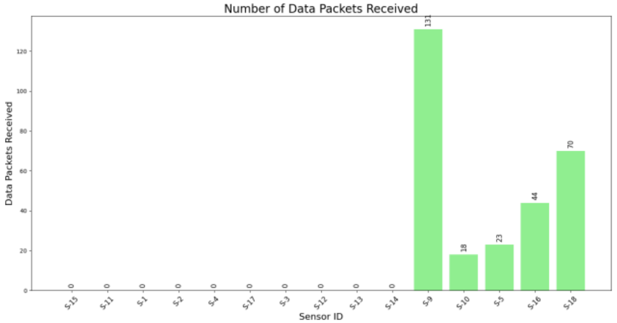


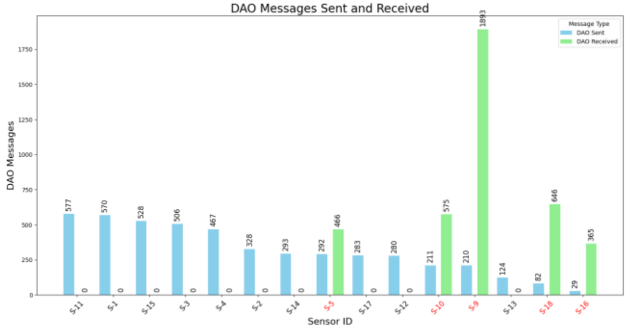


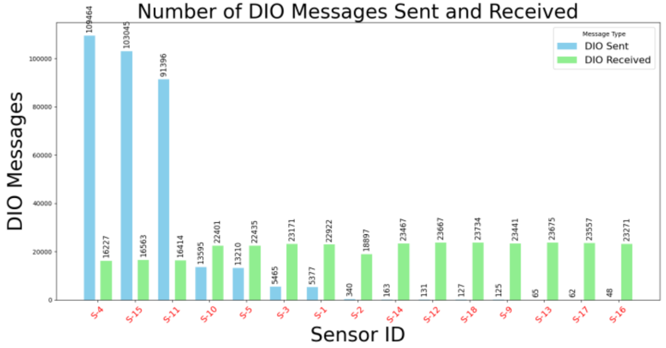
Feature visualization: 5 malicious nodes; 3 runs, each with a different random seed

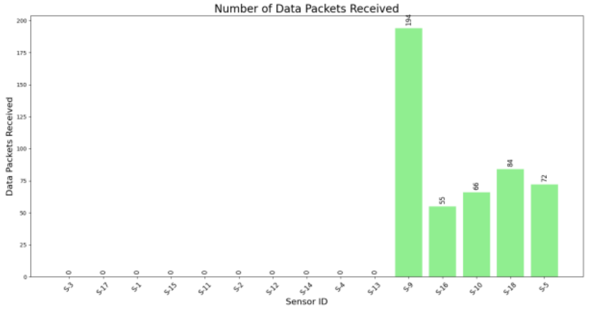


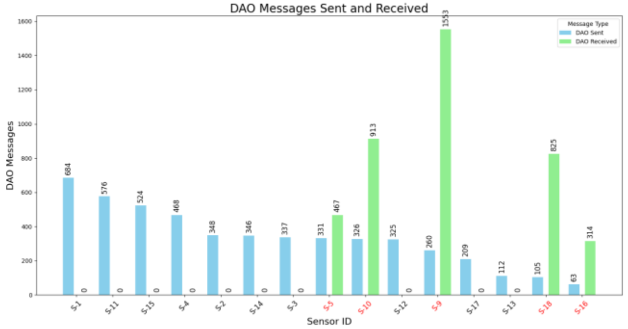


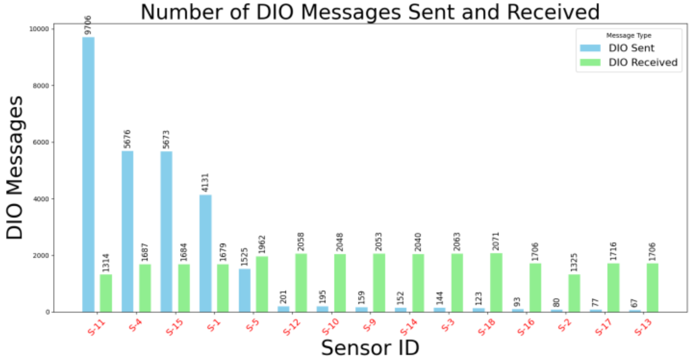


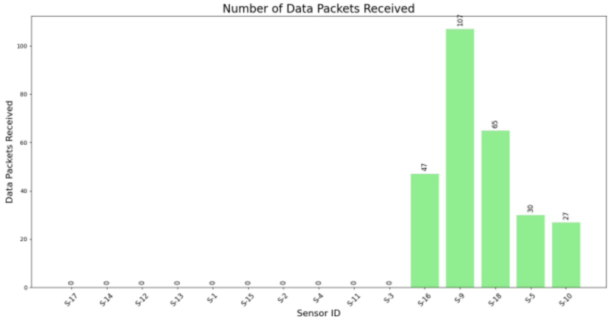




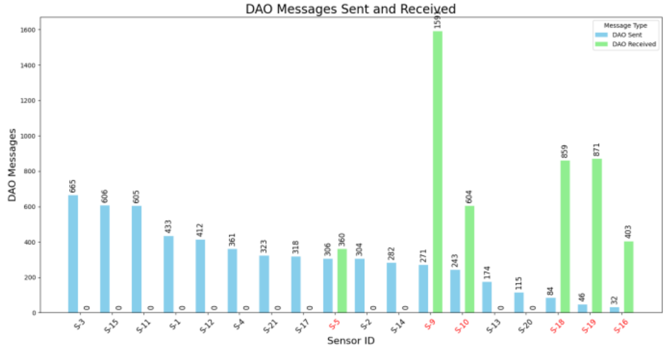


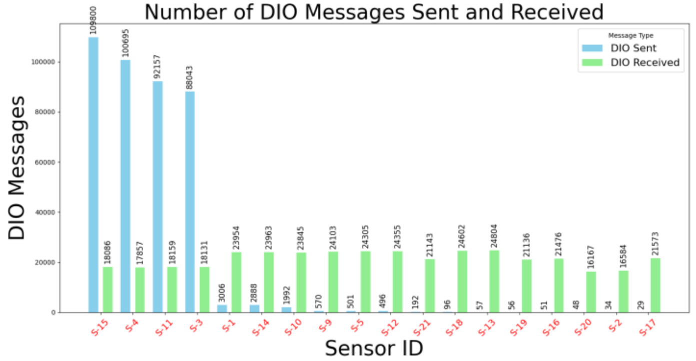


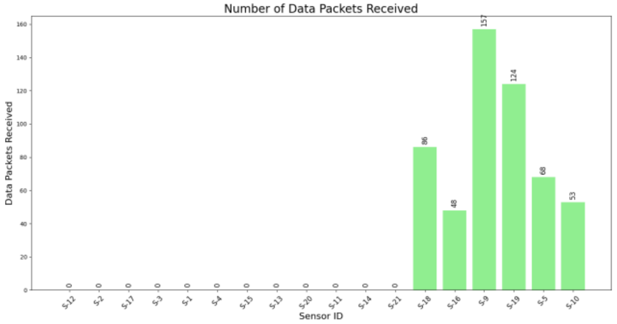


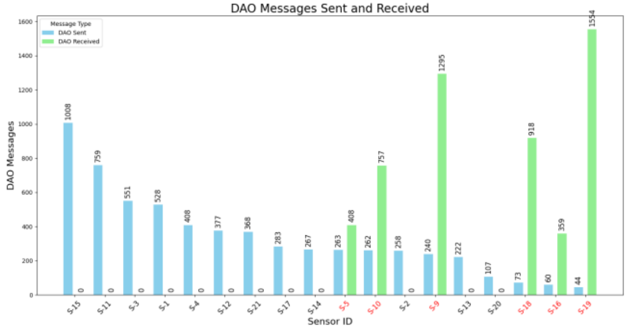


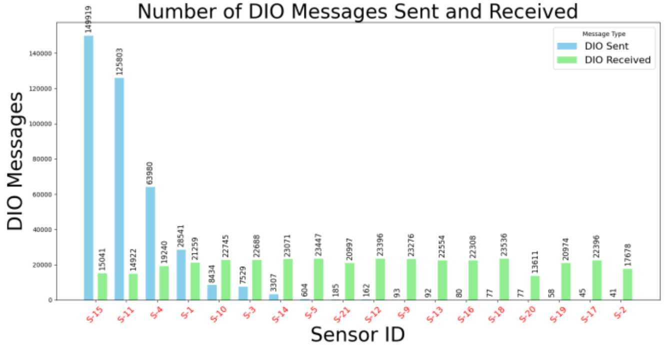
Feature visualization: 6 malicious nodes; 3 runs, each with a different random seed

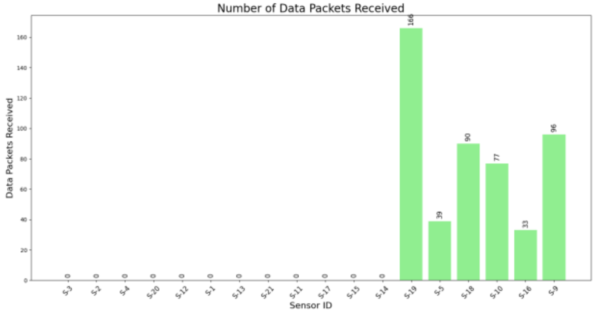


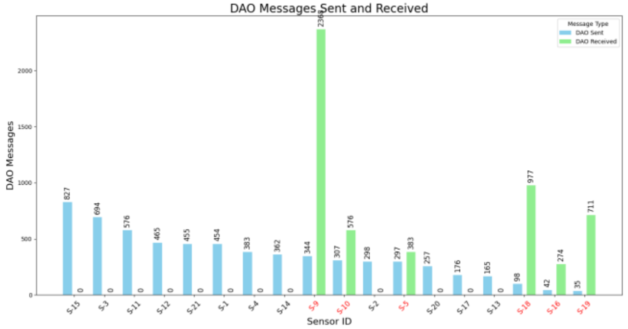


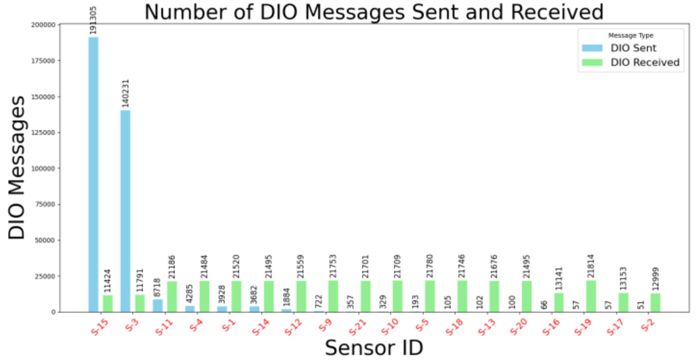


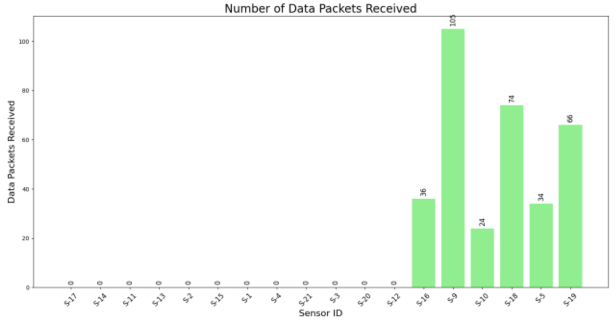




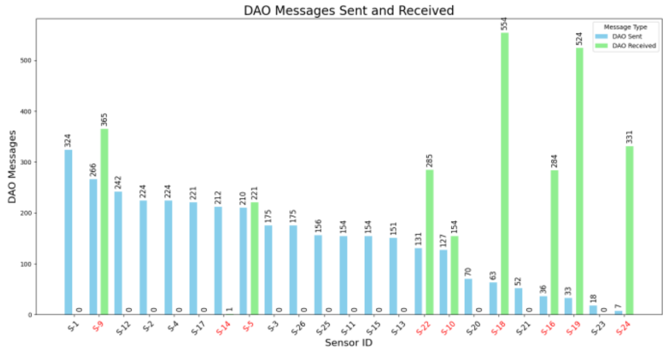


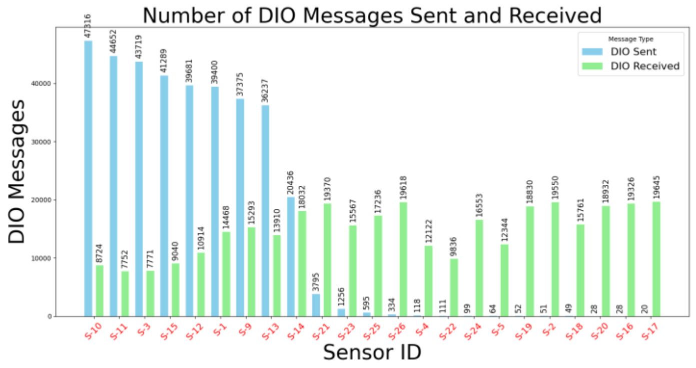


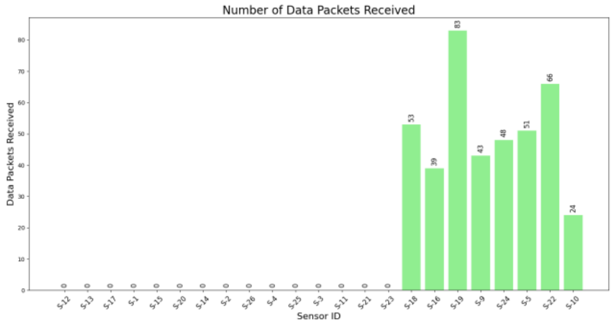


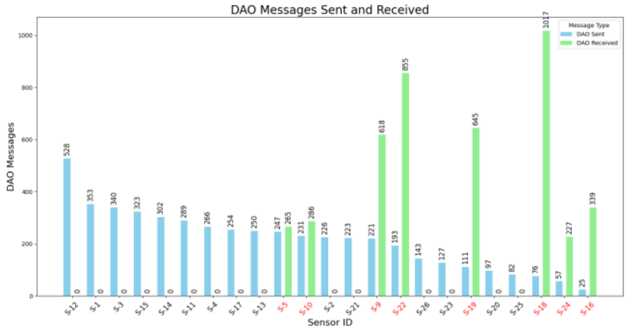


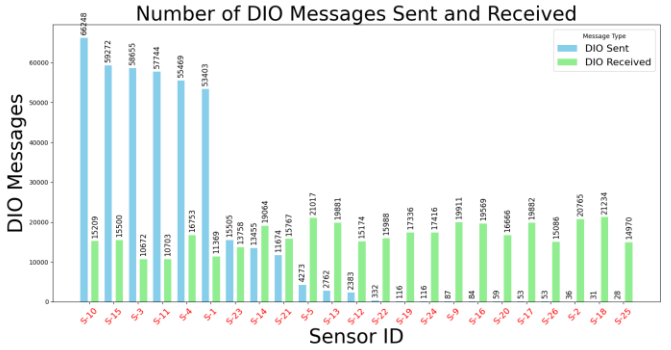
Feature visualization: 8 malicious nodes; 3 runs, each with a different random seed

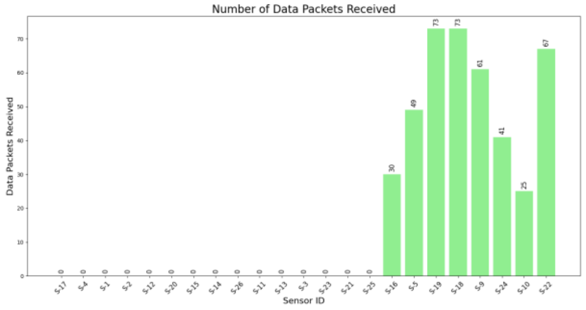


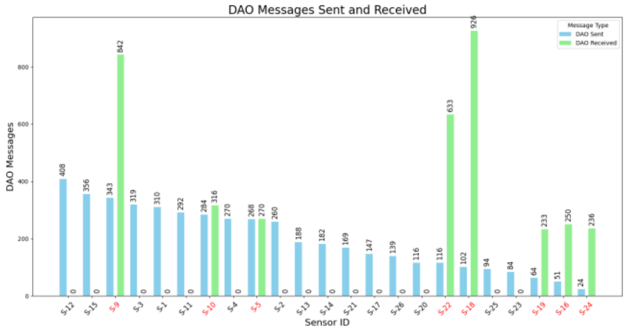


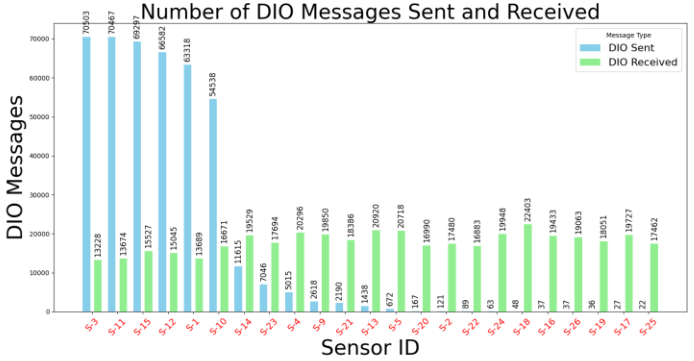


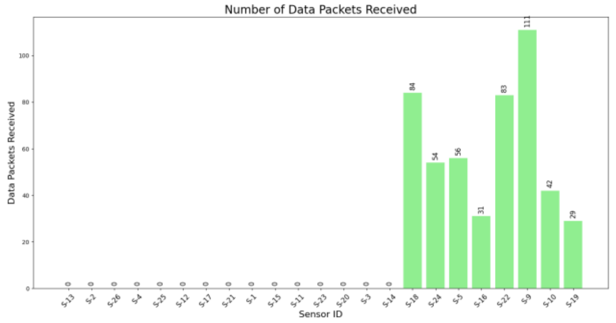




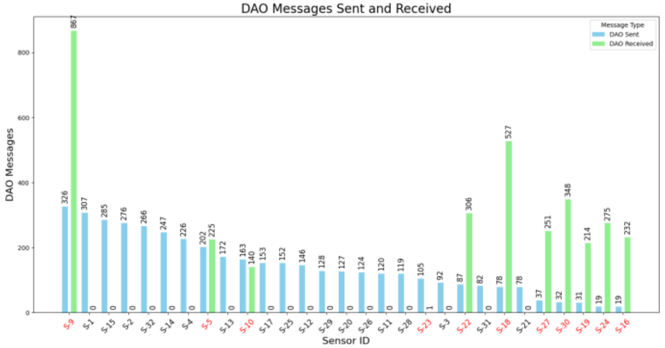


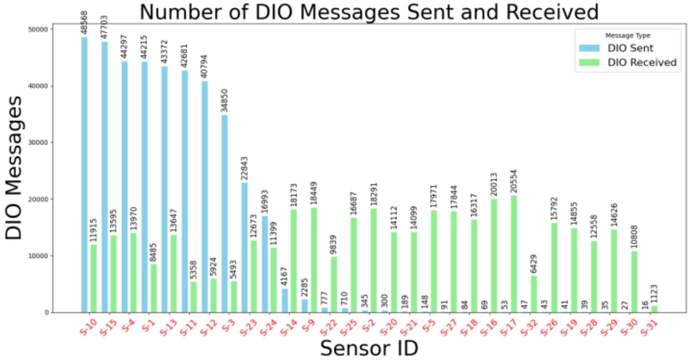


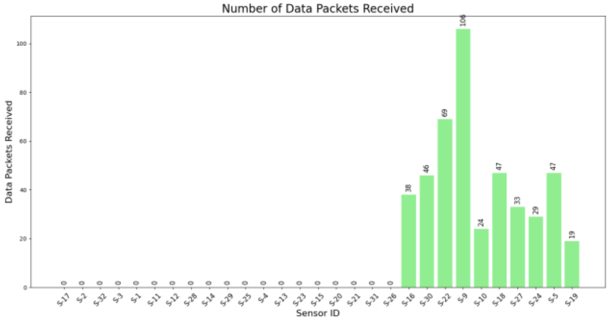


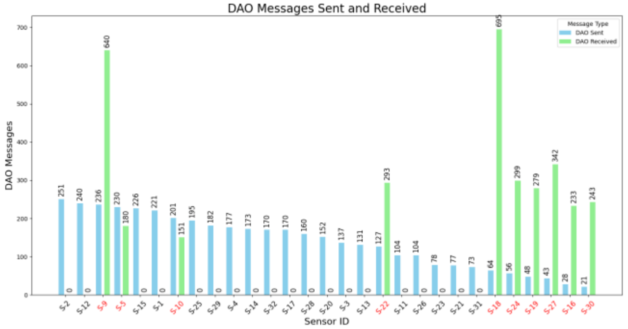


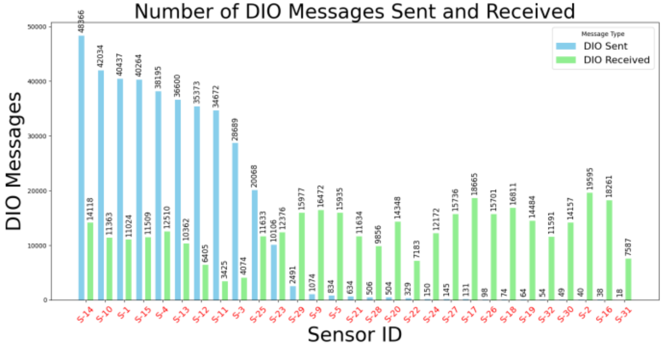
Feature visualization: 10 malicious nodes; 3 runs, each with a different random seed

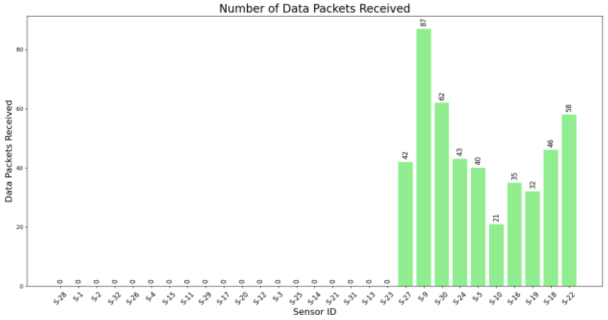

DAO Messages Sent and Received

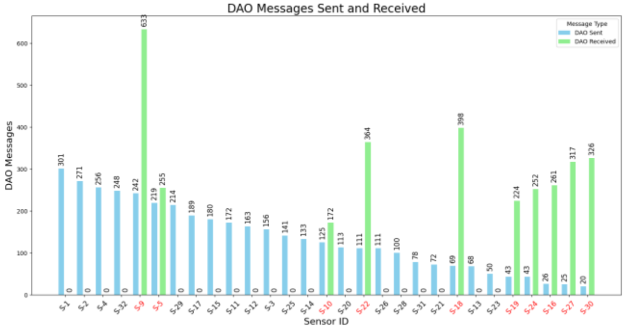

Number of DIO Messages Sent and Received

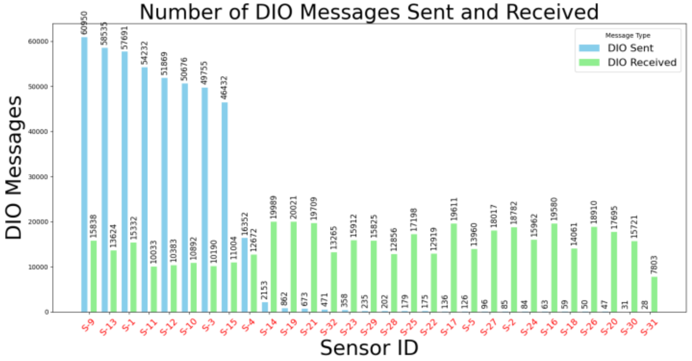

Number of Data Packets Received

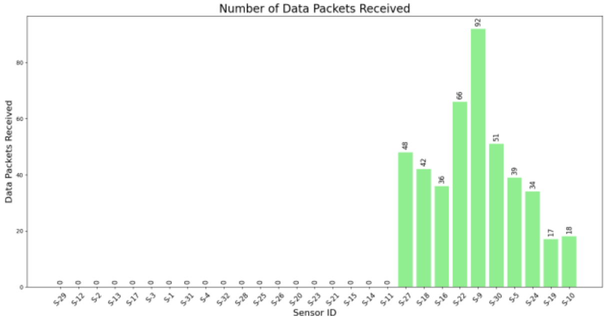

DAO Messages Sent and Received


Number of DIO Messages Sent and Received

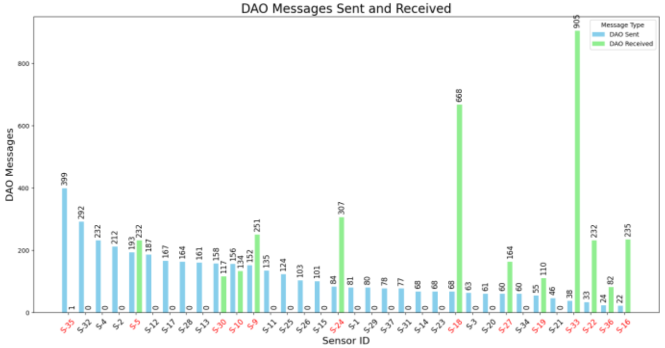

Number of Data Packets Received

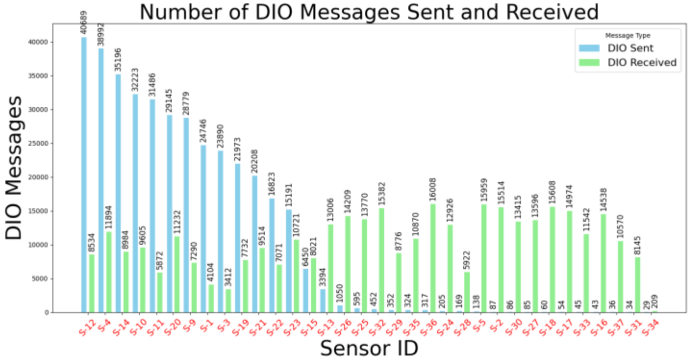

DAO Messages Sent and Received

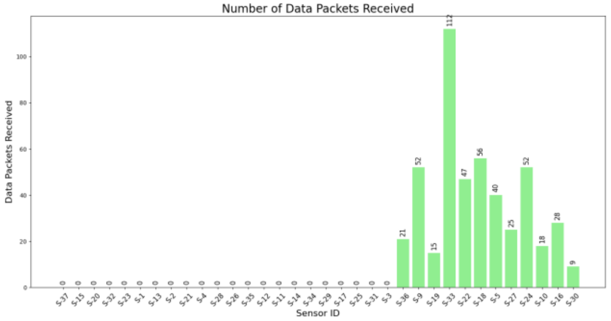

Number of DIO Messages Sent and Received

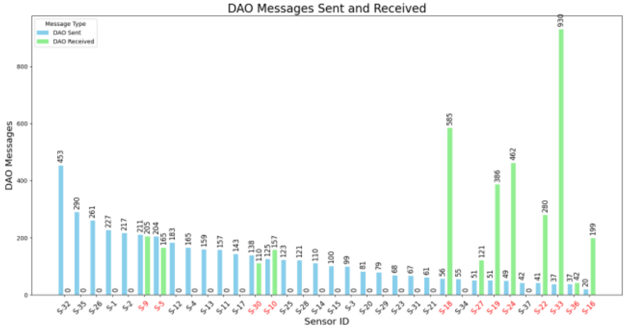

Number of Data Packets Received

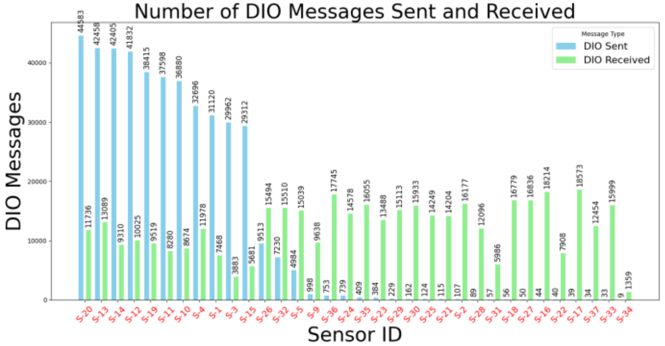
Feature visualization: 12 malicious nodes; 3 runs, each with a different random seed

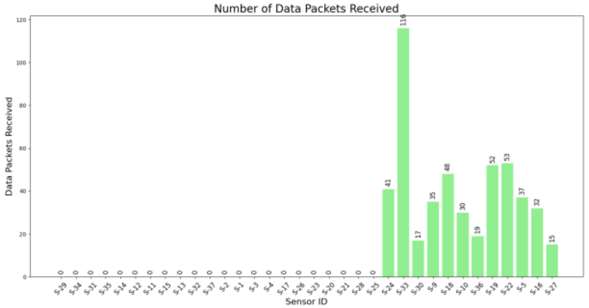

DAO Messages Sent and Received

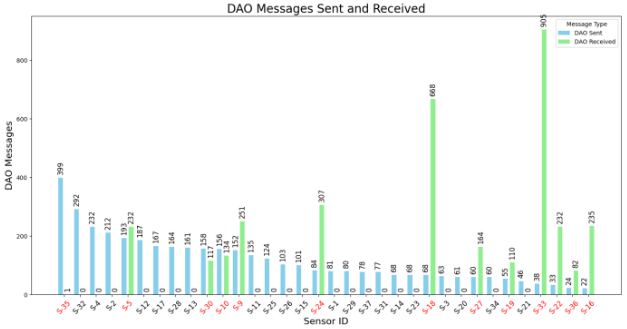

Number of DIO Messages Sent and Received

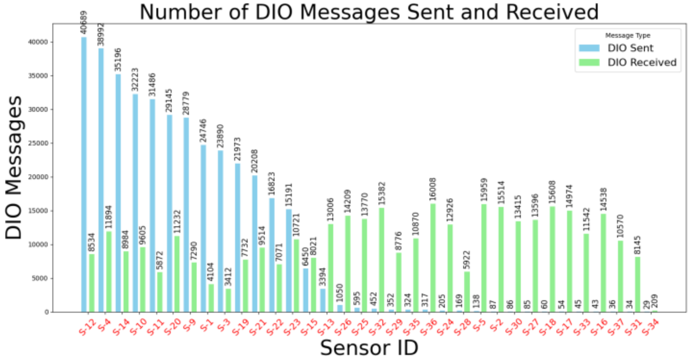

Number of Data Packets Received

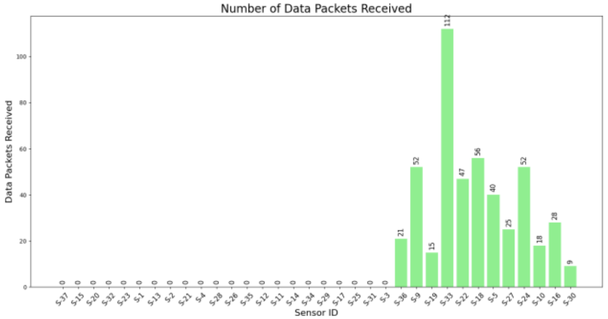

DAO Messages Sent and Received


Number of DIO Messages Sent and Received

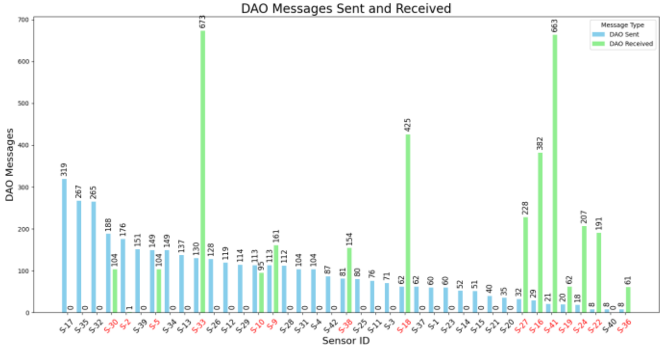

Number of Data Packets Received

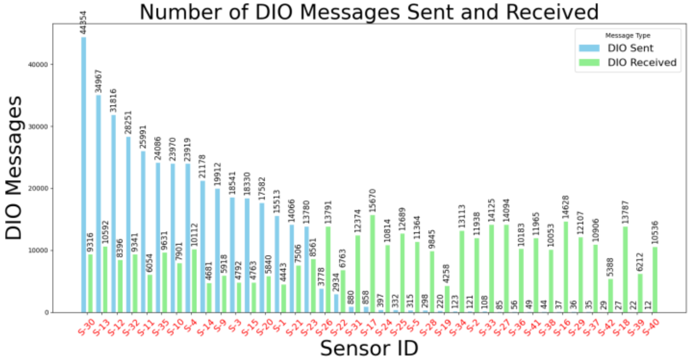

DAO Messages Sent and Received

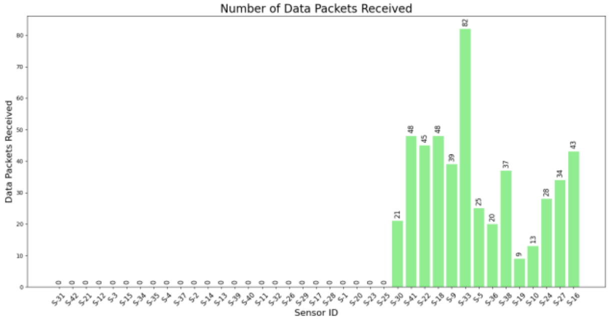

Number of DIO Messages Sent and Received

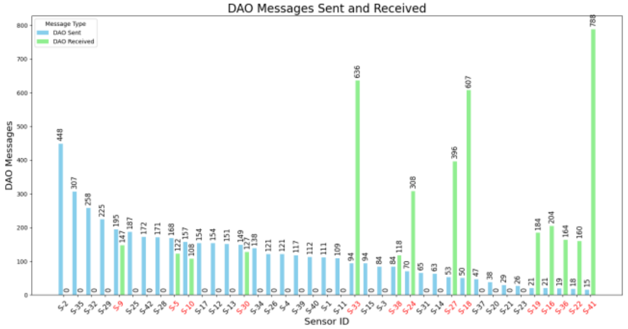

Number of Data Packets Received

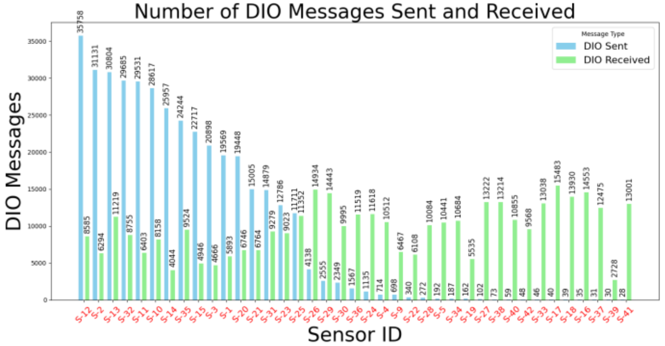
Feature visualization: 14 malicious nodes; 3 runs, each with a different random seed

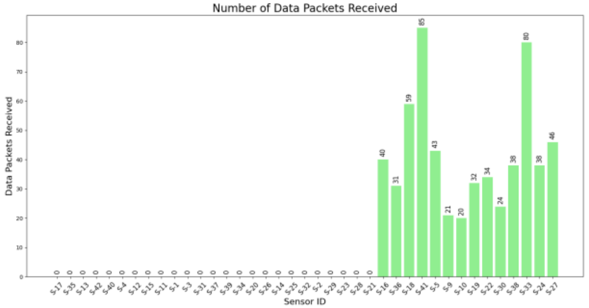


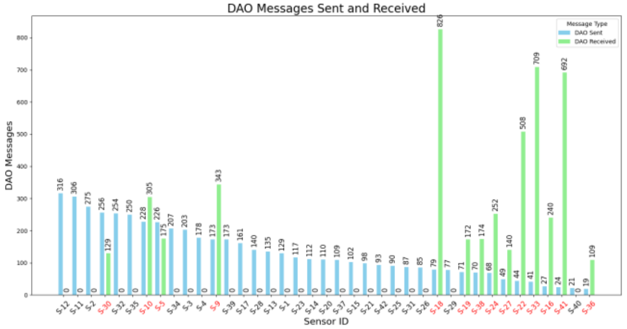


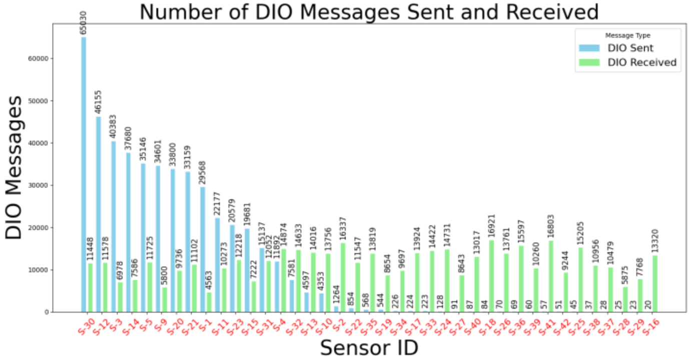


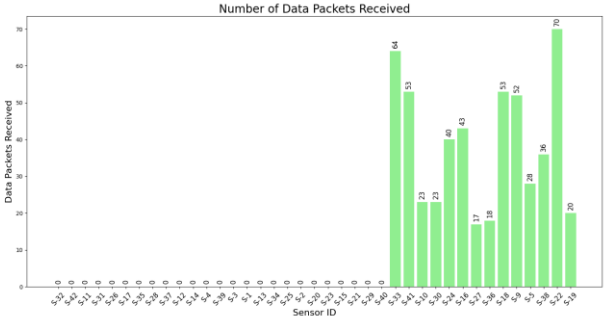












Classifier Training

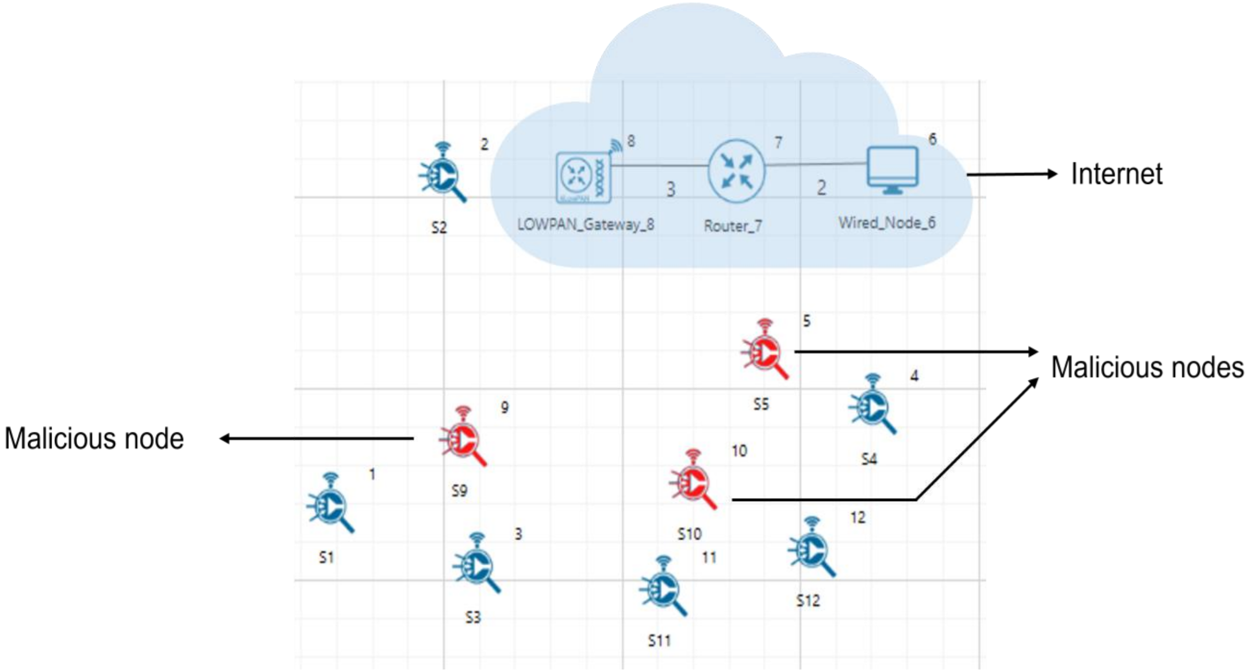
Features data was used to train the following classifiers:

- K-Nearest Neighbor
- Naive Bayes
- Support Vector Machine
- Logistic Regression

Inference

Inference and Test Scenarios

- Created 6 new scenarios with different node counts (9 to 42)
- Malicious node count: 3, 7, 9, 11, 13, and 15
- Simulations run with 3 random seeds for each scenario



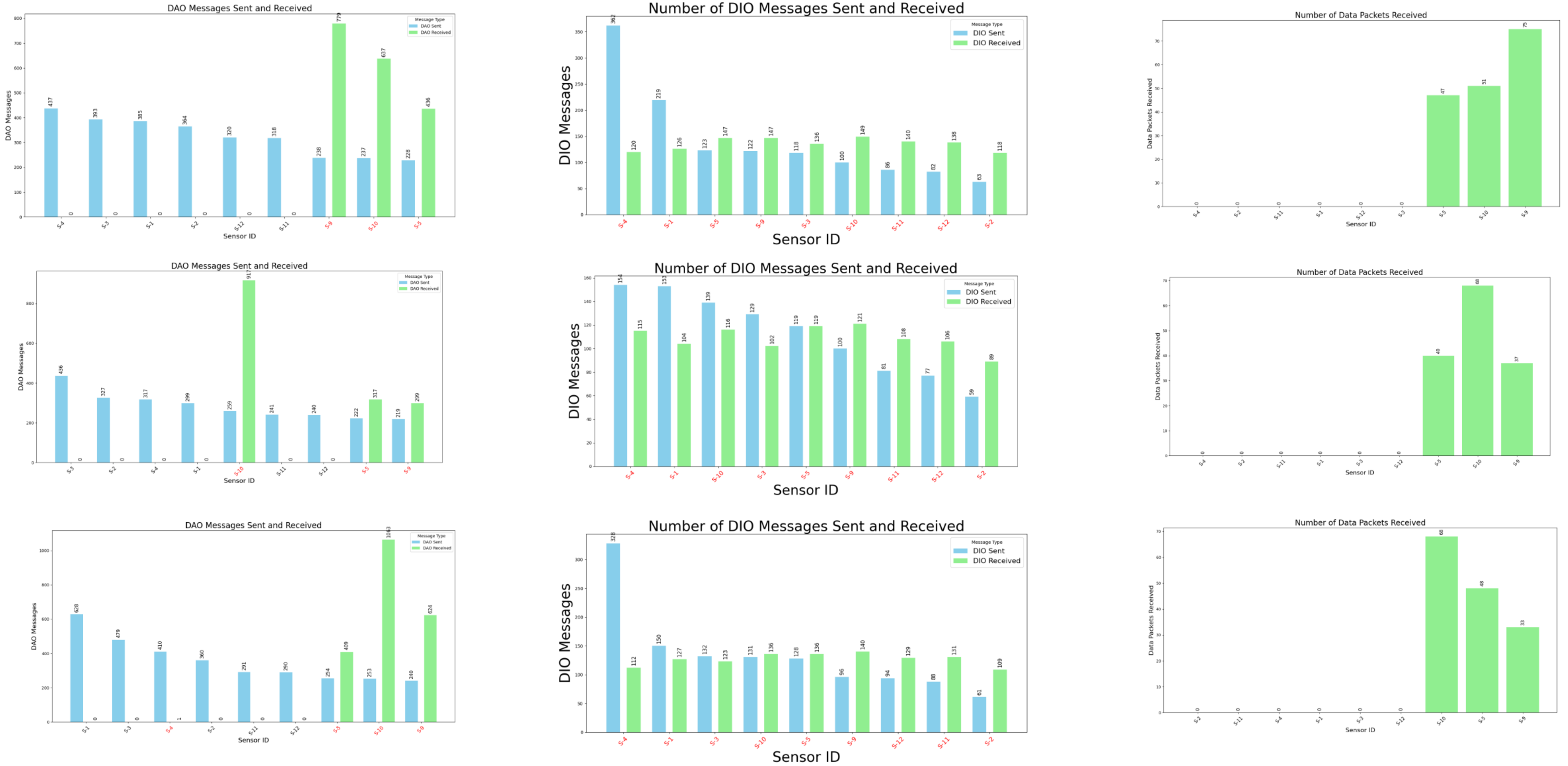
Internet

Malicious nodes

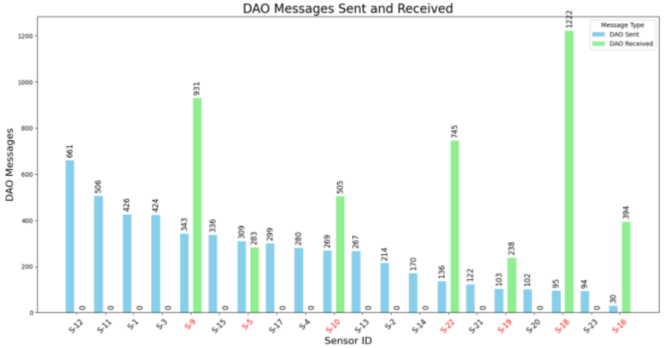
Malicious node

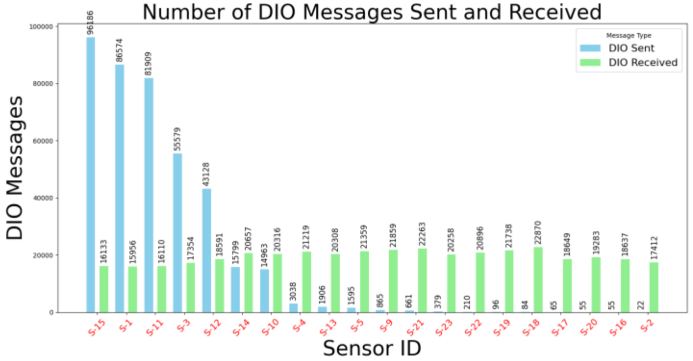
The network topology in IoT using RPL Protocol includes 3 malicious nodes

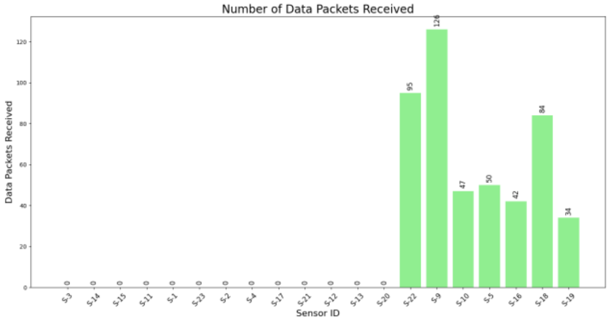
Feature visualization: 3 malicious nodes; 3 runs, each with a different random seed

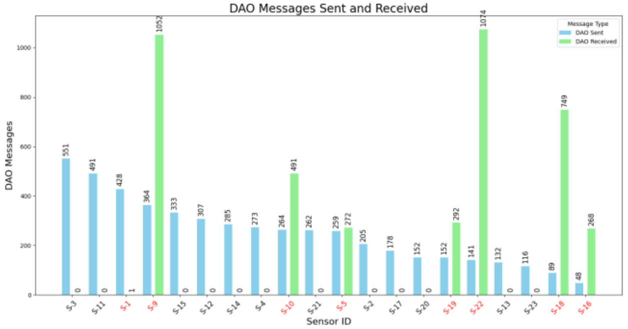


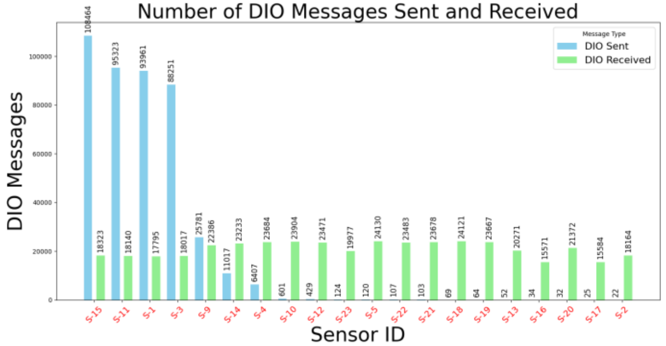
Feature visualization: 7 malicious nodes; 3 runs, each with a different random seed

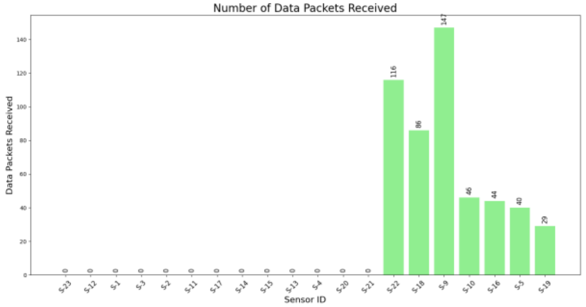


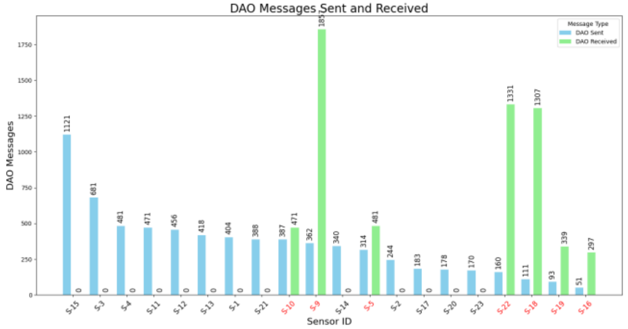


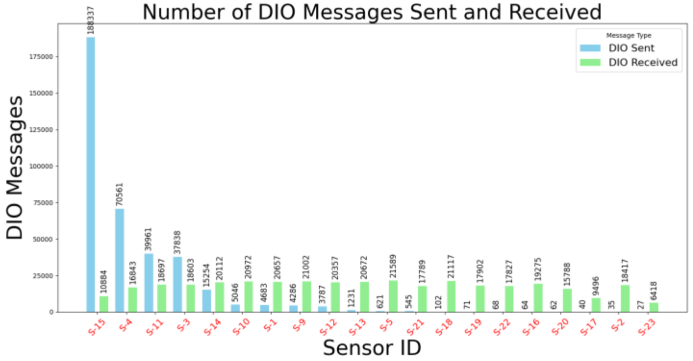


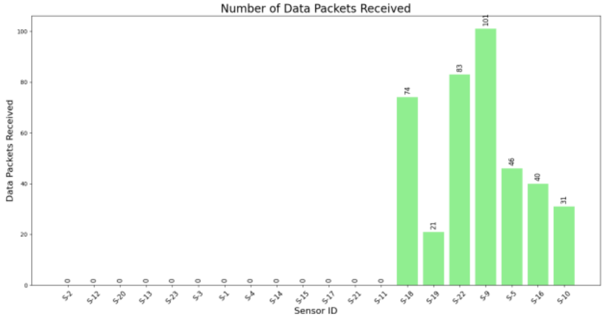




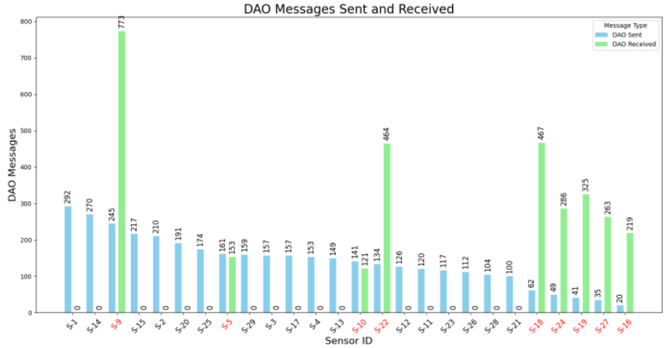


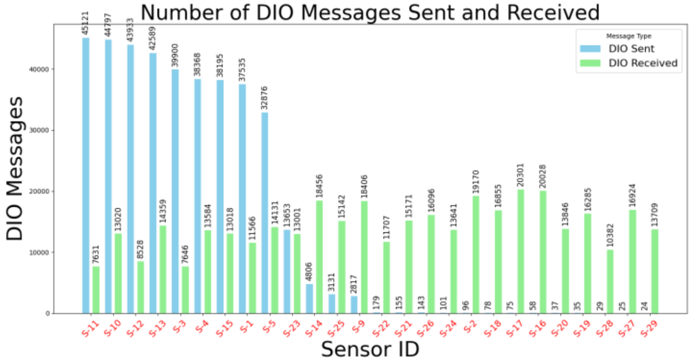


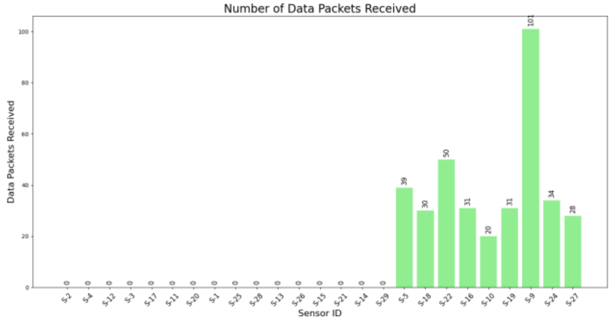


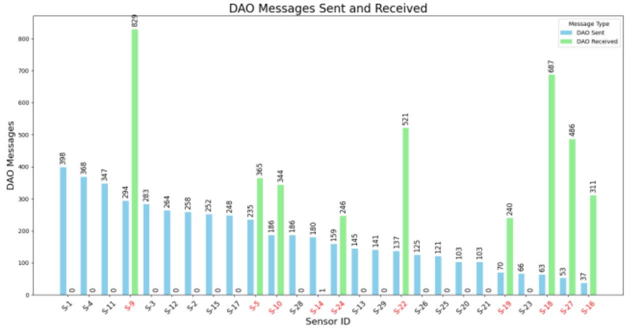


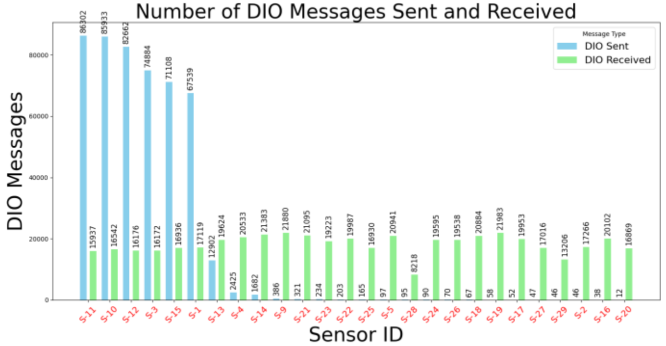
Feature visualization: 9 malicious nodes; 3 runs, each with a different random seed

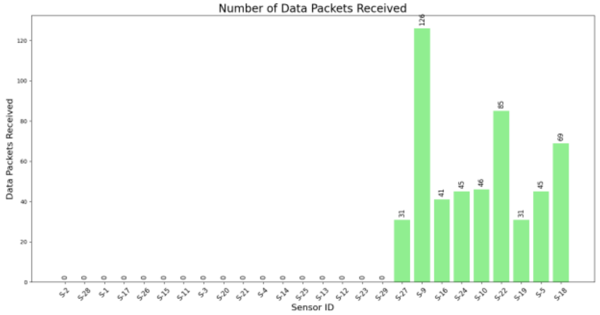

DAO Messages Sent and Received

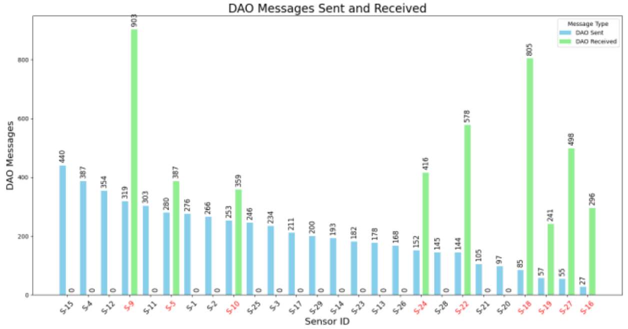

Number of DIO Messages Sent and Received

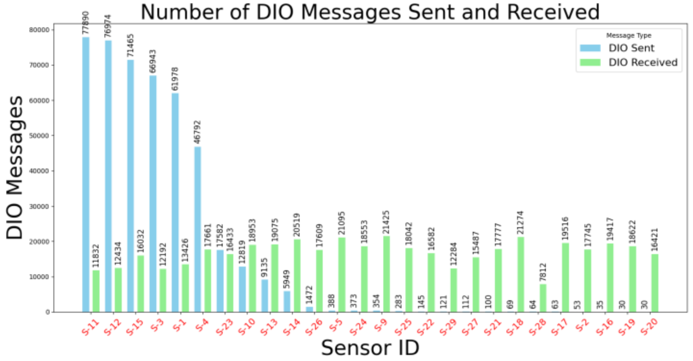

Number of Data Packets Received

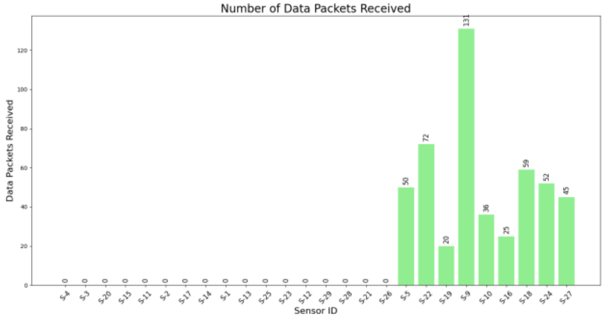

DAO Messages Sent and Received


Number of DIO Messages Sent and Received

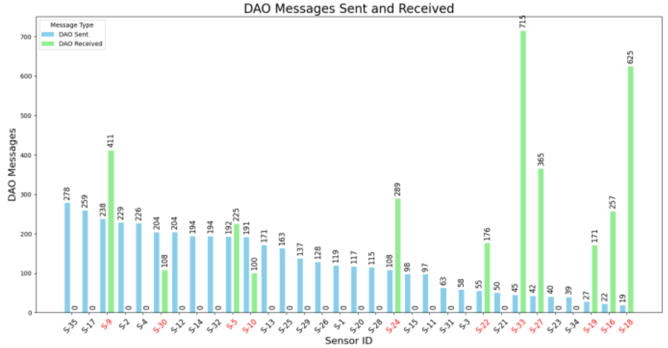

Number of Data Packets Received

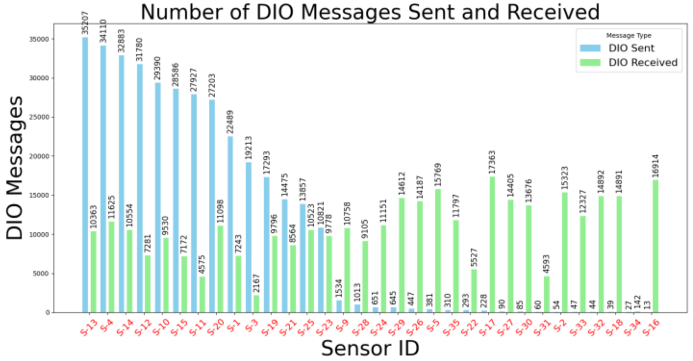

DAO Messages Sent and Received

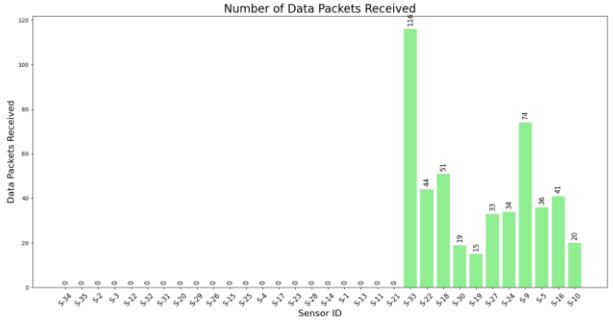

Number of DIO Messages Sent and Received

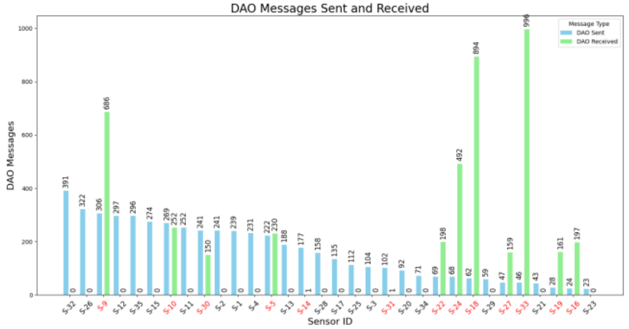

Number of Data Packets Received

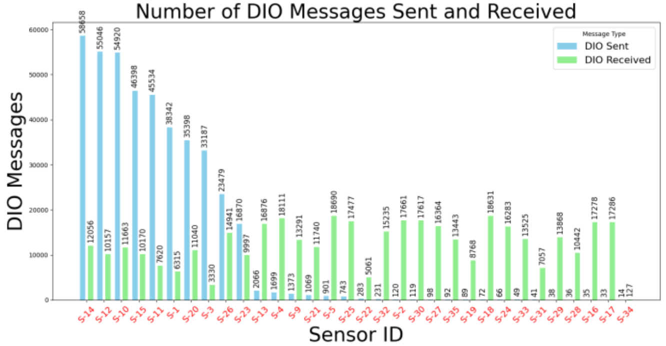
Feature visualization: 11 malicious nodes; 3 runs, each with a different random seed

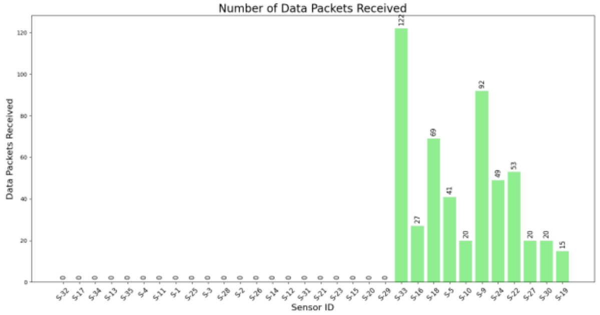


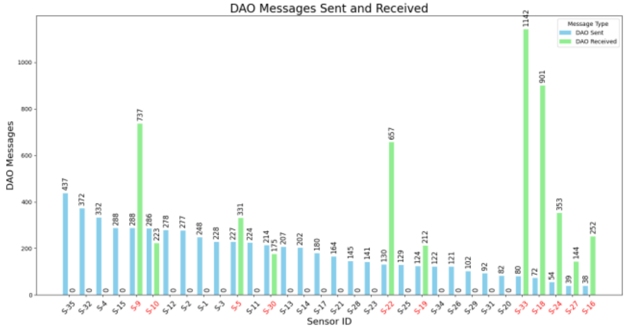


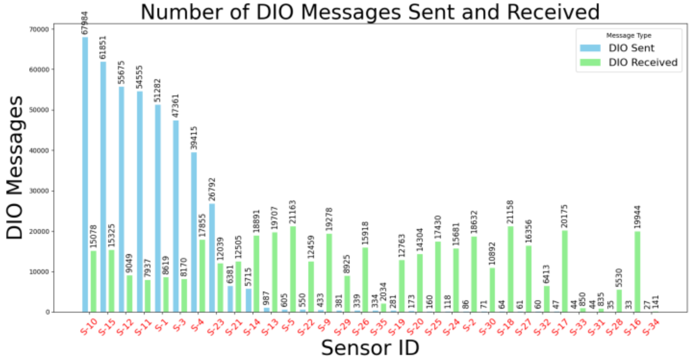


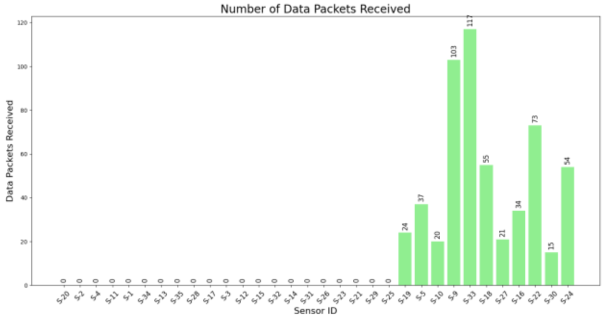




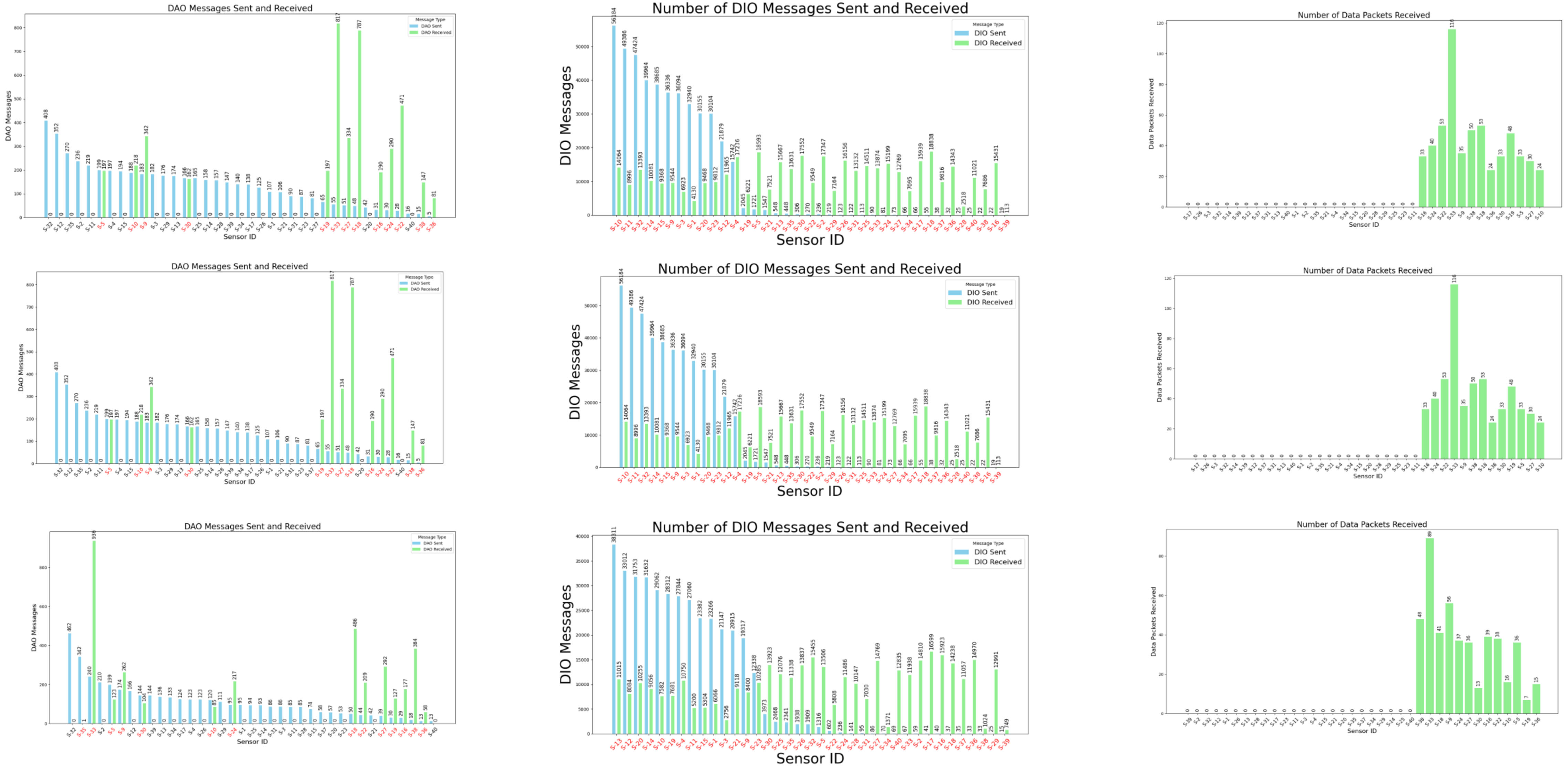




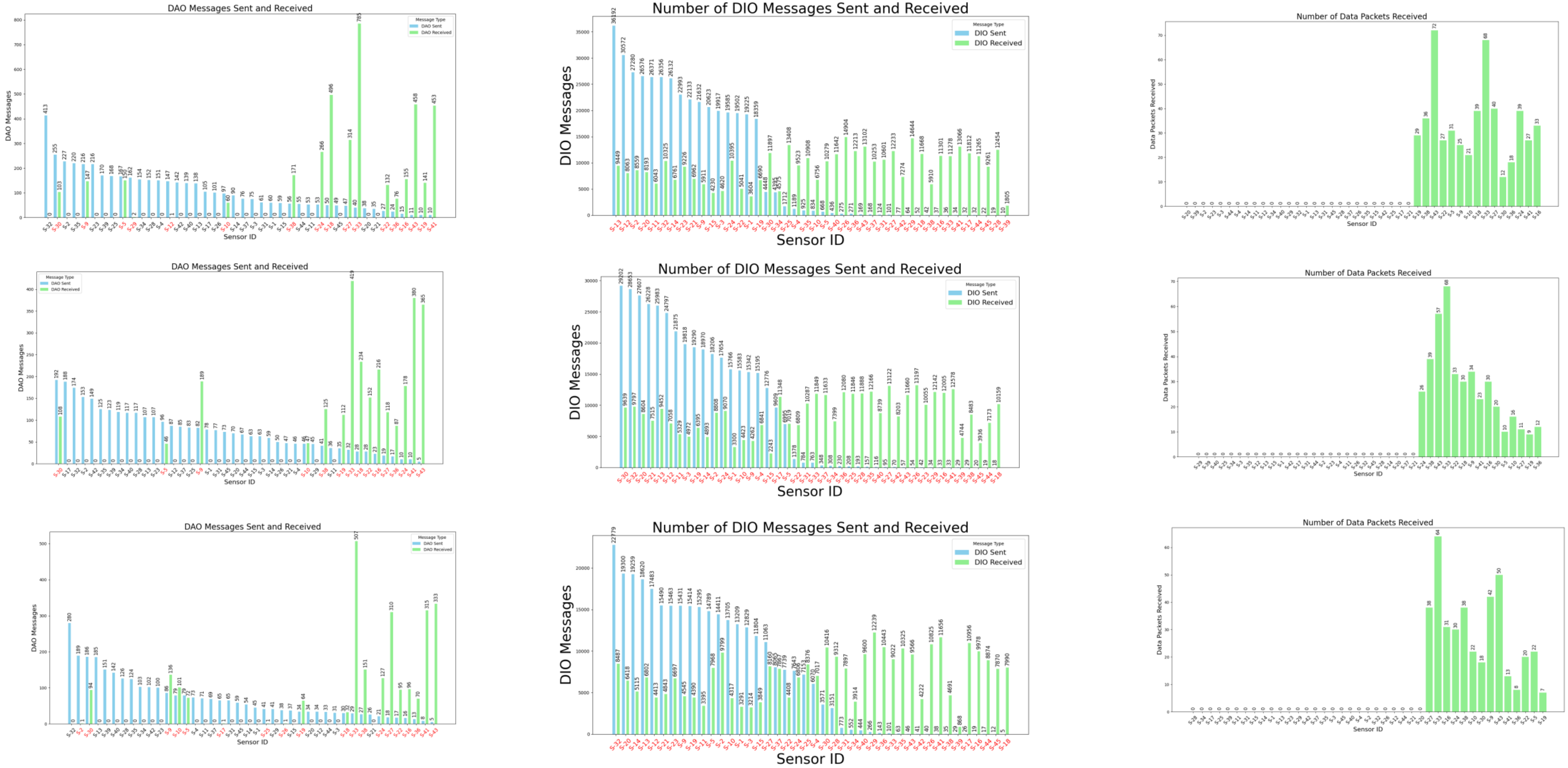




Feature visualization: 13 malicious nodes; 3 runs, each with a different random seed



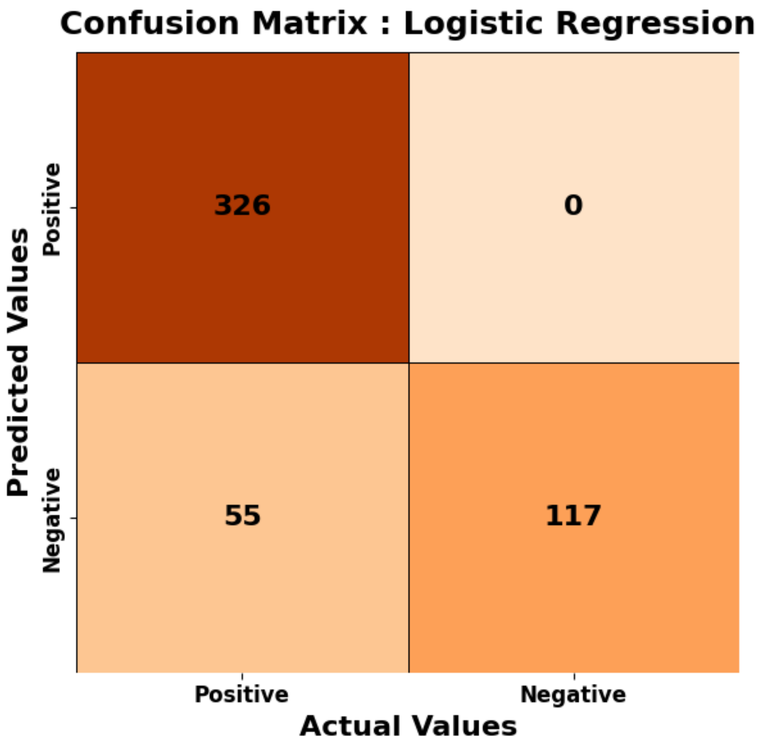
Feature visualization: 15 malicious nodes; 3 runs, each with a different random seed



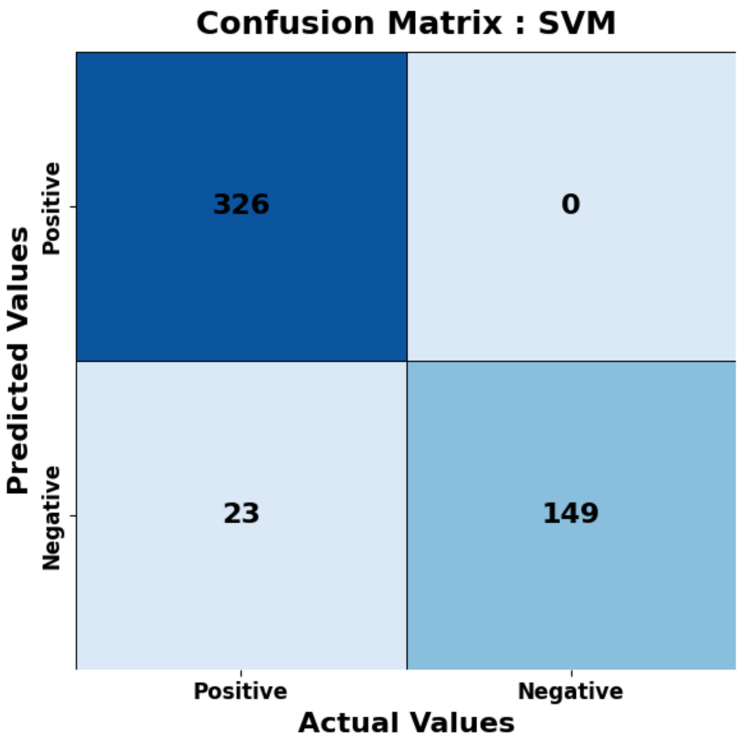
Confusion matrix

- Confusion matrix summarizes the performance of a machine learning model on a set of test data.
    - Displays the number of accurate and inaccurate instances based on the model’s predictions.
    - Used to measure the performance of classification models
- Confusion matrix components:
    - True Positive (TP): Predicted as positive, and it actually is positive.
    - True Negative (TN): Predicted as negative, and it actually is negative.
    - False Positive (FP): Predicted as positive, but it is actually negative.
    - False Negative (FN): Predicted as negative, but it is actually positive
- Performance metrics:
    - Accuracy: The overall correct predictions (TP + TN) divided by the total number of instances.
    - Precision: The number of true positives divided by the total number of predicted positives (TP + FP).
    - Recall: The number of true positives divided by the total number of actual positives (TP + FN).
    - F1 Score: The harmonic mean of precision and recall, providing a balance between the two.

Confusion Matrix: Accuracy, Precision, F1 Score, Recall

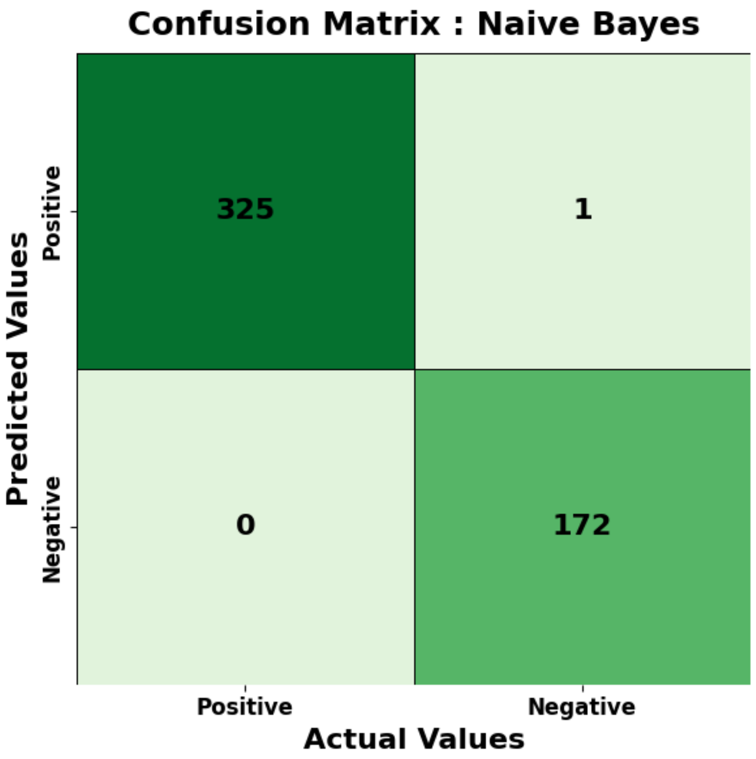


Metric	Value
Accuracy	0.8896
Precision	0.8556
Recall	1.0000
F1 Score	0.9222

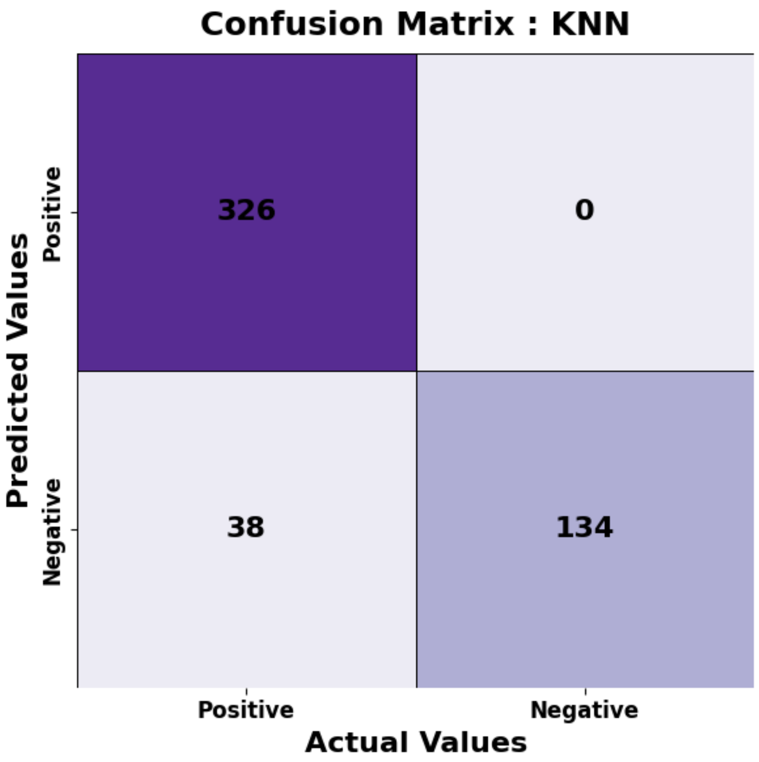


Metric	Value
Accuracy	0.9538
Precision	0.9341
Recall	1.0000
F1 Score	0.9659

Confusion Matrix: Accuracy, Precision, F1 Score, Recall



Metric	Value
Accuracy	0.9980
Precision	1.0000
Recall	0.9969
F1 Score	0.9985



Metric	Value
Accuracy	0.9237
Precision	0.8956
Recall	1.0000
F1 Score	0.9449

Comparison and Future Work

Classifier	True Positives	True Negatives	False Positives	False Negatives	Accuracy	Precision	Recall	F1 Score
Naïve Bayes	324	174	0	0	1.0000	1.0000	1.0000	1.0000
KNN	324	165	9	0	0.9819	0.9730	1.000	0.9863
Logistic Regression	324	155	19	0	0.9618	0.9446	1.000	09715
SVM	324	172	2	0	0.9960	0.9939	1.000	0.9969

Key Observations

- High Precision (>94%): Low false positive rate; malicious classifications are likely correct.
- Near-Perfect Recall (≥99.69%): Classifiers rarely miss malicious nodes.
- Robust F1 Scores (>0.97): Well-balanced performance in identifying threats and avoiding false alarms.

Future Work

- Testing with larger networks
- Exploring other types of IoT network attacks

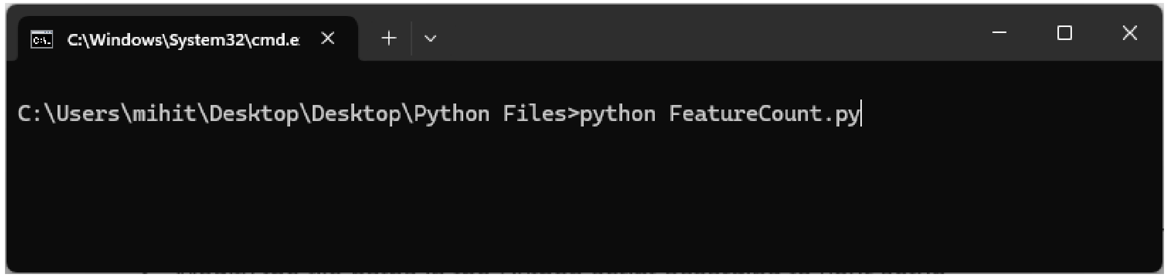
Choukri et al., "RPL rank attack detection using Deep Learning," 2020 *International Conference on Innovation and Intelligence for Informatics.*

Appendix: How-to-Guide

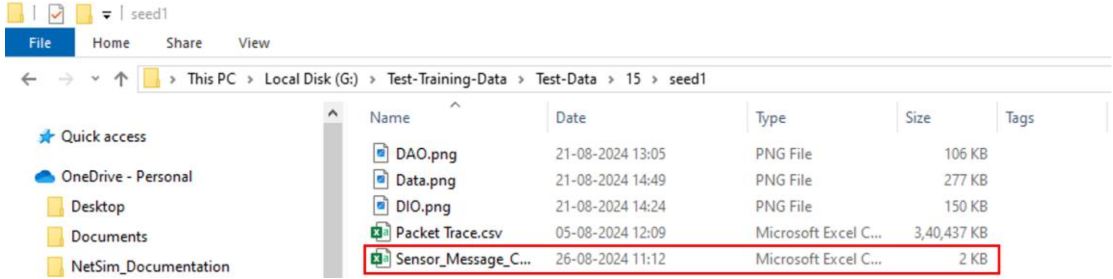
How to classify the data?

To generate an excel file containing the 5 feature message counts for each sensor, follow these steps:

- Modify the file paths in the Python script according to your setup.
- Open the command prompt.
- Navigate to the folder containing the Python script.
- Run the FeatureCount.py script to process the packet trace file and generate the Excel file.
- You can place the Python script anywhere, as long as the file paths are correctly set to locate the necessary data files, including the test and training data scenarios that contain the packet trace files.
- After running the script, it will generate the Sensor_Message_Count.csv file in each folder containing a packet trace.



```
C:\Users\mihit\Desktop\Desktop\Python Files>python FeatureCount.py
```

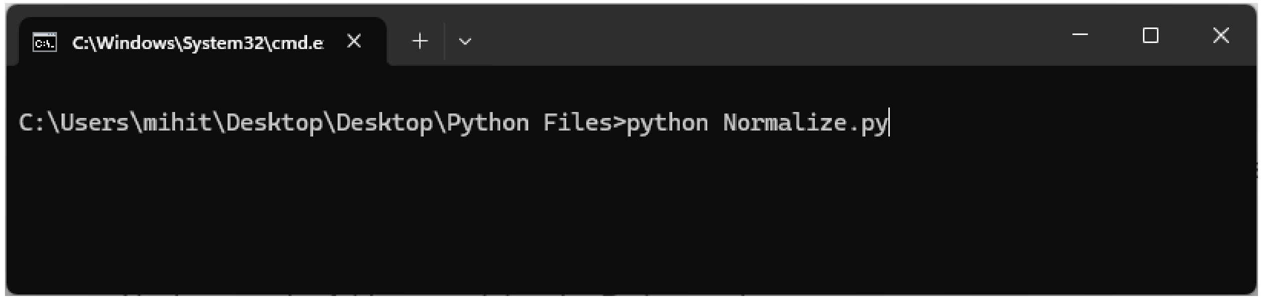


Name	Date	Type	Size	Tags
DAO.png	21-08-2024 13:05	PNG File	106 KB	
Data.png	21-08-2024 14:49	PNG File	277 KB	
DIO.png	21-08-2024 14:24	PNG File	150 KB	
Packet Trace.csv	05-08-2024 12:09	Microsoft Excel C...	3,40,437 KB	
Sensor_Message_C...	26-08-2024 11:12	Microsoft Excel C...	2 KB	

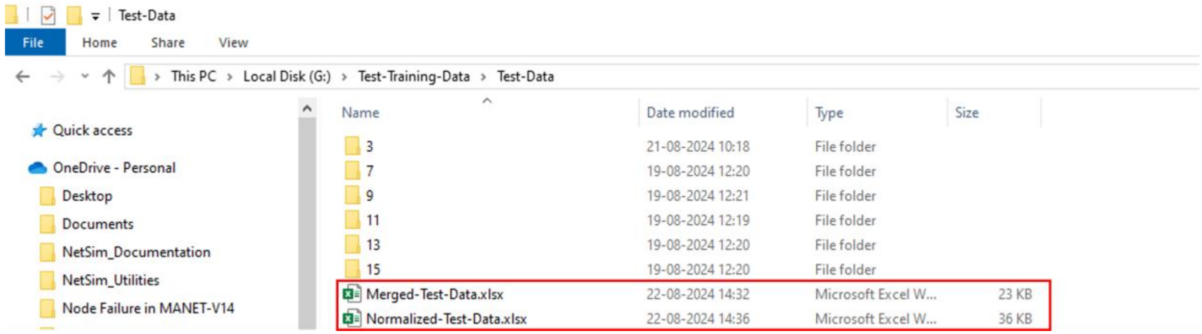
How to Normalize the data?

To generate an excel file containing the normalization of 5 feature message counts for each sensor, follow these steps:

- Modify the file paths in the Python script according to your setup.
- Open the command prompt.
- Navigate to the folder containing the Python script.
- Run the Normalize.py script to process the Sensor_Message_Counts file and generate the normalized Excel file.
- You can place the Python script anywhere, as long as the file paths are correctly set to locate the necessary data files, including the test and training data scenarios that contain the Sensor_Message_Counts.csv.
- After running the script, it will generate the merged and normalized data file in the folder containing the Python script.



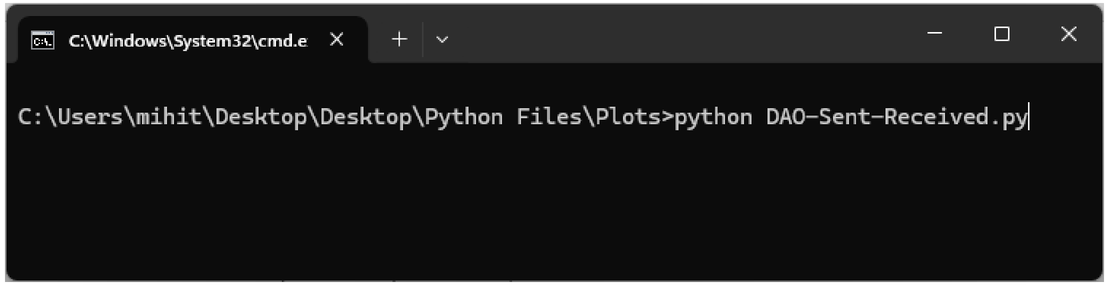
```
C:\Users\mihit\Desktop\Desktop\Python Files>python Normalize.py
```



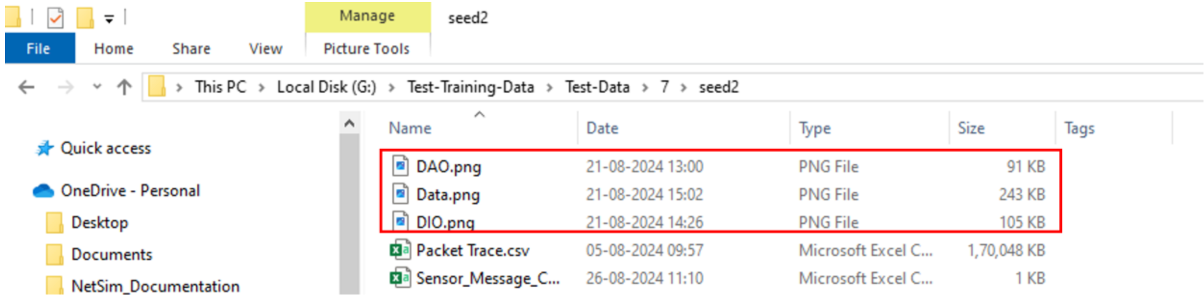
Name	Date modified	Type	Size
3	21-08-2024 10:18	File folder	
7	19-08-2024 12:20	File folder	
9	19-08-2024 12:21	File folder	
11	19-08-2024 12:19	File folder	
13	19-08-2024 12:20	File folder	
15	19-08-2024 12:20	File folder	
Merged-Test-Data.xlsx	22-08-2024 14:32	Microsoft Excel W...	23 KB
Normalized-Test-Data.xlsx	22-08-2024 14:36	Microsoft Excel W...	36 KB

How to generate plots?

- Download the workspace and place it in your desired location.
- The Python scripts can be placed anywhere, as long as the file paths are correctly set.
- Modify the file paths in the Python scripts according to your setup.
- Open the command prompt.
- Navigate to the folder containing the plot-related Python scripts.
- Run the DAO, DIO, and Packet Received scripts present in the plots folder to process the packet trace files and generate the plots.
- Ensure the necessary packet trace files are accessible based on the paths defined in the scripts.



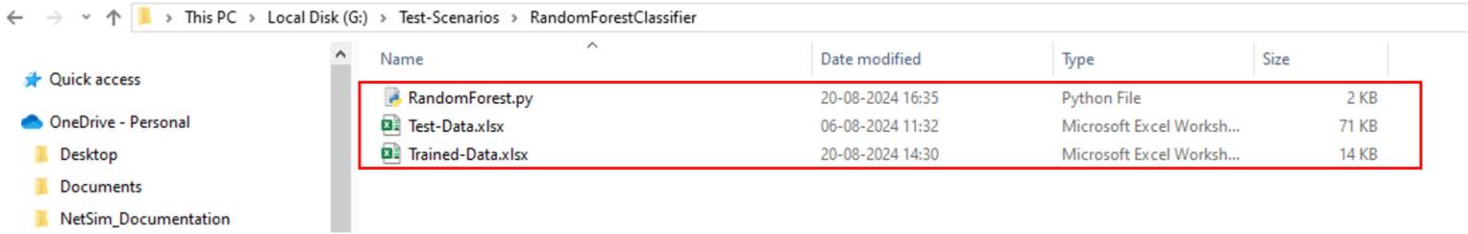
```
C:\Users\mihit\Desktop\Desktop\Python Files\Plots>python DAO-Sent-Received.py
```



Name	Date	Type	Size	Tags
DAO.png	21-08-2024 13:00	PNG File	91 KB	
Data.png	21-08-2024 15:02	PNG File	243 KB	
DIO.png	21-08-2024 14:26	PNG File	105 KB	
Packet Trace.csv	05-08-2024 09:57	Microsoft Excel C...	1,70,048 KB	
Sensor_Message_C...	26-08-2024 11:10	Microsoft Excel C...	1 KB	

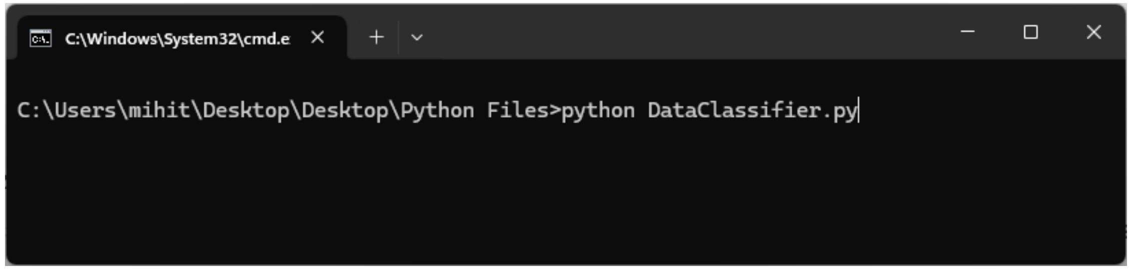
How to run the classifiers?

- Along with normalizing the data, the process will generate `test_data` and `training_data` files.
- Run the `Data Classifier.py` script using the trained data by providing the file paths for both the `Trained-Data` and `Test-Data`.
- Modify the file paths in the `Data Classifier.py` script according to your setup.
- Open the command prompt and run the script as shown below:



This PC › Local Disk (G:) › Test-Scenarios › RandomForestClassifier

Name	Date modified	Type	Size
RandomForest.py	20-08-2024 16:35	Python File	2 KB
Test-Data.xlsx	06-08-2024 11:32	Microsoft Excel Worksh...	71 KB
Trained-Data.xlsx	20-08-2024 14:30	Microsoft Excel Worksh...	14 KB

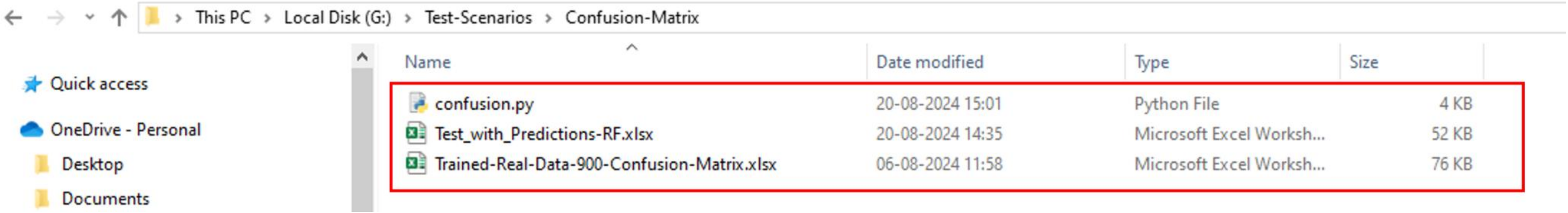


```
C:\Users\mihit\Desktop\Desktop\Python Files>python DataClassifier.py
```

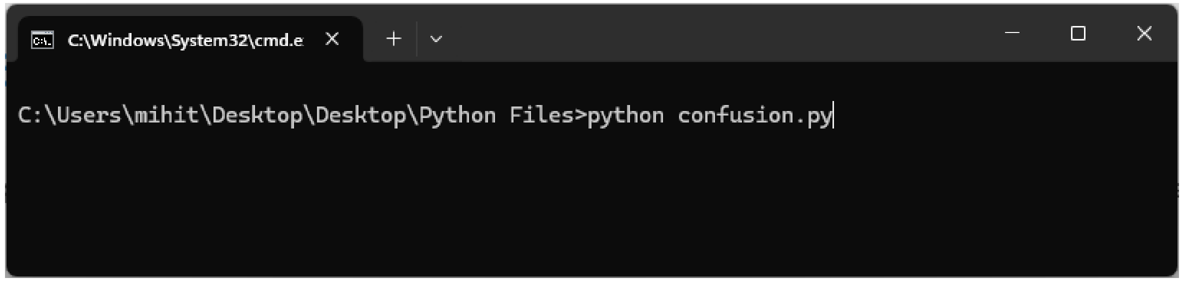
The Python script generates six sets of predicted labels and uses this data to create confusion matrices.

How to get the confusion matrix?

- After obtaining the predicted labels from the classifiers, use the predicted label Excel files along with the real training data.
- Place the predicted label Excel files and the `confusion.py` script in the same folder.
- Modify the file paths in the `confusion.py` script according to your setup.
- Open the command prompt and navigate to the folder containing the `confusion.py` script.
- Run the script as shown below:
- The script will generate a confusion matrix for each classifier based on the data provided.


This PC › Local Disk (G:) › Test-Scenarios › Confusion-Matrix
Quick access
OneDrive - Personal
Desktop
Documents

Name	Date modified	Type	Size
confusion.py	20-08-2024 15:01	Python File	4 KB
Test_with_Predictions-RF.xlsx	20-08-2024 14:35	Microsoft Excel Worksh...	52 KB
Trained-Real-Data-900-Confusion-Matrix.xlsx	06-08-2024 11:58	Microsoft Excel Worksh...	76 KB



```
C:\Users\mihit\Desktop\Desktop\Python Files>python confusion.py
```