NetSim IoT

ML based classifier to detect attacks in RPL based IoT networks

27Aug2024

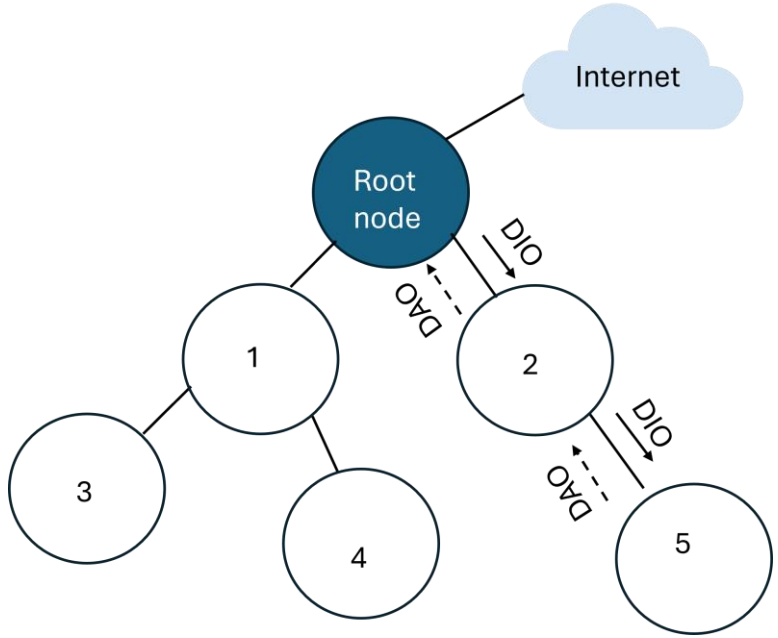
Applicable Release: NetSim v14.1 or higher
Applicable Version(s): NetSim Standard
Project download link: https://github.com/NetSim-TETCOS/ML-Classifier-to-detect-network-attack-in-IoT-v14.1/archive/refs/heads/main.zip
The URL has the exported NetSim scenario for the examples used in this document and 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 stands for Routing Protocol for Low-Power and Lossy Networks.
- It is a network layer IPv6-based routing protocol designed specifically for Low-Power and Lossy Networks (LLNs).
- The motivation of RPL is to construct a Directed Acyclic Graph (DAG) rooted at the sink, which will minimize the cost of reaching the sink from any node in the network as per the Objective Function (OF).
- Terminology used in RPL:
  - DAG (Directed Acyclic Graph): A directed graph having the property that all edges are oriented in such a way that no cycles exist.
  - DAG root: A DAG root is a node within the DAG that has no outgoing edge.
  - DODAG ID: Each DODAG has an IPV6 ID. This ID is given to its root only.
  - Rank: Rank defines the individual node positions with respect 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

- The OF can be to minimize a particular metric, such as hop count or ETX (Expected Transmission count).
- NetSim implementation: OF prioritizes routes with the best link quality.
- The link quality in RPL depends on the received power and the receiver sensitivity of the nodes in the network.
- In NetSim, the Link Quality between any two nodes is calculated as follows
  - In both directions, calculate $\left(1-\frac{p}{r_s}\right)$, where p is the received power and r_s is the receiver sensitivity, both measured in dBm. Let us denote them as Transmit link quality, TLq and receive link quality RLq.
  - Then link quality, $Lq = \frac{TLq+RLq}{2}$

Rank calculation in NetSim

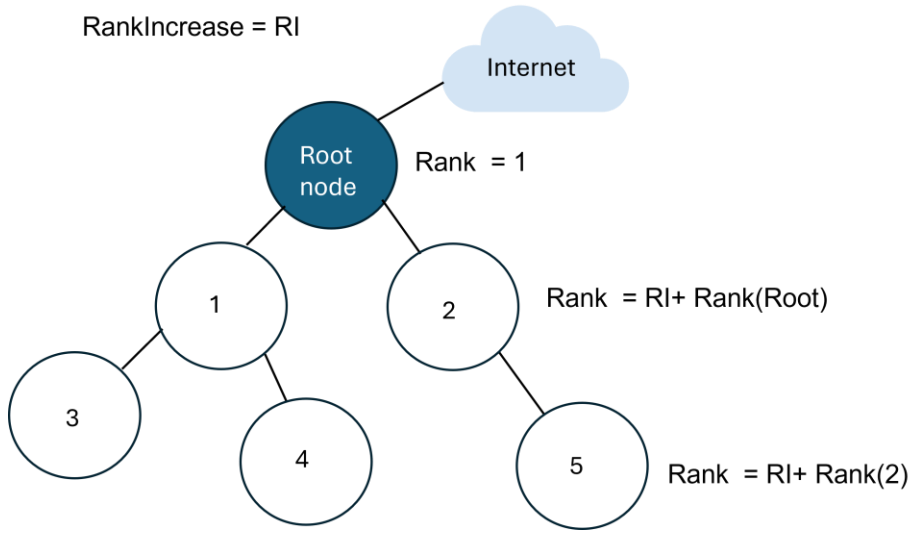
- The Rank of a node is a scalar representation of the location of that node within a DODAG.
- The rank is a measure in some sense of the distance of the node from the root. It monotonically increases as we move away from the root.
- Rank is used to avoid and detect loops.
- The rank calculation is based on the objective function defined.
- Root node has a Rank 1. This root node in IoT is also the border router.
- The rank increase depends on the link quality and is given by the expression

$$RankIncrease(RI)$$

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

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

where, MaxIncrement = 16 and MinIncrement = 1 as per RFC 6550, and Lq is the Link quality.



DODAG Topology with Node Ranks in an IoT Network

Rank calculations in NetSim

- The rank of the root node is set to 1, and the parent node of S2 is the root node.
- The received power at S2 can be calculated using the following formula,

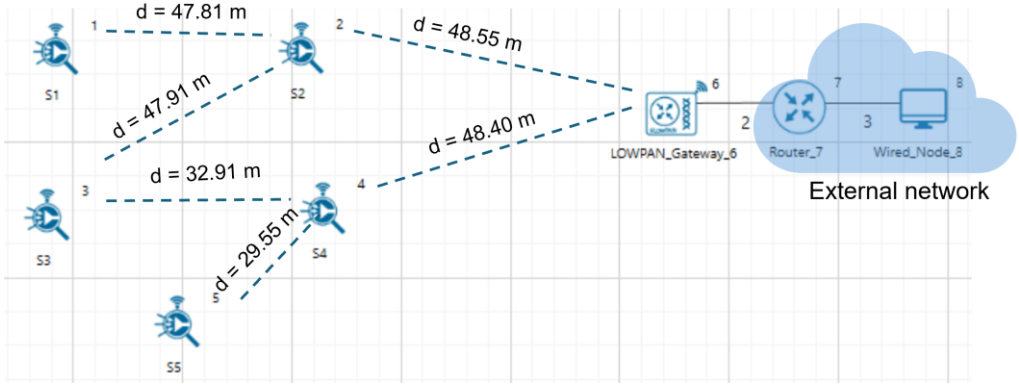
$$P_r(dBm) = P_t + G_t + G_r + 20\ \log_{10}\left(\frac{\lambda}{4\pi d0}\right) + 10 \times \eta \times \log_{10}\left(\frac{d_0}{d}\right)$$

$$P_r(dBm) = 1 + 0 + 0 + 20\log_{10}\left(\frac{0.125}{4 \times 3.14 \times 8}\right) + 10 \times 3 \times \log_{10}\left(\frac{8}{48.55}\right)$$

$$P_r(dBm) = -81.86$$

where, $P_t = 1mW$, $d$is the distance between s2 and root node, and is equal to 48.55m,$d_0 = 8, G_t = 0, G_r = 0, \eta = 3,\ \lambda = \frac{c}{f} = 0.125m, f = 2400MHz$

- One way link quality $Lq = \left(1 - \frac{P_t}{r_s}\right) = 1 - \left(\frac{-81.86}{-85}\right) = 1 - 0.963 = 0.036$



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

$$Lq = \frac{TLq + RLq}{2} = \frac{0.036 + 0.036}{2} = 0.036$$

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

Where, Lq = 0.036, MaxIncrement = 16, and MinIncrement = 1

$$RankIncrease = Floor((16 - 1) \times (1 - 0.036)^2 + 1) = ((15 \times 0.929) + 1) = floor(14.939) = 14$$

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

$$Rank = 14 + 1 = 15$$

- 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

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

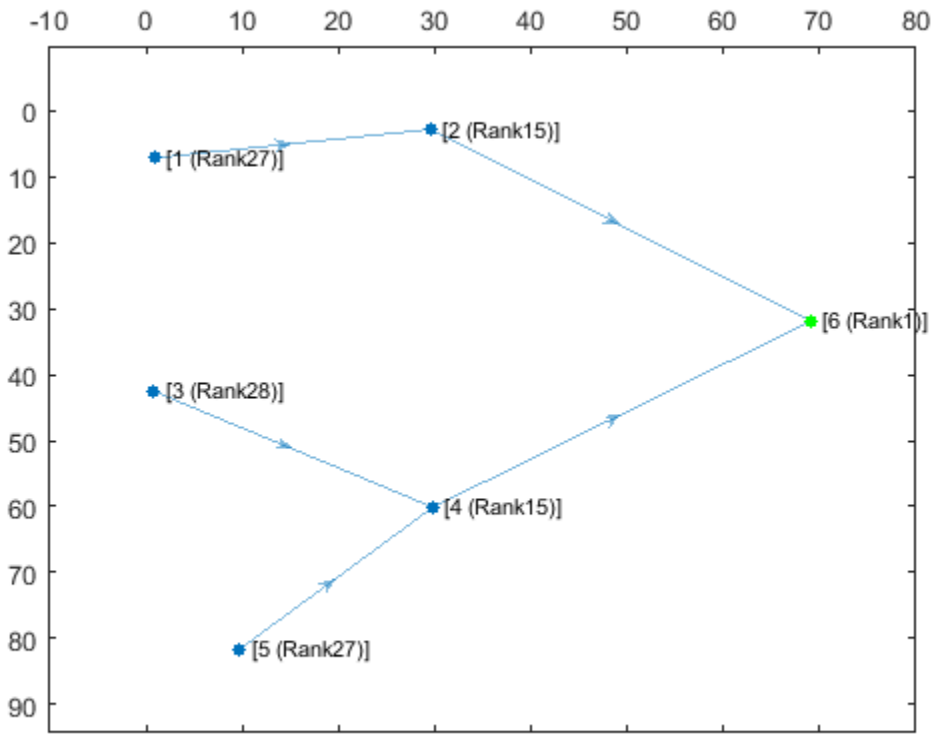
$$RankIncrease = Floor((16 - 1) \times (1 - 0.137)^2 + 1)$$

$$= ((15 \times 0.744) + 1) = floor(12.16) = 12$$

- 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

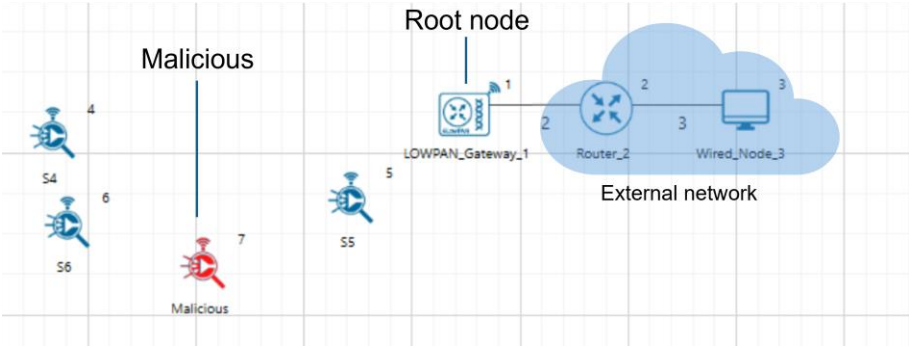
Rank attack in RPL using NetSim

- In RPL, the transmitter broadcasts the DIO during DODAG formation.
- Upon receiving the DIO from the transmitter, the receiver updates its parent list, sibling list, and rank, then sends a DAO message with route information.
- A malicious node, upon receiving the DIO message, does not update its rank; instead, it always advertises a fake rank.
- Other nodes, upon hearing the malicious node's DIO message, update their rank according to the fake rank.
- After the formation of the DODAG, if the node transmitting the packet has a malicious node as the preferred parent, it transmits the packet to it. However, the malicious node, instead of forwarding the packet to its parent, simply drops the packet, resulting in zero throughput.

M. M. k. Abdul Rehman, “Rank Attack using Objective Function in RPL for Low Power and Lossy Networks,” 2016 IEEE.

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

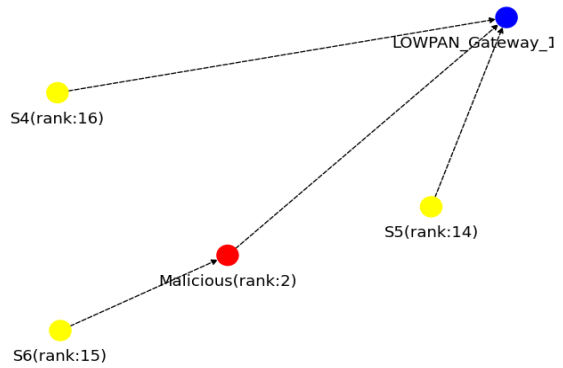
- The results can be observed in the Results window, showing that the network has zero throughput.
- Users can also observe Packet trace that after Sensor 7 receives packets, it does not forward them, resulting in no data packet transmission from Sensor 7.
- Additionally, users can generate the DODAG 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.

	A	B	C	D	E	F	G	H
	PACKET_ID	SEGMENT_ID	PACKET_TYPE	CONTROL_PACKET_TYPE/APP_NAME	SOURCE_ID	DESTINATION_ID	TRANSMITTER_ID	RECEIVER_ID
2	2	0	Sensing	App2_SENSOR_APP	SENSOR-6	NODE-3	SENSOR-6	SENSOR-7
3	2	0	Sensing	App1_SENSOR_APP	SENSOR-4	NODE-3	SENSOR-4	SENSOR-7
4	3	0	Sensing	App1_SENSOR_APP	SENSOR-4	NODE-3	SENSOR-4	SENSOR-7
5	3	0	Sensing	App2_SENSOR_APP	SENSOR-6	NODE-3	SENSOR-6	SENSOR-7
6	4	0	Sensing	App1_SENSOR_APP	SENSOR-4	NODE-3	SENSOR-4	SENSOR-7
7	4	0	Sensing	App2_SENSOR_APP	SENSOR-6	NODE-3	SENSOR-6	SENSOR-7
8	5	0	Sensing	App1_SENSOR_APP	SENSOR-4	NODE-3	SENSOR-4	SENSOR-7
9	5	0	Sensing	App2_SENSOR_APP	SENSOR-6	NODE-3	SENSOR-6	SENSOR-7
10	6	0	Sensing	App1_SENSOR_APP	SENSOR-4	NODE-3	SENSOR-4	SENSOR-7
11	6	0	Sensing	App2_SENSOR_APP	SENSOR-6	NODE-3	SENSOR-6	SENSOR-7
12	7	0	Sensing	App1_SENSOR_APP	SENSOR-4	NODE-3	SENSOR-4	SENSOR-7
13	7	0	Sensing	App2_SENSOR_APP	SENSOR-6	NODE-3	SENSOR-6	SENSOR-7
14	8	0	Sensing	App2_SENSOR_APP	SENSOR-6	NODE-3	SENSOR-6	SENSOR-7
15	9	0	Sensing	App2_SENSOR_APP	SENSOR-6	NODE-3	SENSOR-6	SENSOR-7
16	9	0	Sensing	App1_SENSOR_APP	SENSOR-4	NODE-3	SENSOR-4	SENSOR-7
17	10	0	Sensing	App1_SENSOR_APP	SENSOR-4	NODE-3	SENSOR-4	SENSOR-7
18	11	0	Sensing	App1_SENSOR_APP	SENSOR-4	NODE-3	SENSOR-4	SENSOR-7
19	11	0	Sensing	App2_SENSOR_APP	SENSOR-6	NODE-3	SENSOR-6	SENSOR-7
20	12	0	Sensing	App2_SENSOR_APP	SENSOR-6	NODE-3	SENSOR-6	SENSOR-7
21	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.

Attack scenarios with malicious nodes - Training data

- We created 8 scenarios with different numbers of nodes: 6, 14, 15, 18, 23, 29, 34 and 39. In each scenario, there were 2, 4, 5, 6, 8, 10, 12 and 14 malicious nodes, respectively.
    - The simulations were run with 3 random seeds.
- We then enabled packet trace for all scenarios and used a python script to calculate the number of DAO, DIO, and data packets received by each sensor from packet trace.
- The features considered for classifying the data are:
    - DAO Sent
    - DAO Received
    - DIO Sent
    - DIO Received
    - Data Packets Received

Data processing

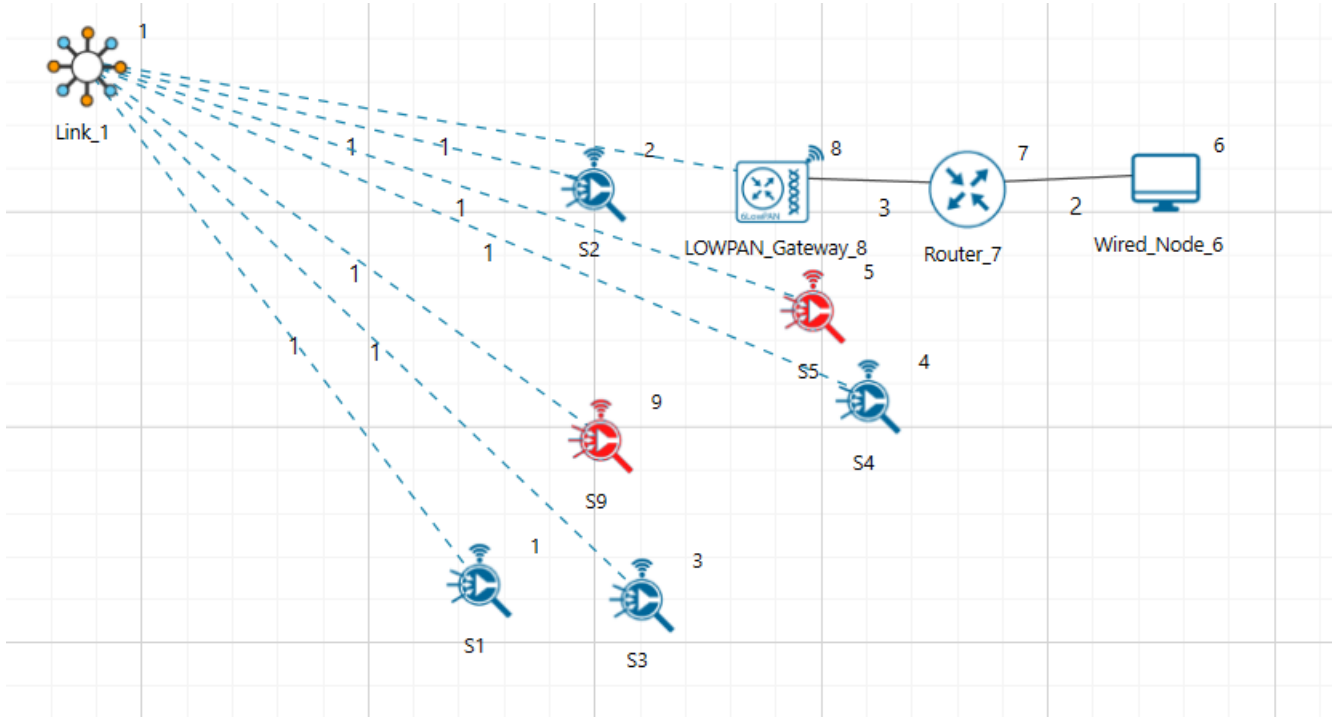
- Extracted data from packet trace into an Excel sheet using python script.
- There were a total of 534 sensors, each with 5 features.
- Feature normalization: For each feature, the maximum value is calculated across all sensors. Each sensor's value is then divided by this maximum value to normalize the data and obtain values between 0 and 1. After obtaining the normalized values, the data is transposed.
- We then label data points as either malicious or non-malicious by assigning 1 for non malicious and 0 for malicious manually.

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

Training-Data

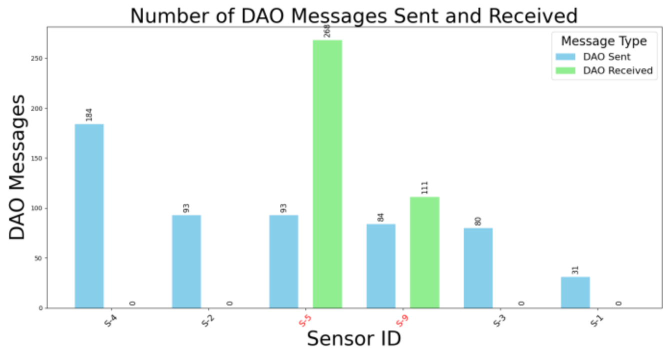
We label the sensors based on the features

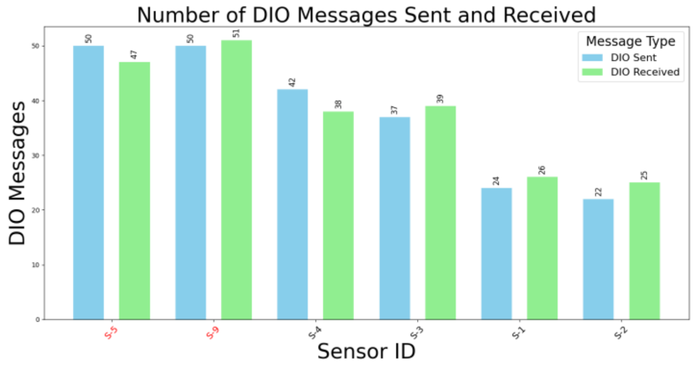
Example Network scenario in NetSim with 2 malicious nodes

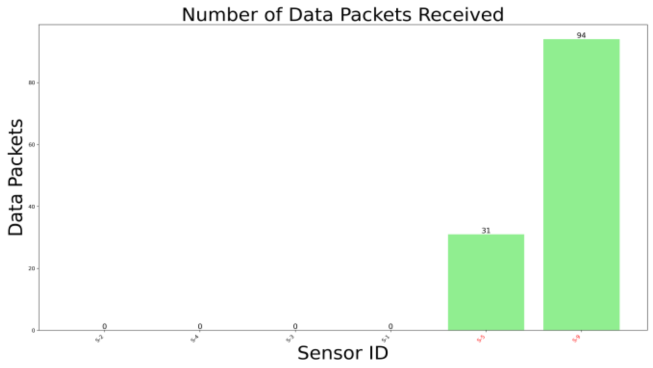


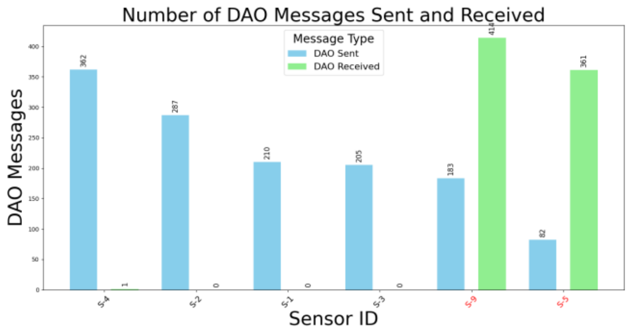
The network topology in IoT using RPL Protocol includes 2 malicious nodes Pathloss Model: Log Distance, Pathloss Exponent: 2

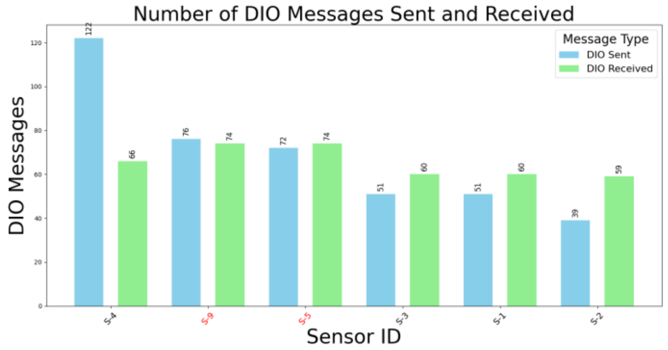
Feature visualization: 2 malicious nodes; 3 runs, each with a different random seed

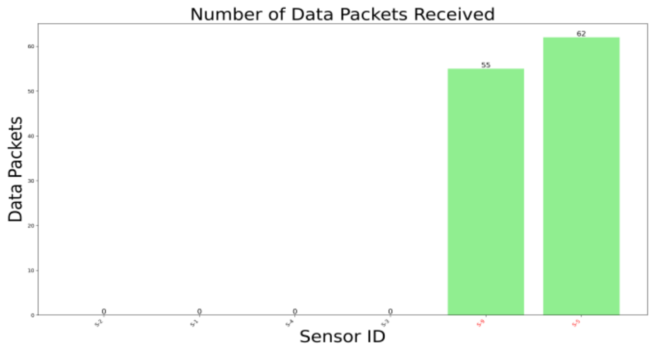


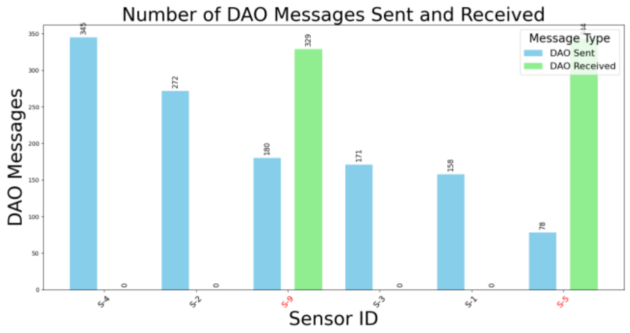


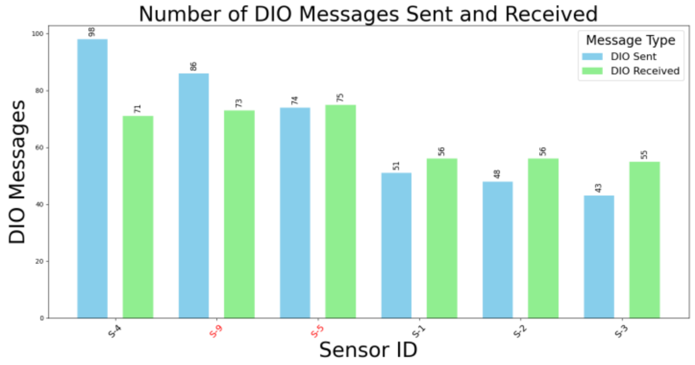


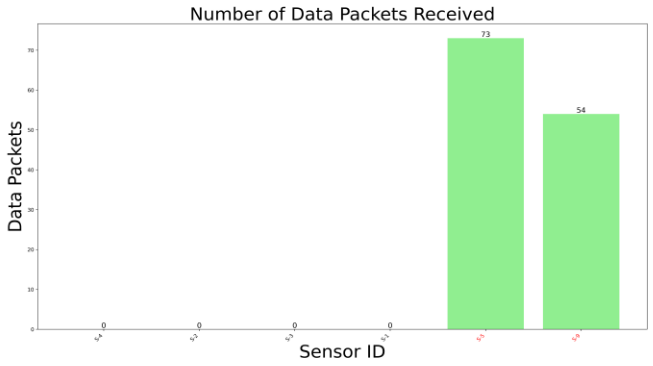




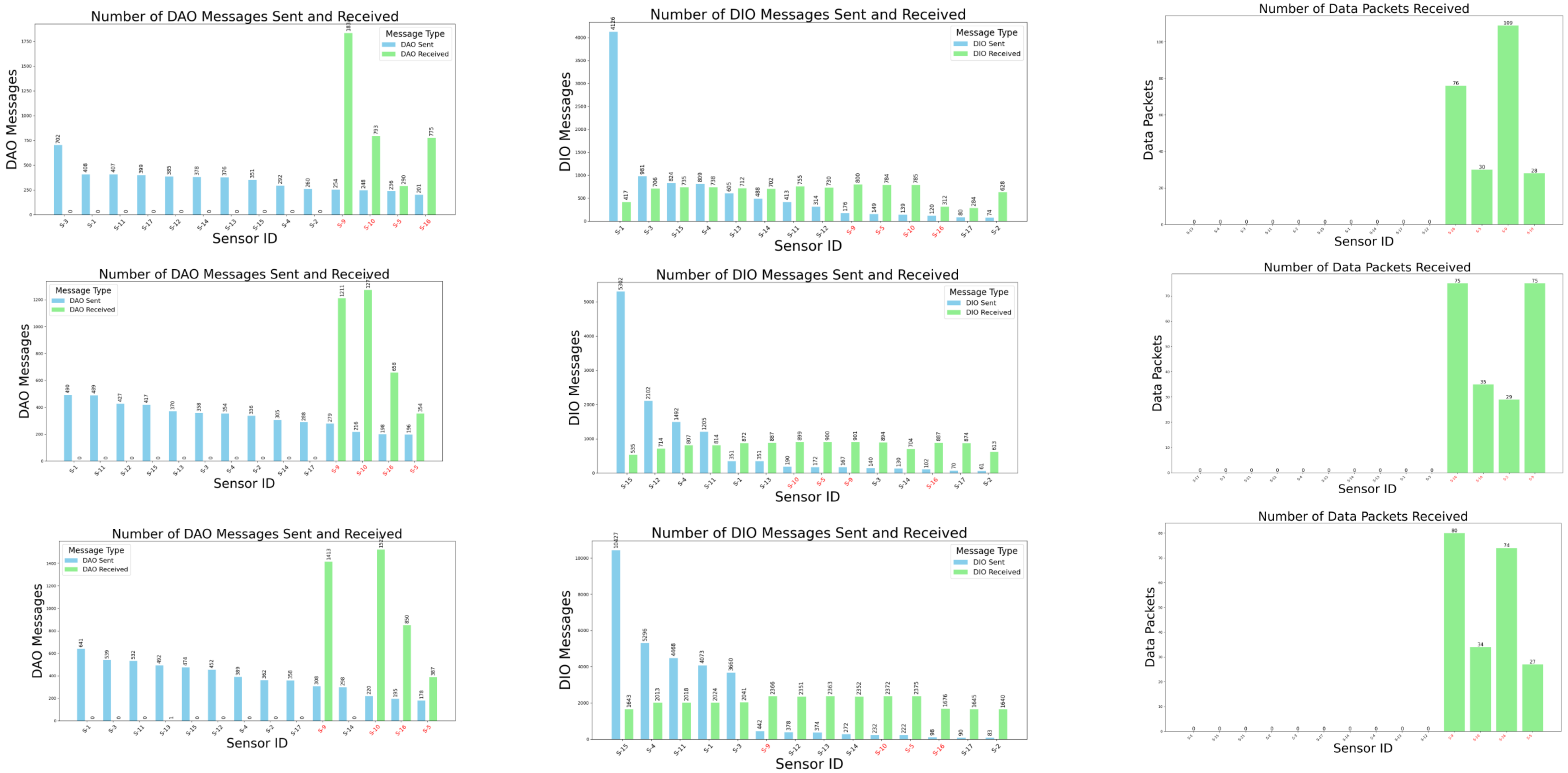




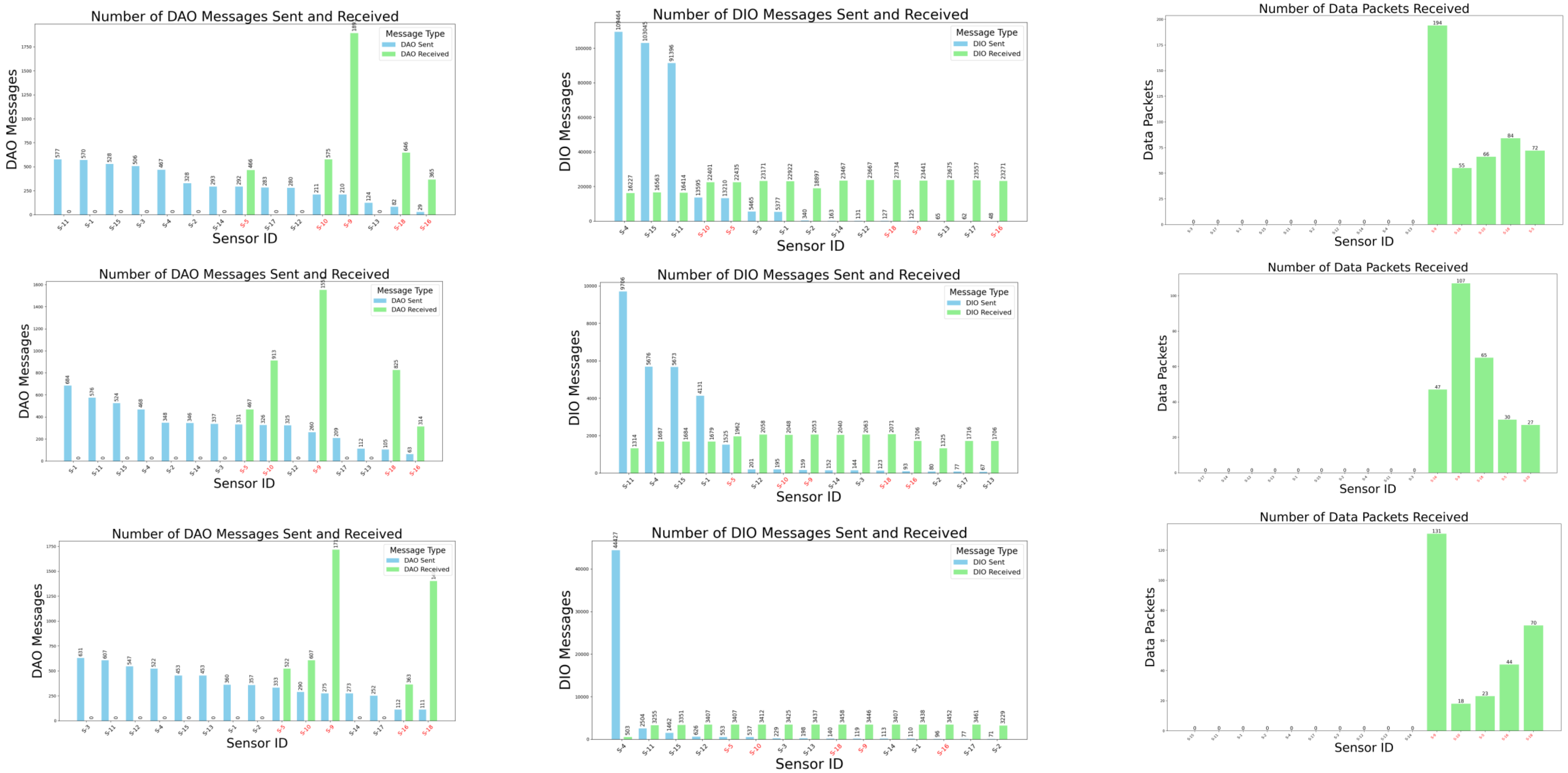




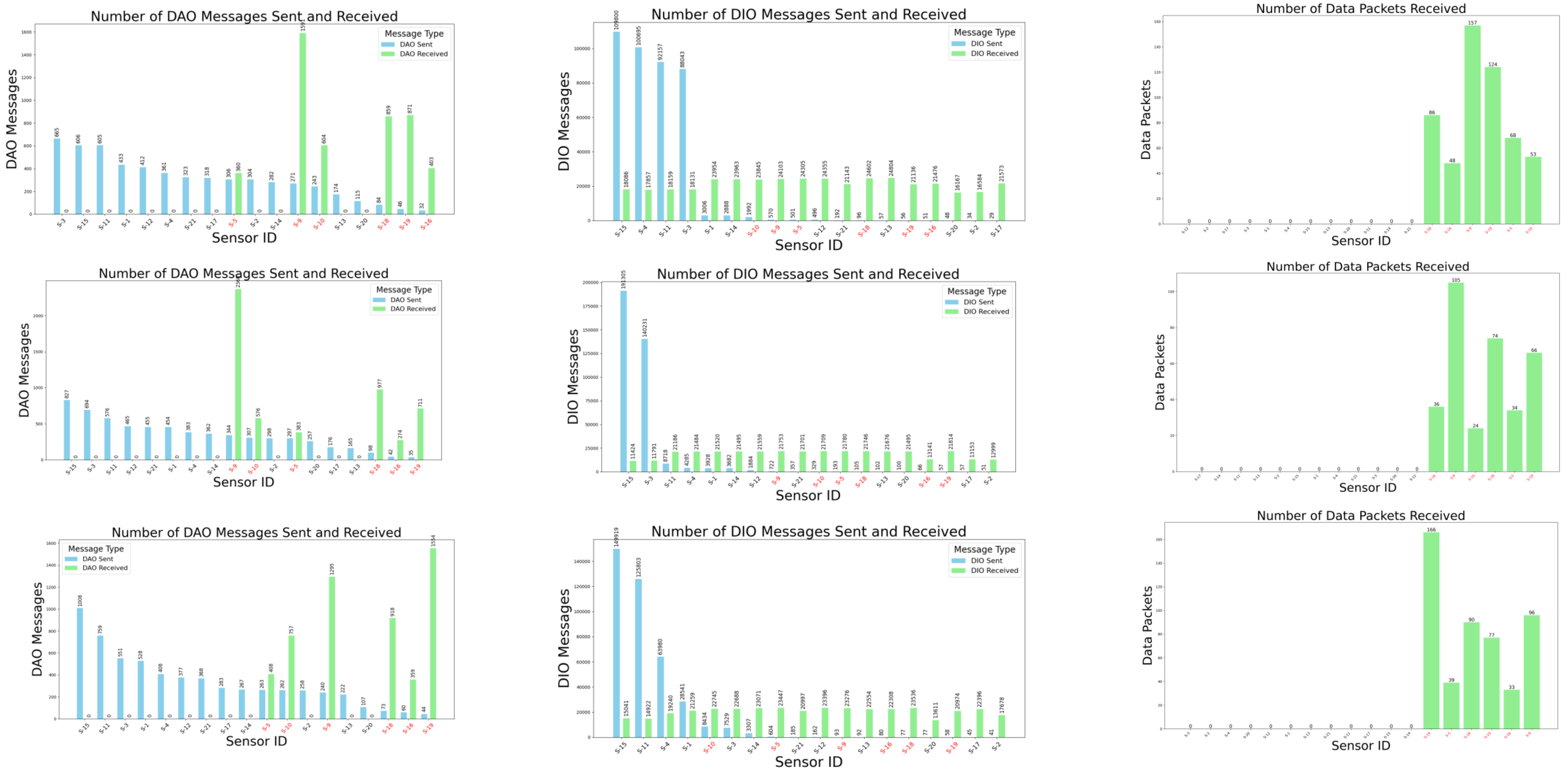
Feature visualization: 4 malicious nodes; 3 runs, each with a different random seed



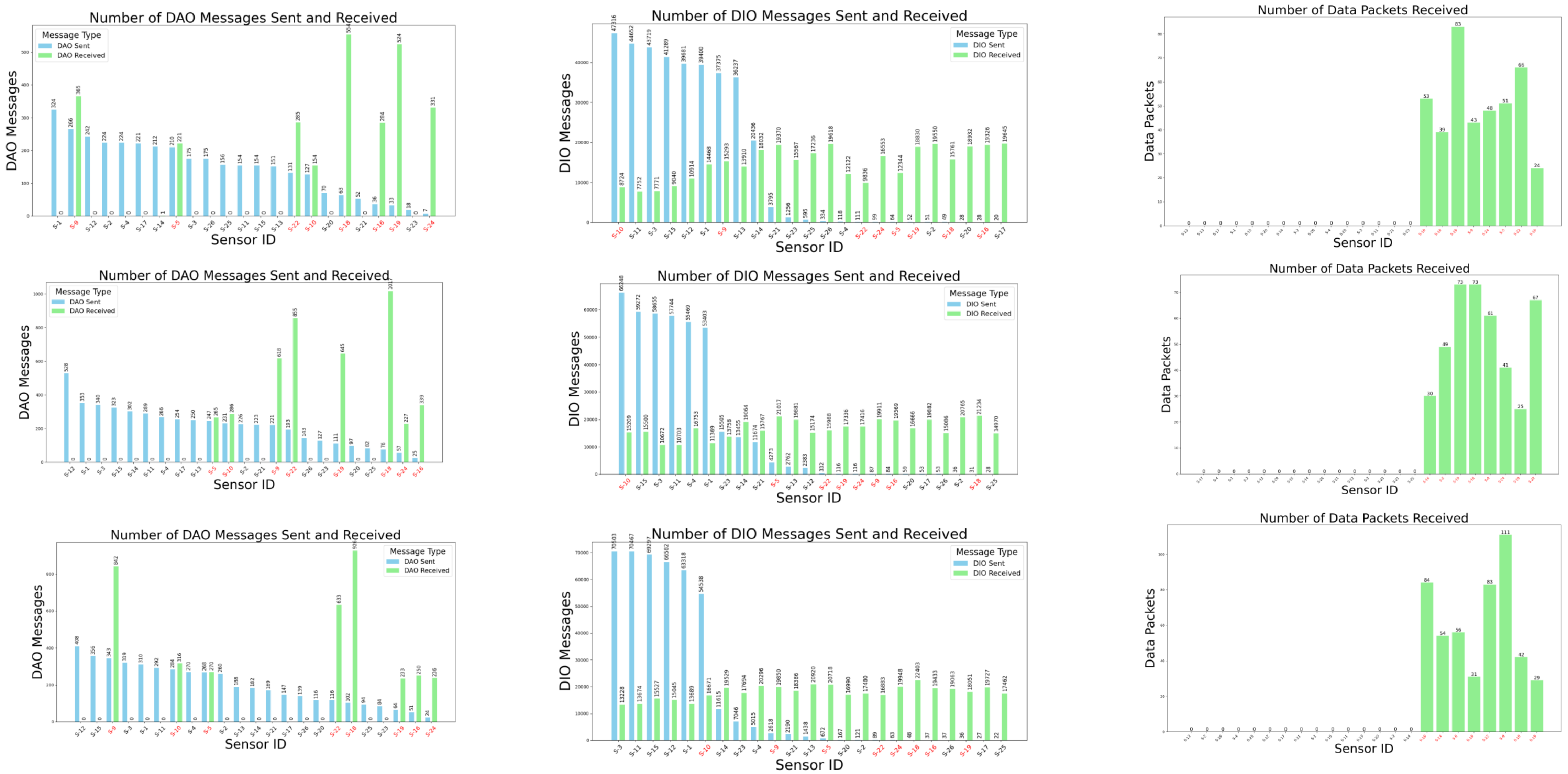
Feature visualization: 5 malicious nodes; 3 runs, each with a different random seed



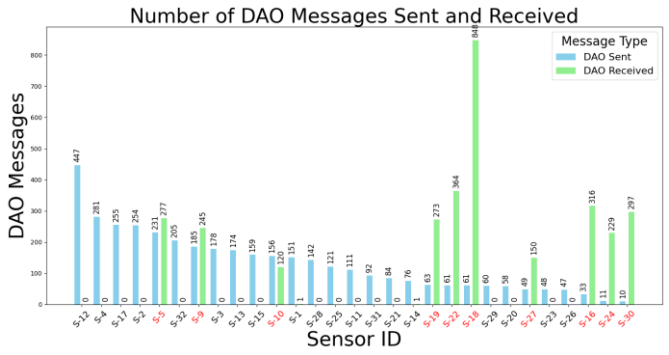
Feature visualization: 6 malicious nodes; 3 runs, each with a different random seed

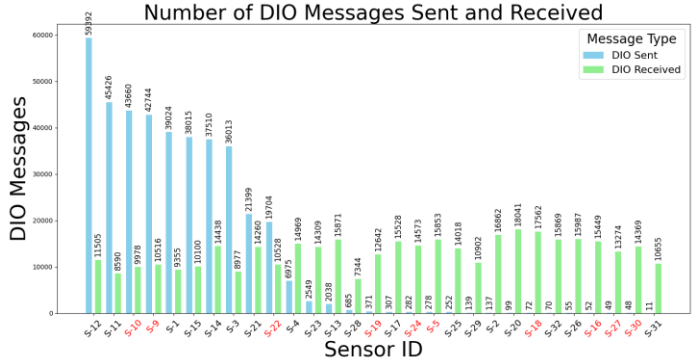


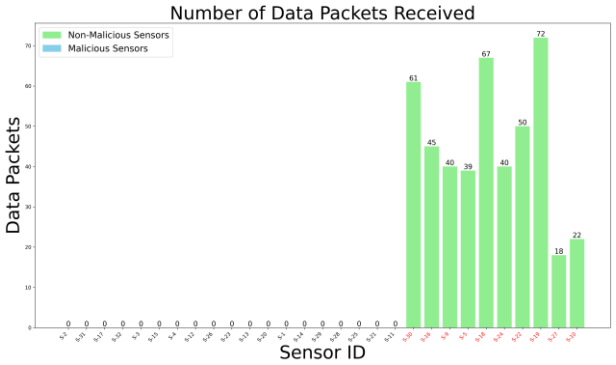
Feature visualization: 8 malicious nodes; 3 runs, each with a different random seed

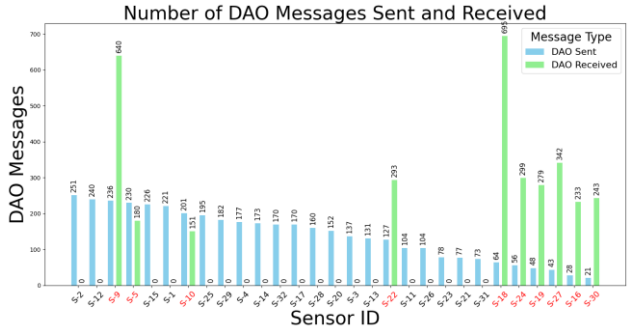


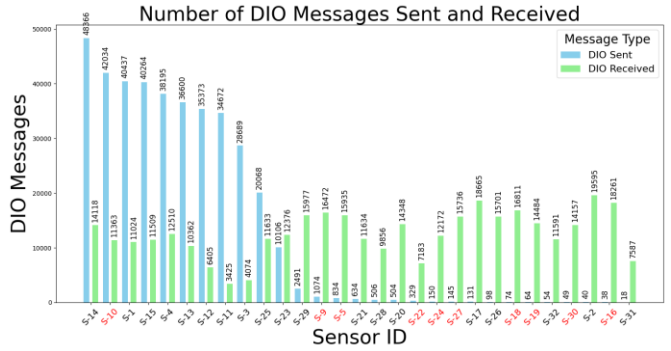
Feature visualization: 10 malicious nodes; 3 runs, each with a different random seed

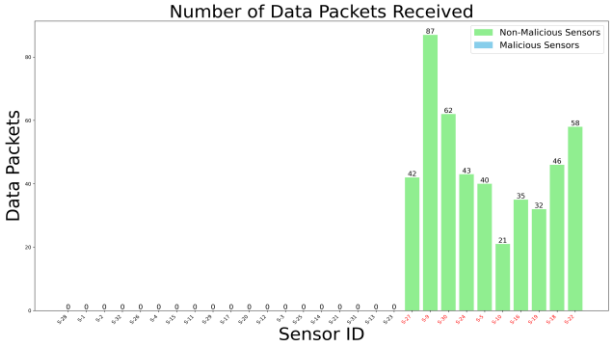


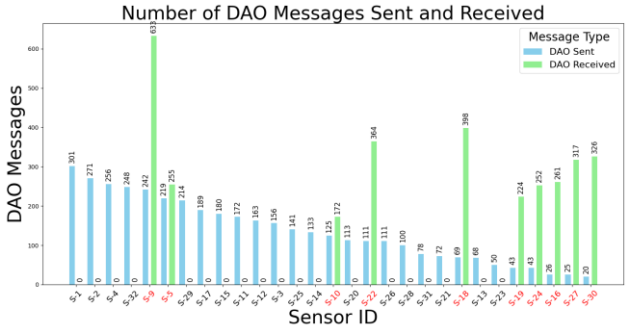


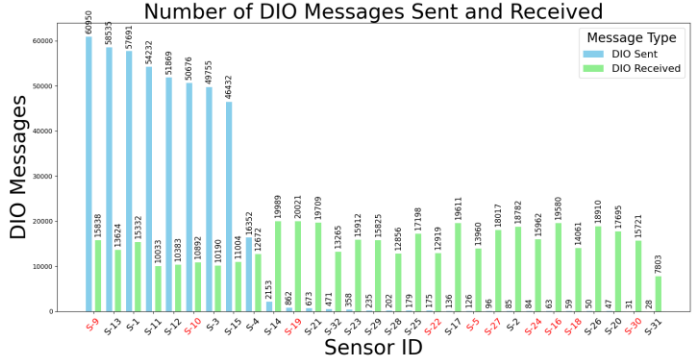


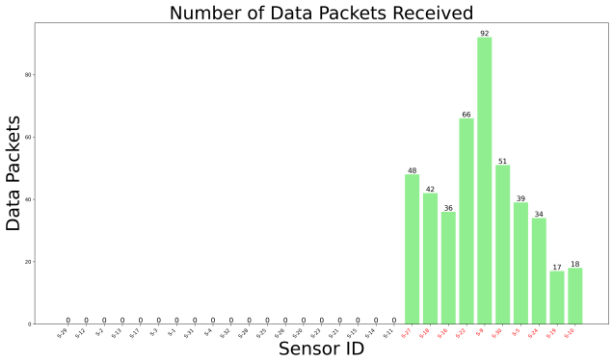




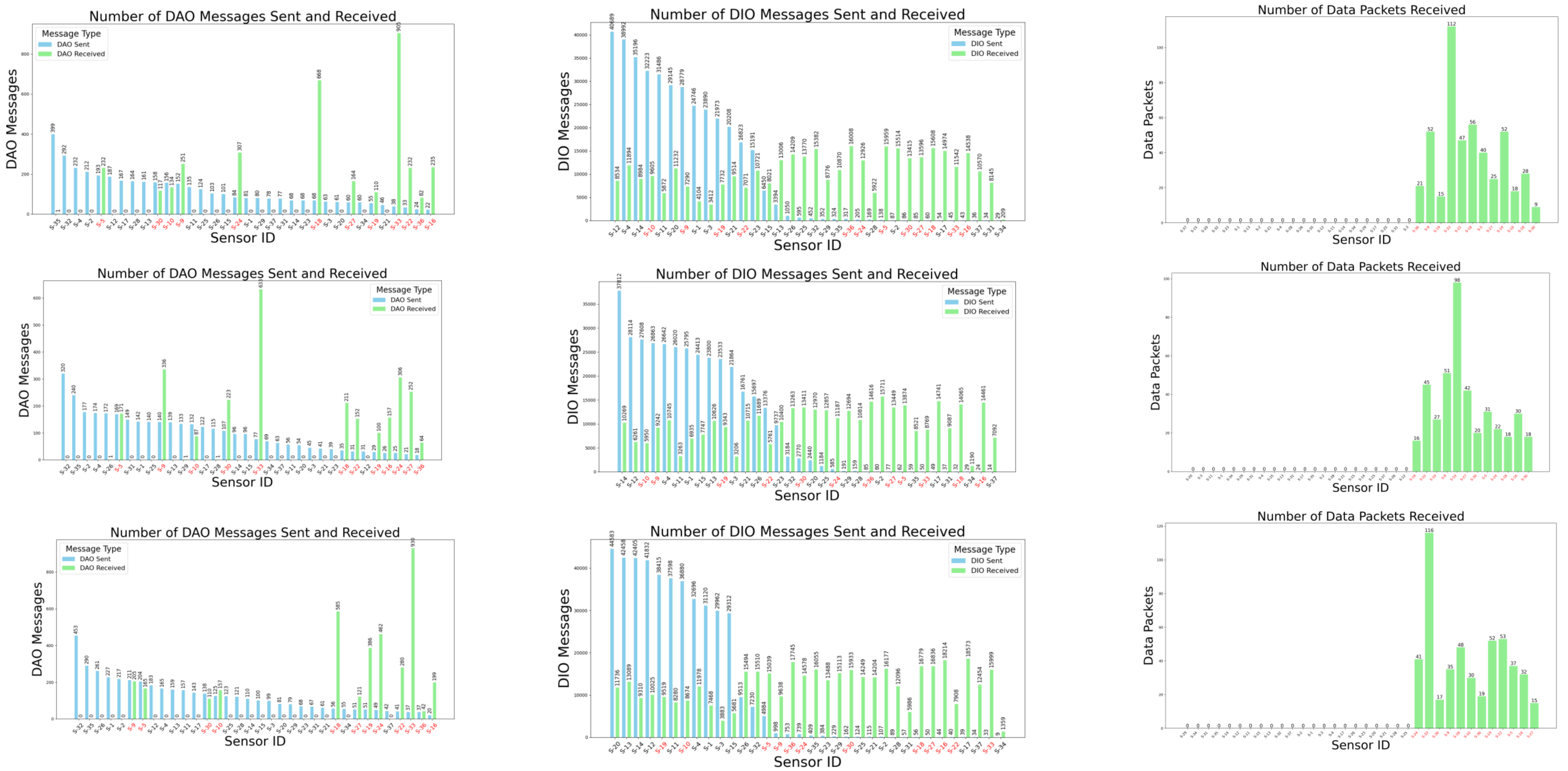








Feature visualization: 12 malicious nodes; 3 runs, each with a different random seed

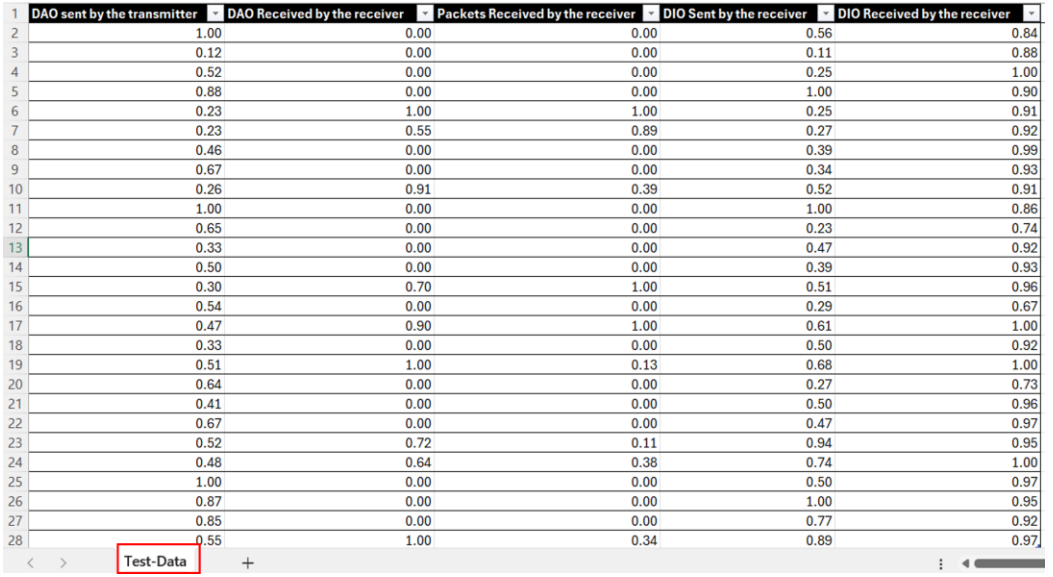


Attack scenarios with malicious nodes - Test data

- Created 6 scenarios with different numbers of nodes 9, 20, 26, 32, 37 and 42. In each scenario, there were 3, 7, 9, 11, 13, and 15 malicious nodes, respectively.
  - The simulations were run with 3 random seeds.
- We then enabled packet trace for all scenarios and used a python script to calculate the number of DAO, DIO, and data packets received by each sensor from the packet trace.
- The features considered for classifying the data are:
  - DAO Sent
  - DAO Received
  - DIO Sent
  - DIO Received
  - Data Packets Received

Data processing

- Extracted data from packet trace into an Excel sheet using python script.
- There were a total of 498 sensors, each with 5 features.
- Feature normalization: For each feature, the maximum value is calculated across all sensors. Each sensor's value is then divided by this maximum value to normalize the data and obtain values between 0 and 1. After obtaining the normalized values, the data is transposed Trained four types of machine learning classifiers on the dataset, named "Test Data".
- Ran the classifiers on the test data and classified data points as either malicious or non-malicious and plotted confusion matrix to compare the test data classifications against the trained data.

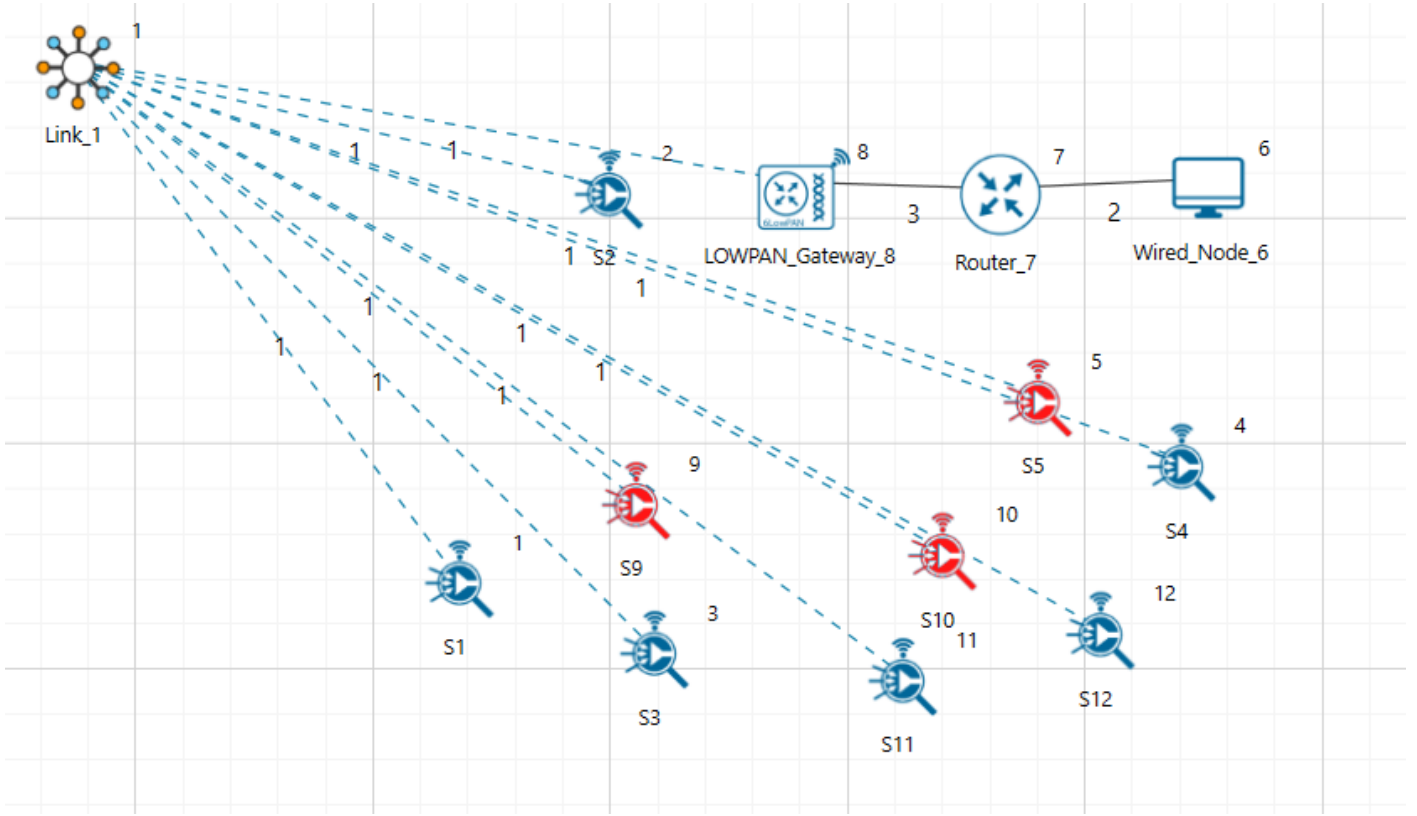


DAO sent by the transmitter	DAO Received by the receiver	Packets Received by the receiver	DIO Sent by the receiver	DIO Received by the receiver
1.00	0.00	0.00	0.56	0.84
0.12	0.00	0.00	0.11	0.88
0.52	0.00	0.00	0.25	1.00
0.88	0.00	0.00	1.00	0.90
0.23	1.00	1.00	0.25	0.91
0.23	0.55	0.89	0.27	0.92
0.46	0.00	0.00	0.39	0.99
0.67	0.00	0.00	0.34	0.93
0.26	0.91	0.39	0.52	0.91
1.00	0.00	0.00	1.00	0.86
0.65	0.00	0.00	0.23	0.74
0.33	0.00	0.00	0.47	0.92
0.50	0.00	0.00	0.39	0.93
0.30	0.70	1.00	0.51	0.96
0.54	0.00	0.00	0.29	0.67
0.47	0.90	1.00	0.61	1.00
0.33	0.00	0.00	0.50	0.92
0.51	1.00	0.13	0.68	1.00
0.64	0.00	0.00	0.27	0.73
0.41	0.00	0.00	0.50	0.96
0.67	0.00	0.00	0.47	0.97
0.52	0.72	0.11	0.94	0.95
0.48	0.64	0.38	0.74	1.00
1.00	0.00	0.00	0.50	0.97
0.87	0.00	0.00	1.00	0.95
0.85	0.00	0.00	0.77	0.92
0.55	1.00	0.34	0.89	0.97

Test-Data

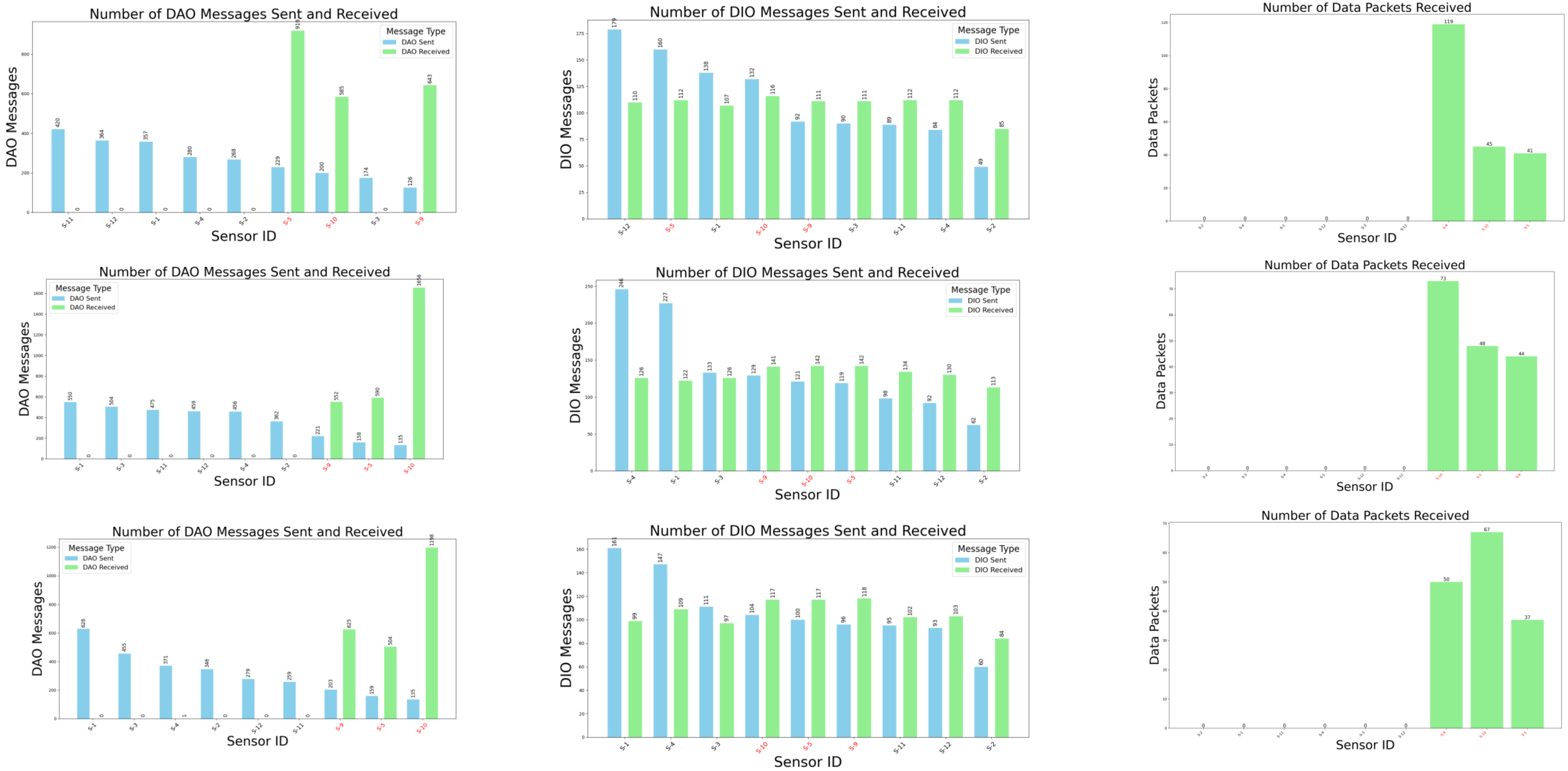
Data Classification–Test data

Network scenario in NetSim-3 malicious nodes

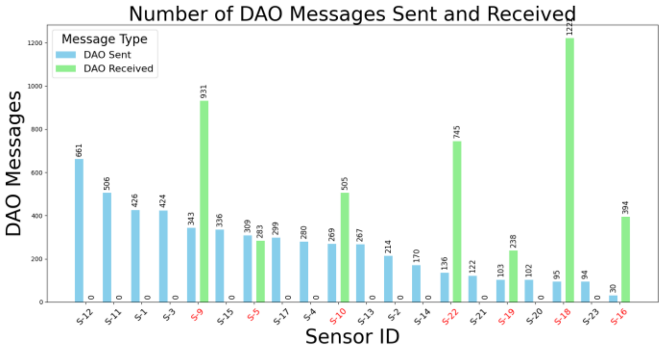


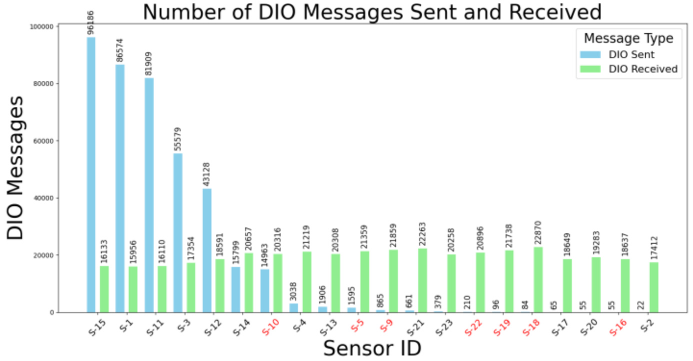
The network topology in IoT using RPL Protocol includes 3 malicious nodes Pathloss Model: Log Distance, Pathloss Exponent: 2

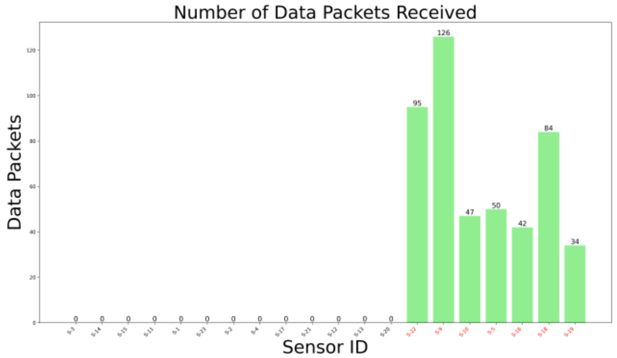
Feature visualization: 3 malicious nodes; 3 runs, each with a different random seed

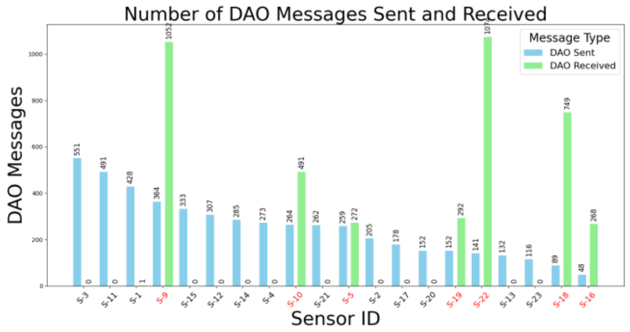


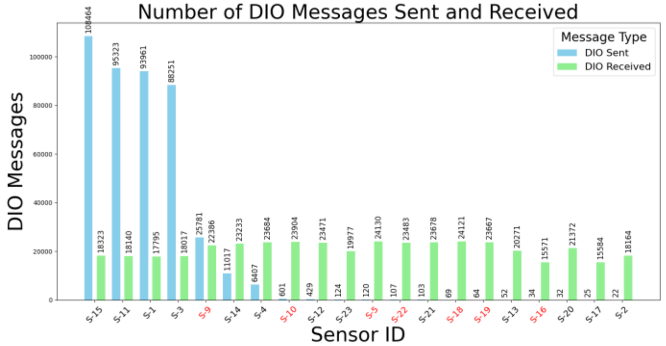
Feature visualization: 7 malicious nodes; 3 runs, each with a different random seed

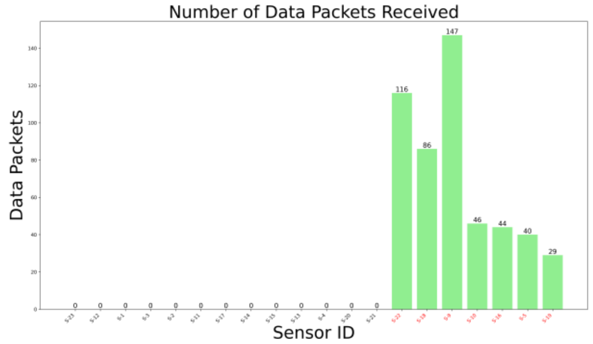


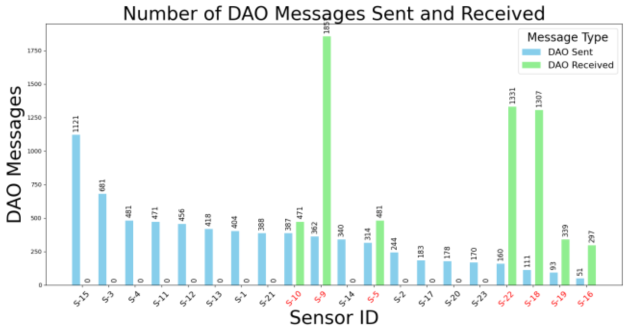


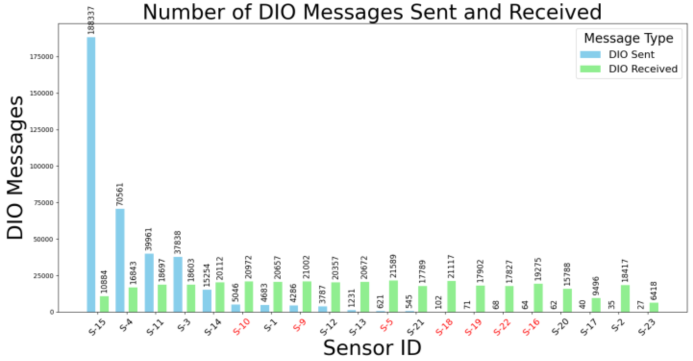


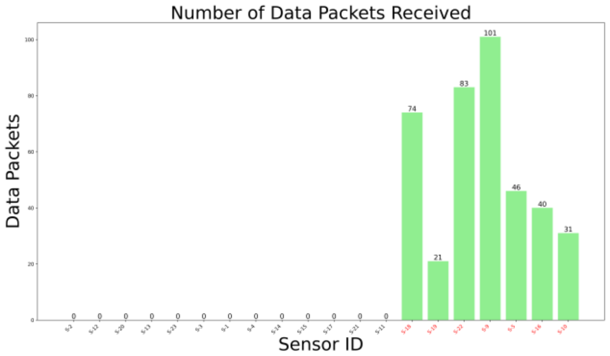




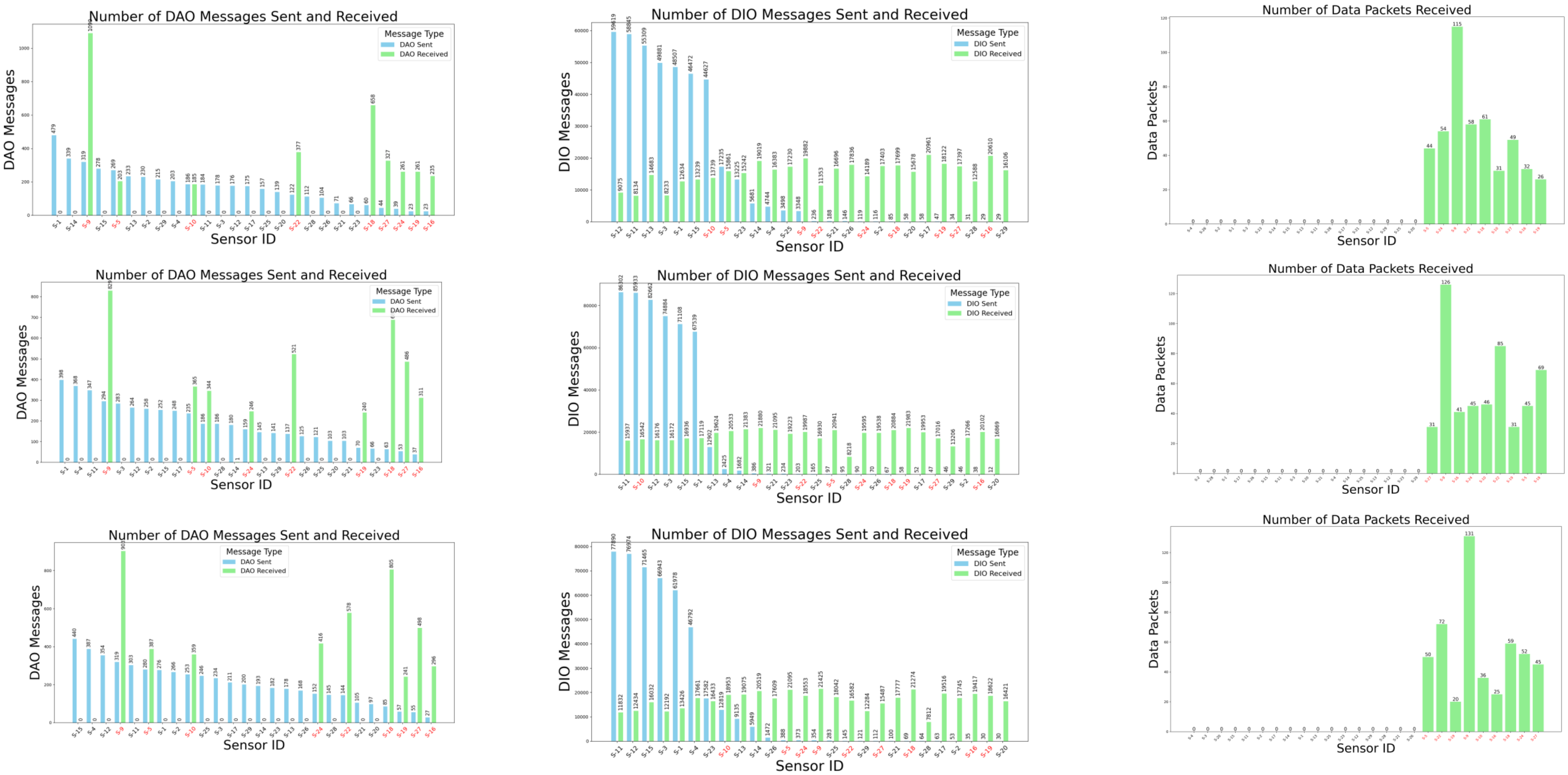




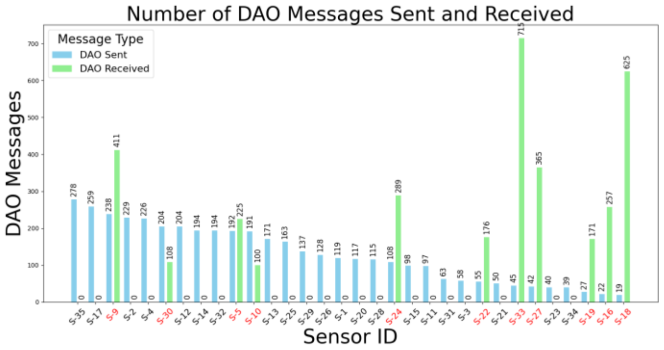


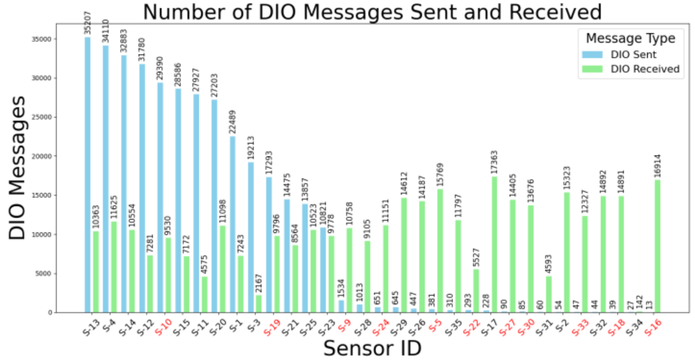


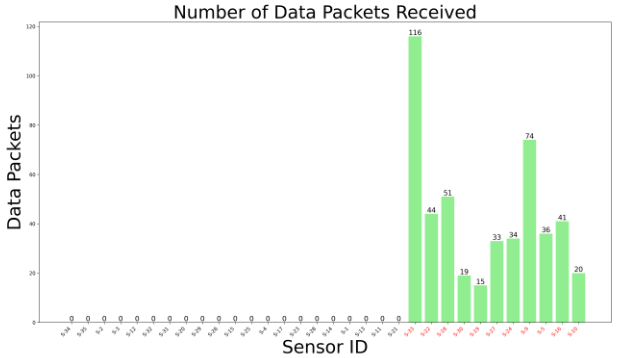
Feature visualization: 9 malicious nodes; 3 runs, each with a different random seed

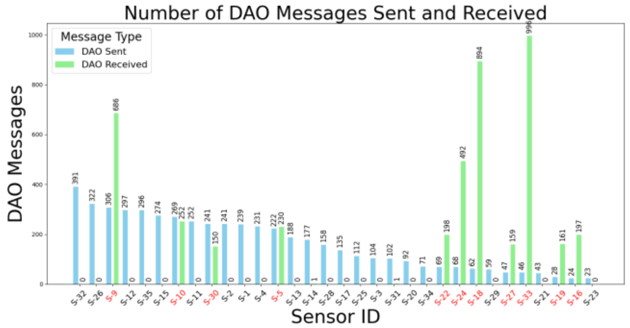


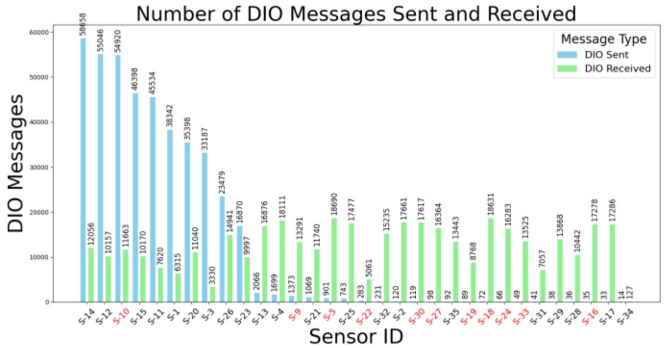
Feature visualization: 11 malicious nodes; 3 runs, each with a different random seed

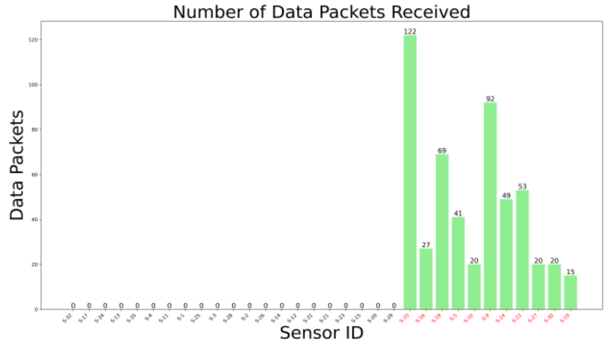


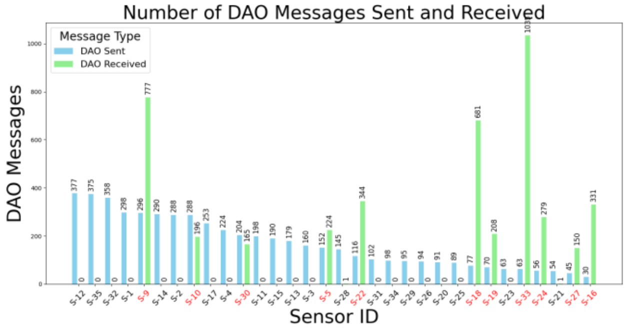


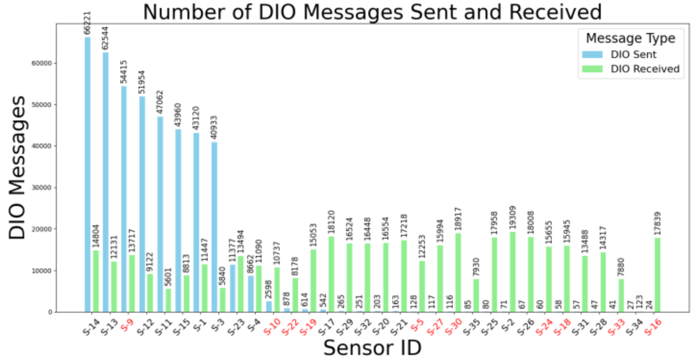


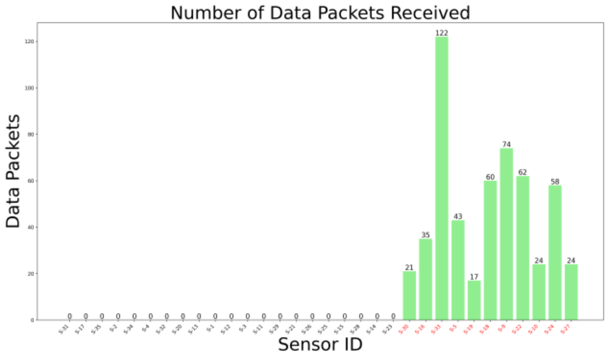




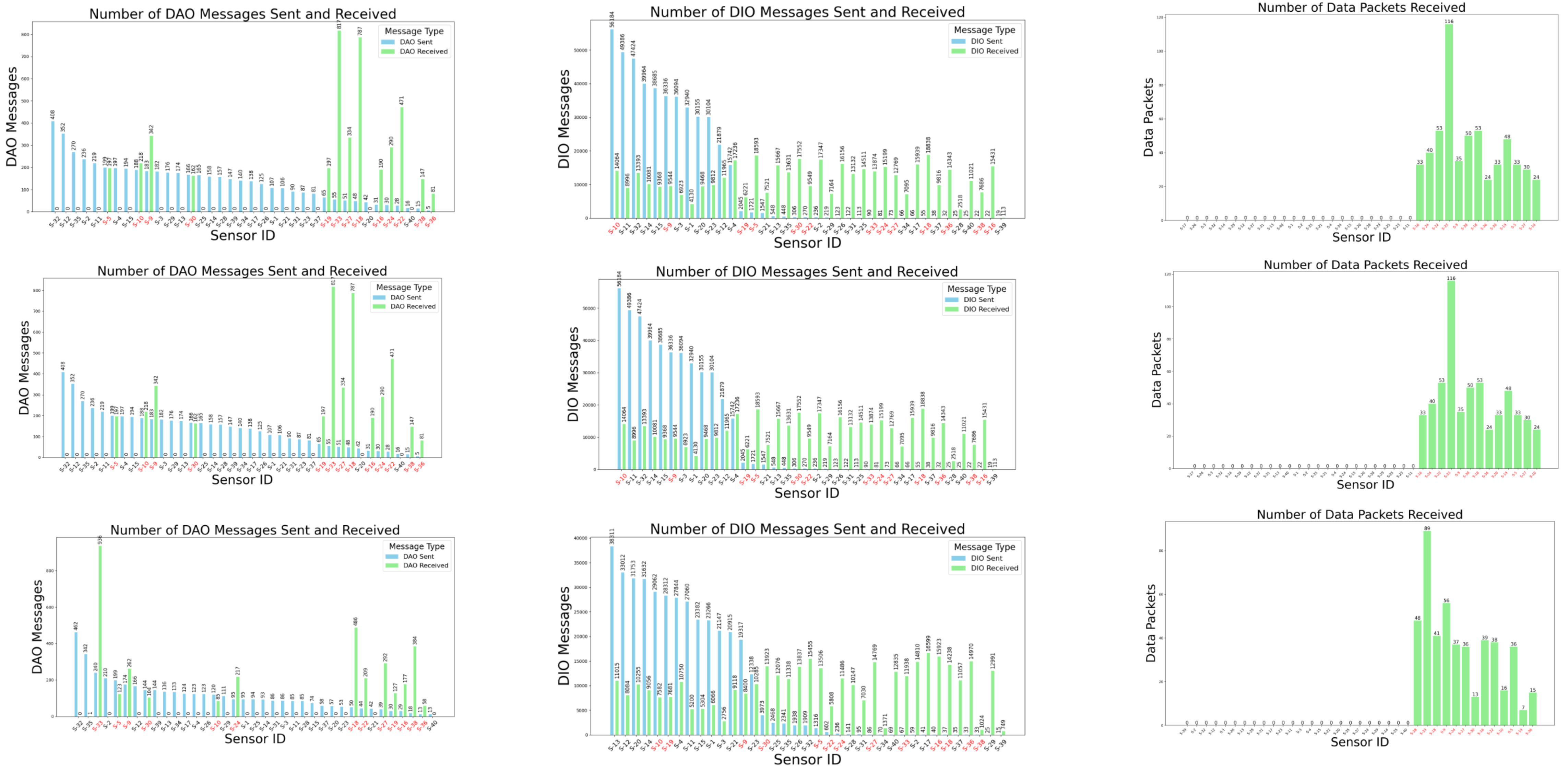




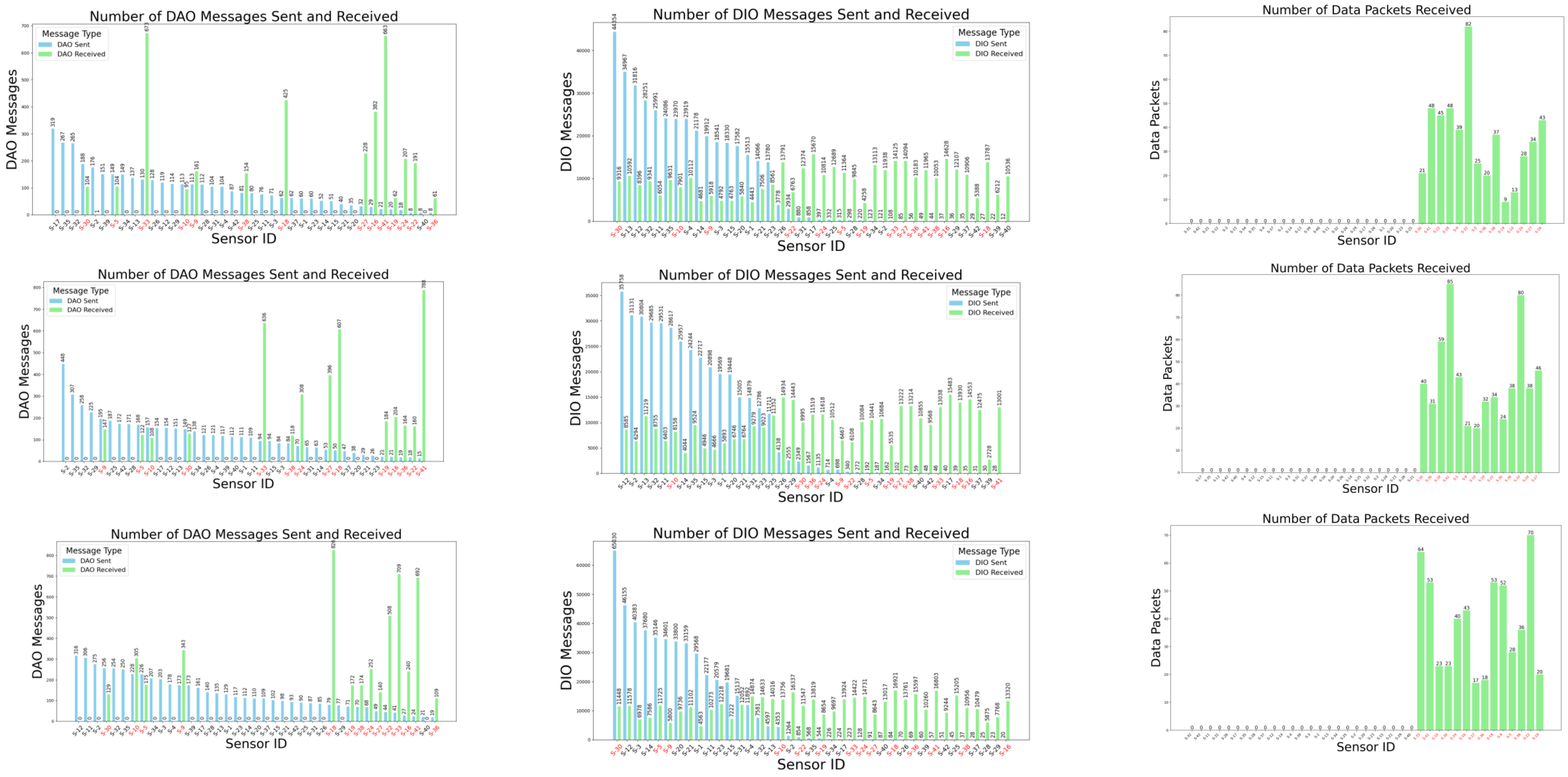




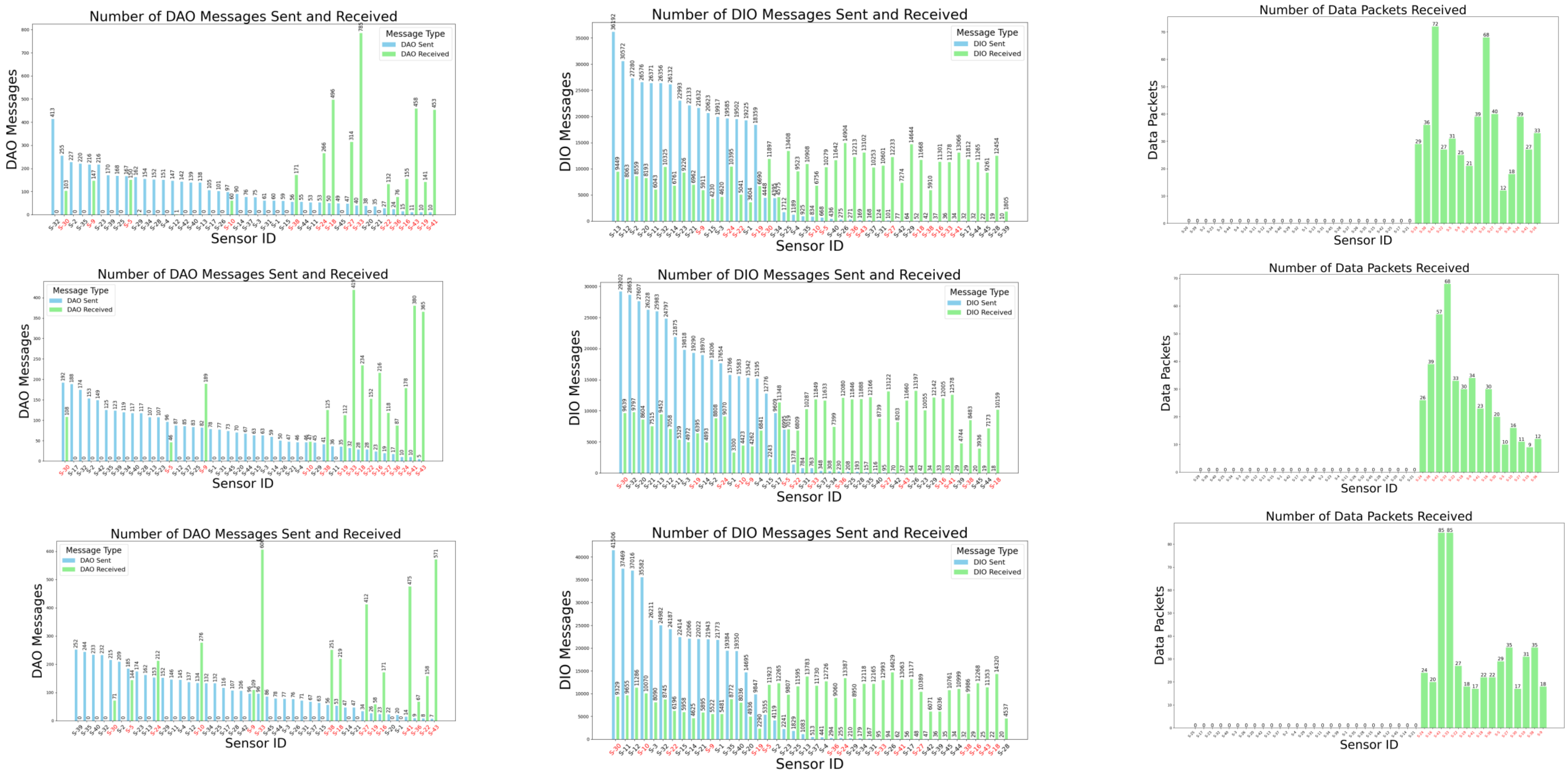
Feature visualization: 13 malicious nodes; 3 runs, each with a different random seed



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



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



Detection of Malicious nodes using Machine Learning Classifiers

K-Nearest Neighbor classifier:

- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.

Naive Bayes classifier:

- Naive Bayes is a probabilistic classifier based on Bayes' Theorem. It assumes that the features are independent given the class label, which simplifies the calculation of the likelihood of a class given the features.
- The class with the highest posterior probability is selected as the prediction.

Support Vector Machine classifier:

- Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems.
- The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

Logistic Regression classifier:

- Logistic regression is a linear model used for binary classification. It predicts the probability that a given input belongs to a particular class using the logistic function.
- The output probability is converted into a class label (0 or 1) by setting a threshold (usually 0.5).

Confusion matrix

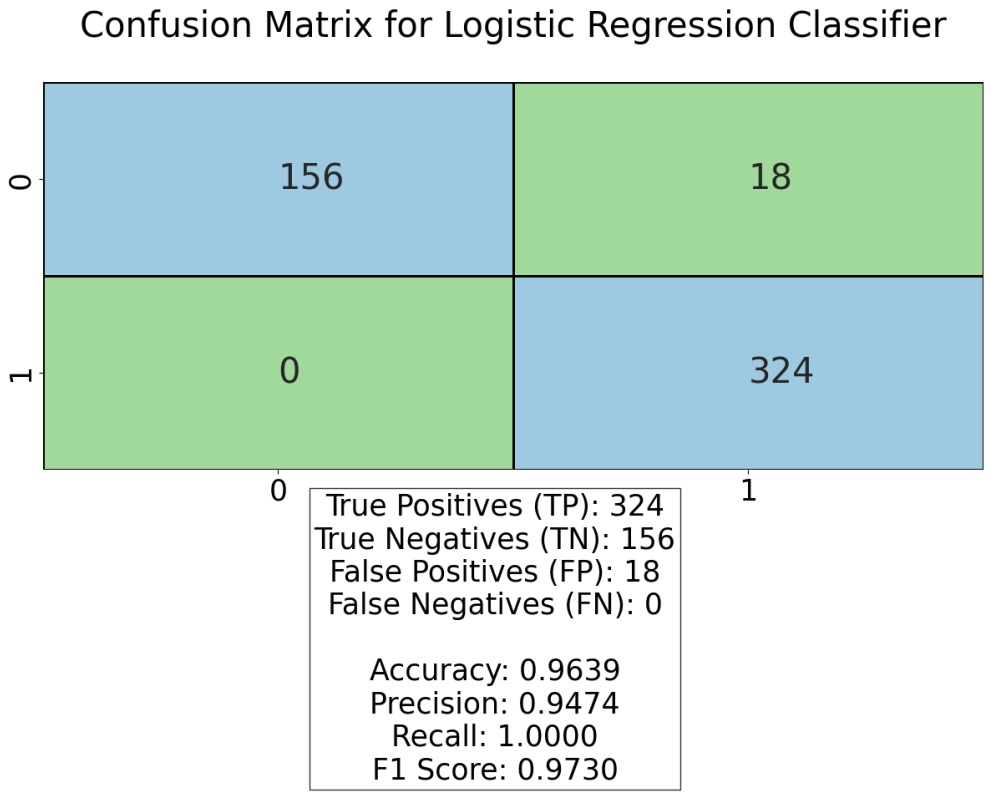
Confusion Matrix:

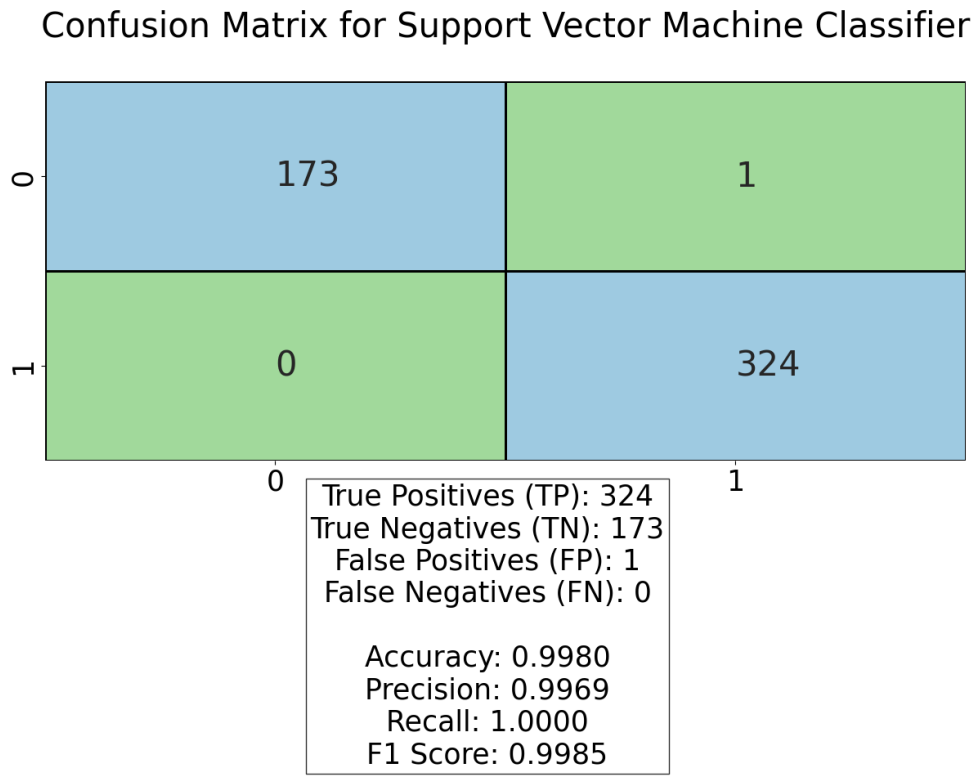
- Confusion matrix summarizes the performance of a machine learning model on a set of test data. It is a means of displaying the number of accurate and inaccurate instances based on the model's predictions. It is often used to measure the performance of classification models, which aim to predict a categorical label for each input instance.
- The matrix displays the number of instances produced by the model on the test data.

True Positive (TP)	The model correctly predicted a positive outcome (the actual outcome was positive
True Negative (TN)	The model correctly predicted a negative outcome (the actual outcome was negative
False Positive(FP)	The model incorrectly predicted a positive outcome (the actual outcome was negative). Also known as a Type I error
False Negative(FN)	The model incorrectly predicted a negative outcome (the actual outcome was positive). Also known as a Type.

- Created a python script to generate confusion matrices for six types of classifiers to evaluate the accuracy, precision, F1 score, and recall of the test data against the manually labeled test data.
- Ran the predicted data of each classifier against the labeled test data to obtain the confusion matrix**.**

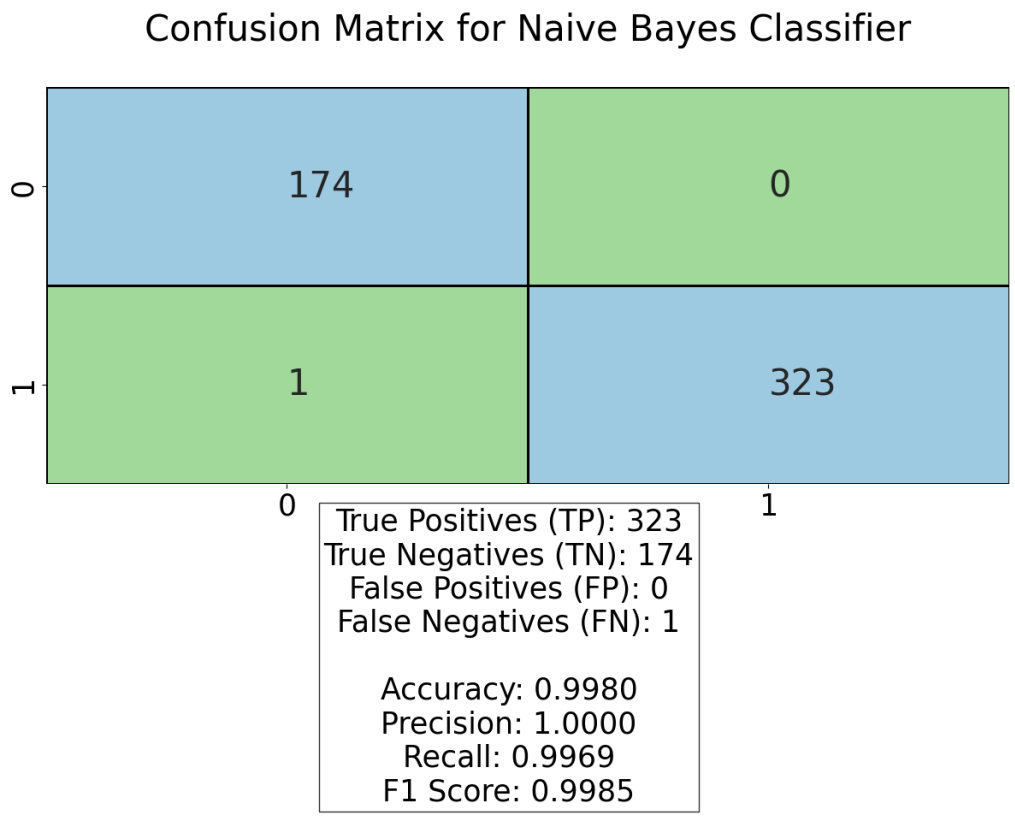
Confusion Matrix: Accuracy, Precision, F1 Score, Recall

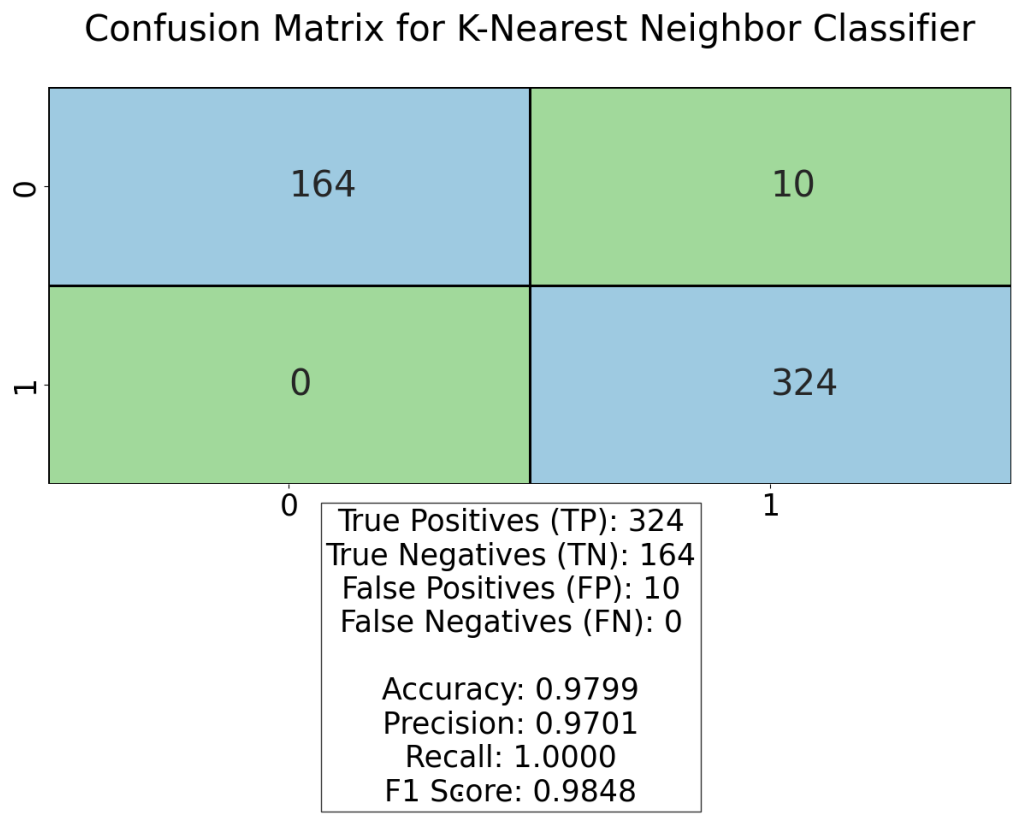




.

Confusion Matrix: Accuracy, Precision, F1 Score, Recall





Comparison of four classifiers

Classifier	True Positives	True Negatives	False Positives	False Negatives	Accuracy	Precision	Recall	F1 Score
Naïve Bayes	323	174	0	1	0.9980	1.00	0.9969	0.9985
K-Nearest Neighbor	324	164	10	0	0.9799	0.9701	1.000	0.9848
Logistic Regression	324	156	18	0	0.96390	0.9474	1.000	0.9730
Support Vector Machine	324	173	1	0	0.9980	0.9969	1.000	0.9985

Naive Bayes Classifier and **Support Vector Machine (SVM)** both achieve the highest accuracy at 99.80%.

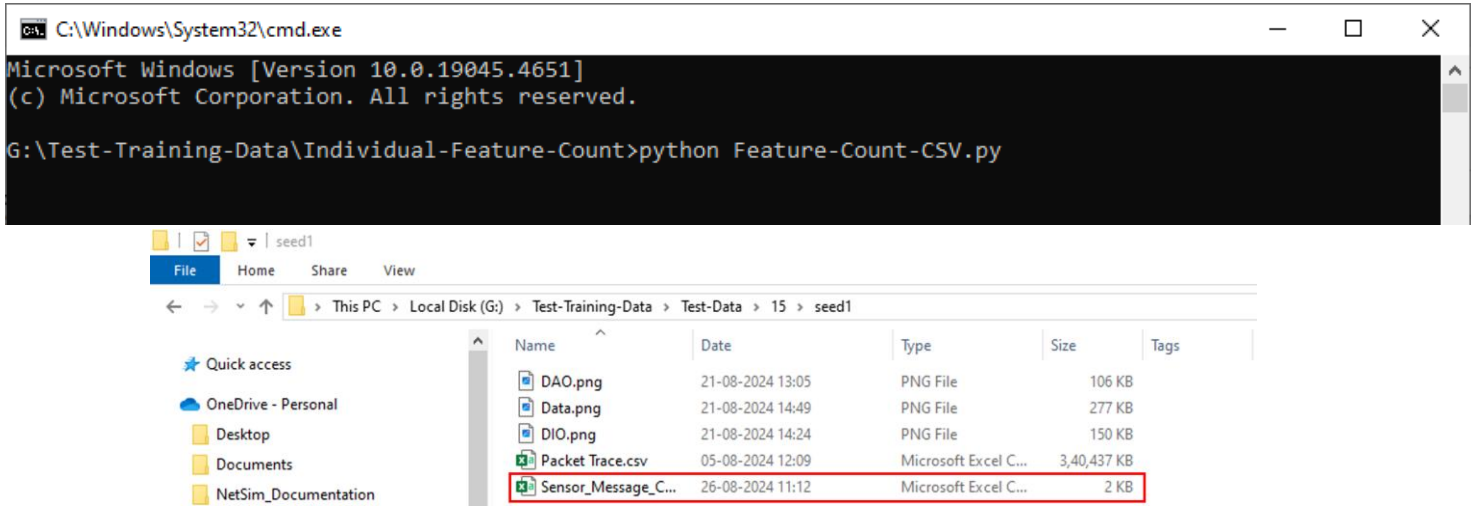
H. L. wijdan Choukri, “RPL rank attack detection using Deep Learning,” 2020 International Conference on Innovation and Intelligence for Informatics,, 2021

Appendix

How to classify the data?

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

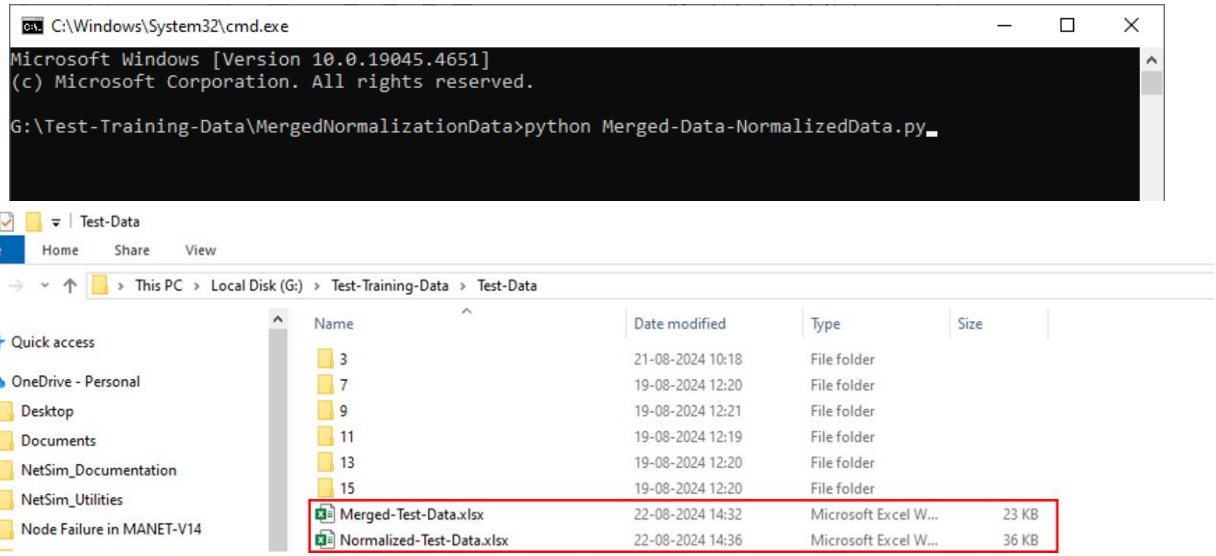
- Modify the file paths in the python script according to your setup.
- Open the command prompt.
- Navigate to the folder containing the python script.
- Run the Feature-Count-CSV script to process the packet trace file and generate the excel file.
- You can place the python script anywhere, as long 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 that contains packet traces.



How to Normalize the data?

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

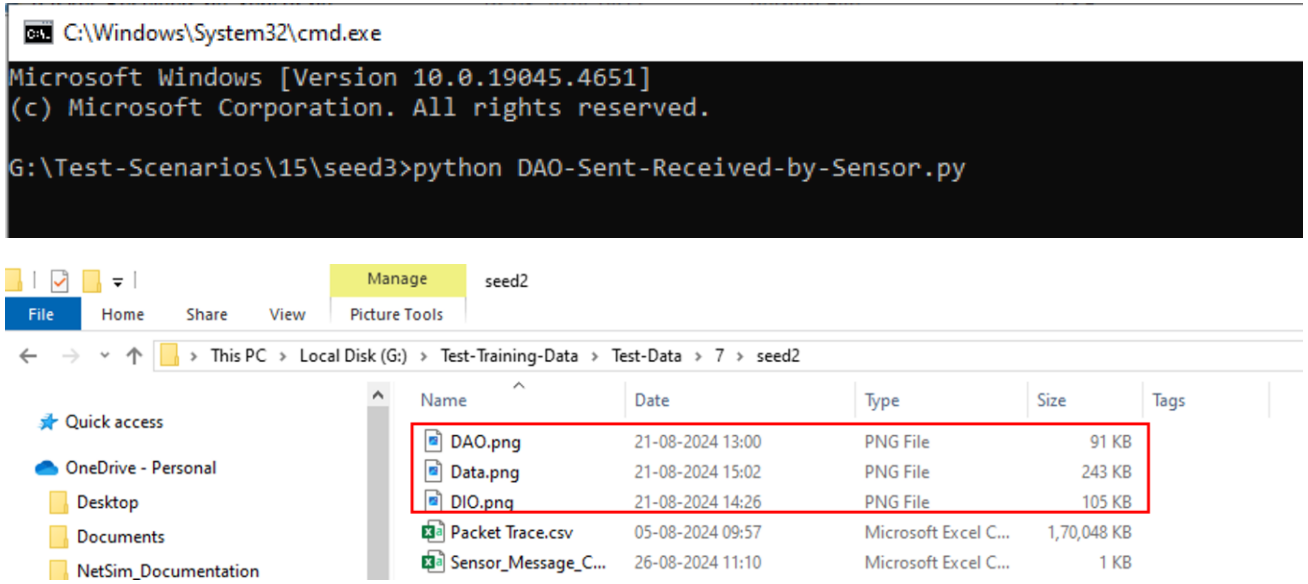
- Modify the file paths in the python script according to your setup.
- Open the command prompt.
- Navigate to the folder containing the python script.
- Run the Merged-Data-NormalizedData script to process the Sensor_Message_Counts file and generate the normalize excel file.
- You can place the python 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 that contains the python script.



How to generate plots?

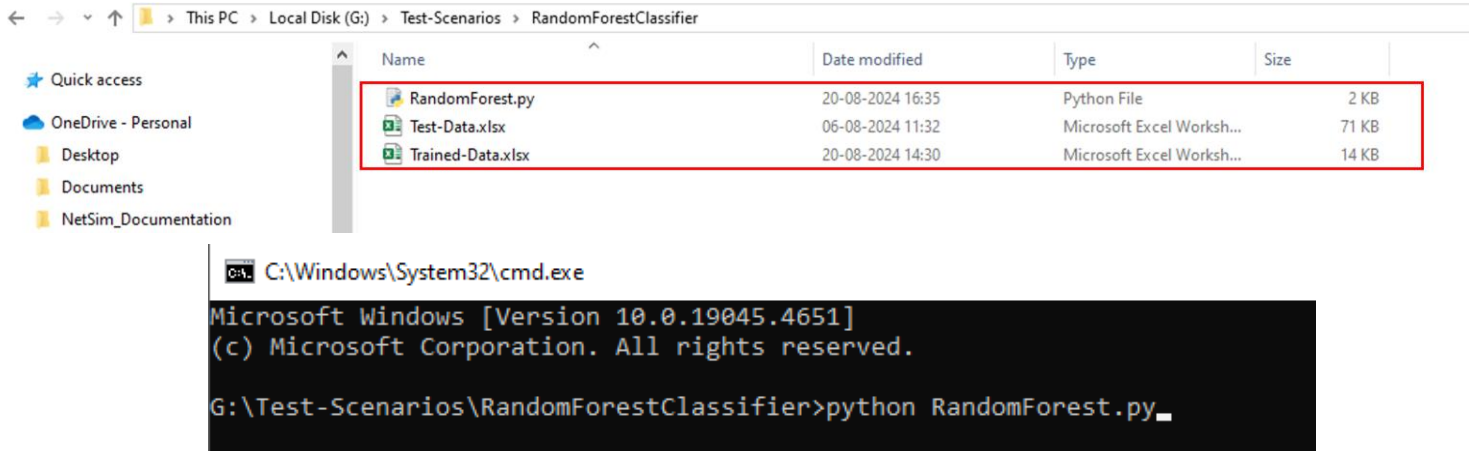
Download the Workspace and place it in your desired location.

- The Python scripts can be placed anywhere, as long as the 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 Python scripts.
- Run the DAO , DIO , Packet Received scripts to process the packet trace files and generate the plots.
- Make sure the necessary packet trace files are accessible based on the paths defined in the scripts.



How to run the classifiers?

- After normalizing the data, name it 'Test-Data’.
- Run each classifier against the trained data by providing the file locations for both the Trained-Data and Test data.
- Place the Test-Data , and Training-data in the same folder.
- Modify the path in the python script of each classifier according to your setup.
- Open the command prompt and run the script as shown below



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 these predicted label excel files along with the real training data to run the python script and generate the confusion matrix.
- Place the predicted label excel files, and the real training data in the same folder.
- Modify the paths in the python script of Confusion-Matrix according to your setup.
- Open the command prompt and run the script as shown below.
- Each classifier's data generates a confusion matrix for that classifier