

NetSim VANETs

Detecting sybil node attacks in VANETs using Machine Learning

Applicable Release: NetSim v15 or higher

Applicable Version(s): NetSim Standard/Pro

Project download link: https://github.com/NetSim-TETCOS/Sybil_Node_VANETs_v15.0/archive/refs/heads/main.zip

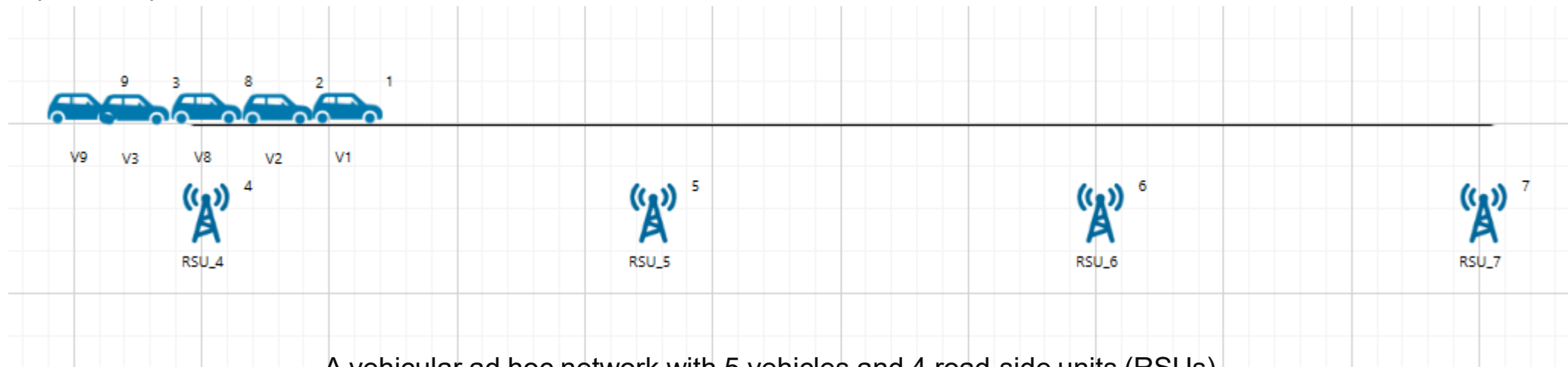
The URL has the exported NetSim scenario for the examples used in this document and the python scripts to run classifiers.

Outline

- Introduction to VANETs
- Sybil attack in VANETs
- Modeling Sybil attack in VANETs using NetSim
- Attack Scenarios with sybil nodes – Training data
 - Attack scenarios with 1,1, 1, 2, 3, 3 sybil nodes
 - Data processing
 - Feature visualization
- Attack Scenarios with sybil nodes – Test
 - Attack scenarios with 1, 1, 1, 3, 3, 3 sybil nodes
 - Data processing
 - Feature visualization
 - Classification
- Detection of sybil nodes using ML based classifiers
- Confusion Matrix: Accuracy, Precision, F1 Score, Recall
- Comparison between different classifiers: Random Forest, K-Nearest Neighbor, Decision Tree, Gradient boost.

Introduction to VANETs

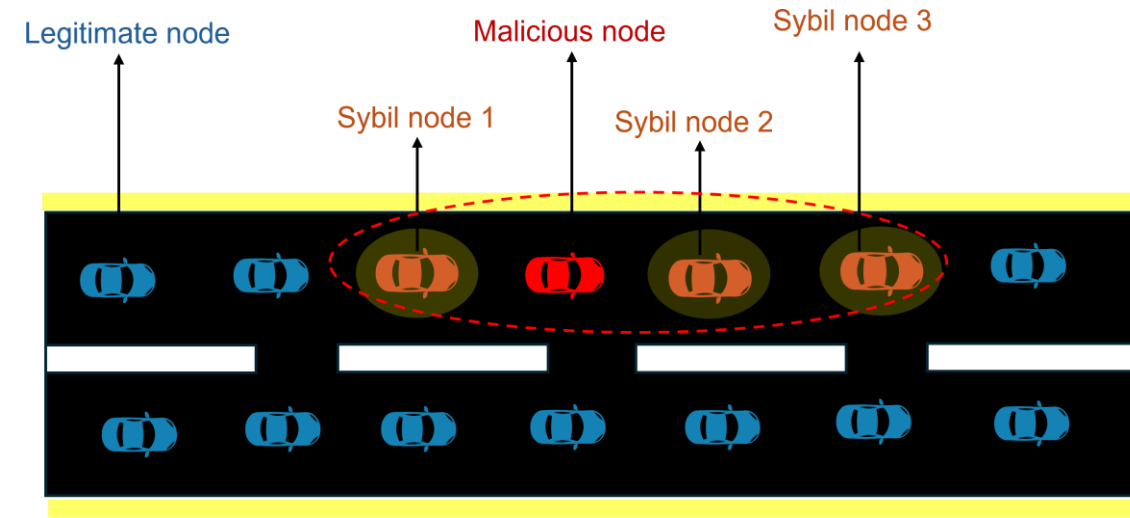
- VANETs (Vehicular Ad Hoc Networks) are crucial for Intelligent Transportation Systems (ITS) and road safety.
- Vehicles communicate with each other (V2V) and with infrastructure (V2I).
- VANETs operate in dynamic environments, with changing topologies due to vehicle speed and varying road conditions.
- The communication architecture follows the IEEE 802.11p standard for PHY and MAC layers and the IEEE 1609 standard for upper layers, enabling Wireless Access in Vehicular Environments (WAVE).



A vehicular ad hoc network with 5 vehicles and 4 road-side units (RSUs)

Sybil attack in VANETs

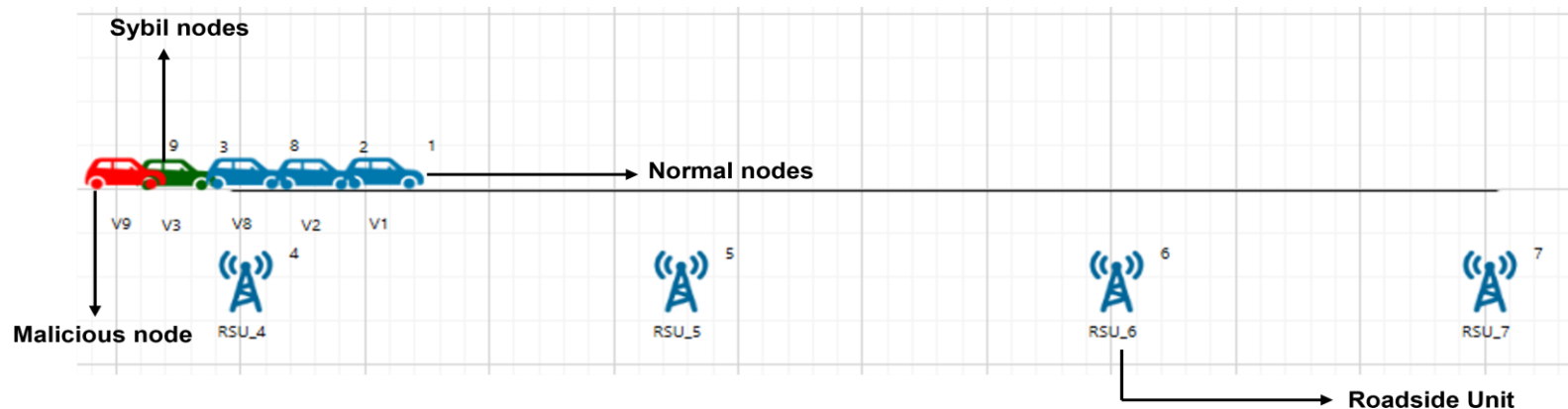
- **Definition:** A Sybil attack in VANETs occurs when a malicious vehicle creates multiple fake identities or "ghost vehicles" in the network.
- **Mechanism:** The attacker broadcasts messages from these fake identities, making it appear as if multiple vehicles are present when in reality, it's just one malicious node.
- **Purpose:** These attacks aim to manipulate traffic data, influence routing decisions, or gain an unfair advantage in safety-critical applications.
- **Impact:** Sybil attacks can lead to false traffic congestion reports, manipulated voting systems in VANETs, and compromised safety applications, potentially causing accidents or traffic disruptions.
- **Detection:** RSSI values are used to estimate the physical location of nodes in the network. In a Sybil attack, multiple fake identities originating from the same physical location should have very similar RSSI values when measured by receiving nodes. By analyzing the similarity of RSSI patterns malicious nodes can be detected.



Sybil attack scenario in VANETs

Sybil attack in VANETs using NetSim

- A Sybil attack in NetSim involves a malicious node (sybil attacker) creating illusory nodes, known as a Sybil nodes.
- The malicious node transmits messages as if it were two separate entities, misleading the network into believing there are two independent participants.
- To model this in NetSim, we create two nodes: V3 (the malicious node) and V9 (the Sybil node), where both nodes behave as if they are distinct entities.
- The scenario has 5 vehicles (3 real + 1 attacker + 1 sybil node) and 4 RSUs, each vehicle transmits data to all RSUs.
- Since the signal strength from the sybil attacker and the sybil node would be similar, RSSI-based classification can be used to detect malicious nodes

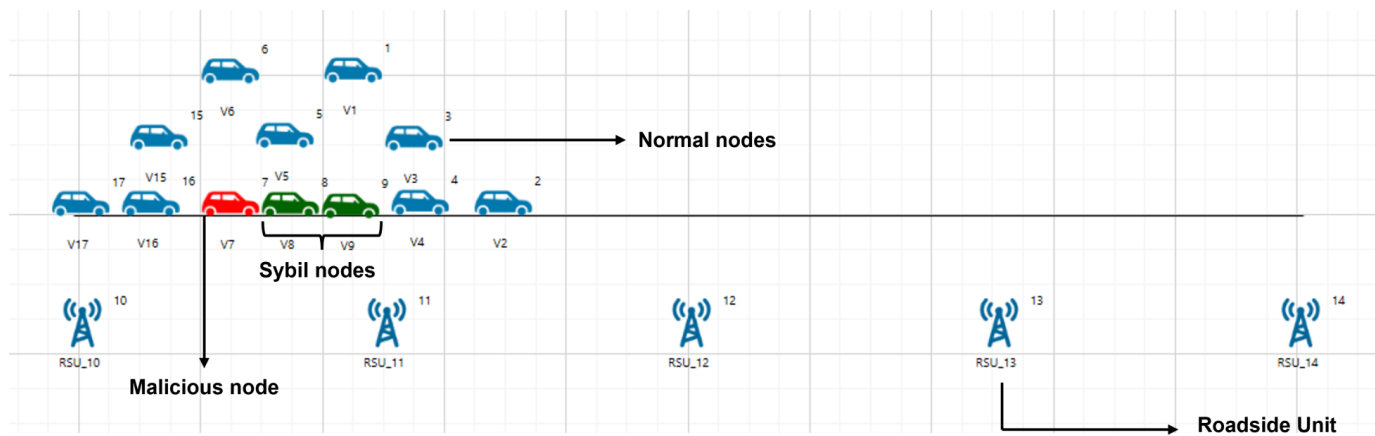


The network topology in VANETs includes 1 malicious node (sybil attacker in red) and 1 illusion (sybil node in orange) and 3 normal vehicular (in blue)

Training

Attack scenarios – Training data generation

- In NetSim, we created 5 scenarios with varying total node counts (5, 7, 9, 11, 12 to 13).
- Total Node count = Normal node count + Sybil attacker count + Sybil node count
- Sybil node count in the 6 scenarios: 1, 1, 1, 2, 3, 3.
 - 1 malicious node in all cases
 - 1, 1, 1, 2, 3, 3 sybil nodes in each respective sample
- In NetSim, we enabled radio measurement log for all scenarios
- We wrote a python script to analyze the log file data and calculate the RSSI Power at each RSU from each vehicle, RSSI difference and RSSI Similarity
- Feature Extraction: RSSI Power, RSSI Difference, RSSI Similarity



The network topology in VANETs includes 1 malicious node (sybil attacker in red) and 2 illusion (sybil node in green) and 3 normal vehicular (in blue)

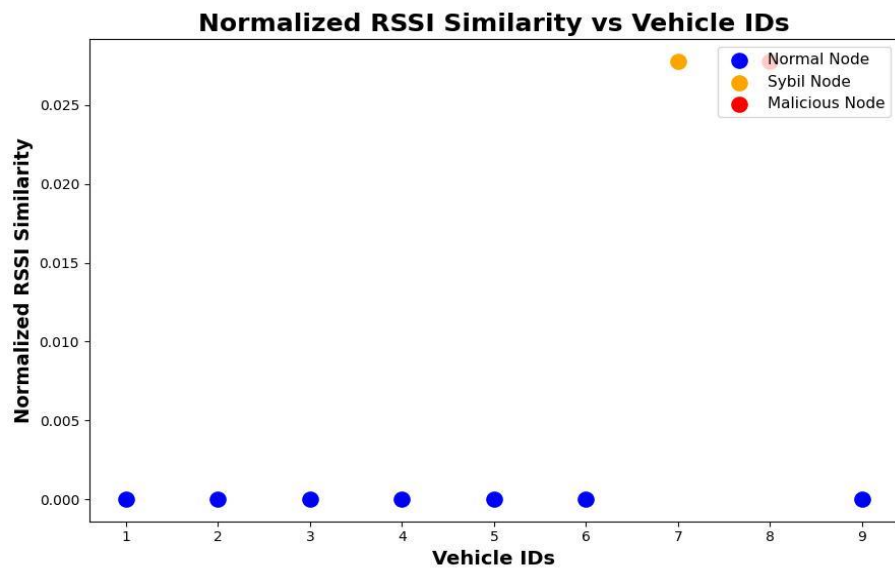
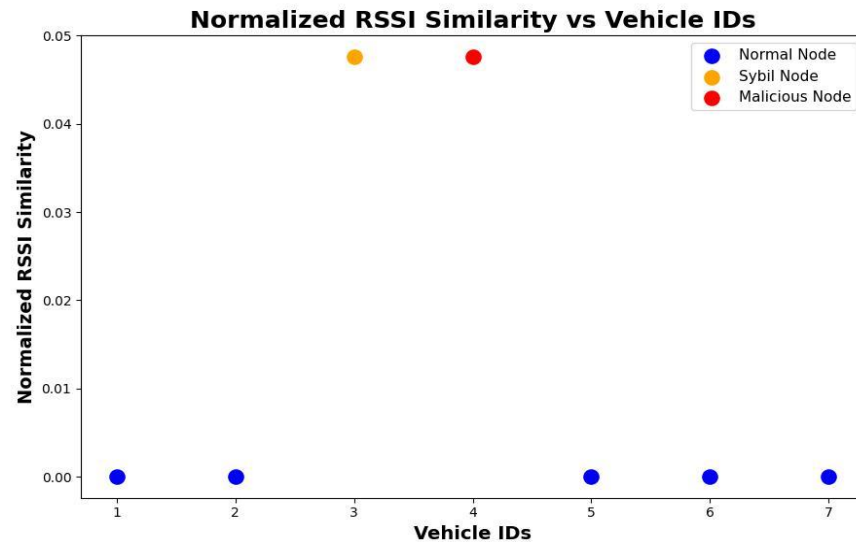
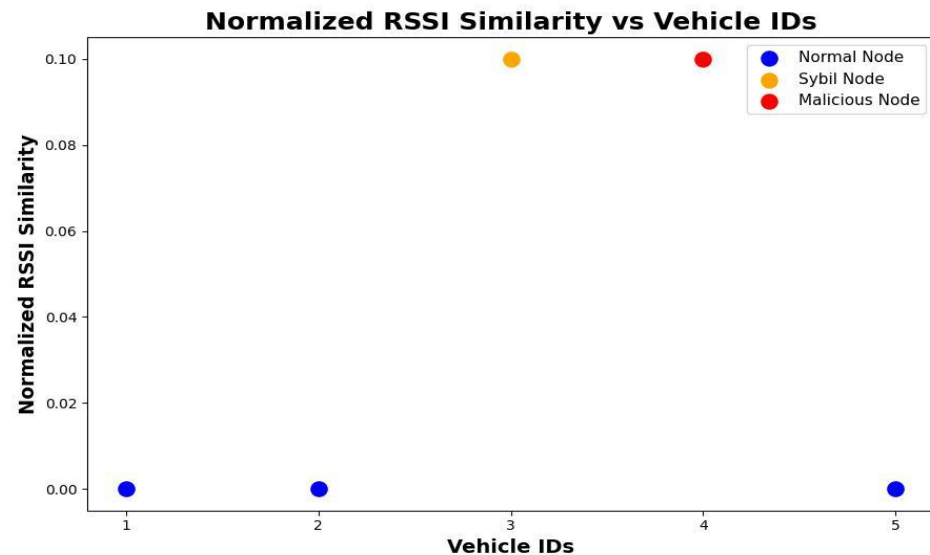
Data processing and feature visualization

- Data extraction from IEEE radio measurement log to Excel using Python script
- Total dataset: 57 vehicles, 3 features each
- Feature normalization process:
 - Calculate the pairwise RSSI differences between vehicles for each RSU.
 - Compute the mean RSSI difference for each vehicle pair.
 - Assign a threshold to classify vehicle pairs based on RSSI differences.
 - Threshold values in our case are 0.1, 0.09, 0.03 & 0.02.
 - Identify similar vehicle pairs based on the threshold.
 - Calculate the proportion of similar pairs from the total vehicle pairs.
 - Identify each unique vehicle pair.
 - Calculate the average RSSI similarity for each unique pair.
 - Label the similarity score as 0 or 1, depending on the RSSI similarity value.

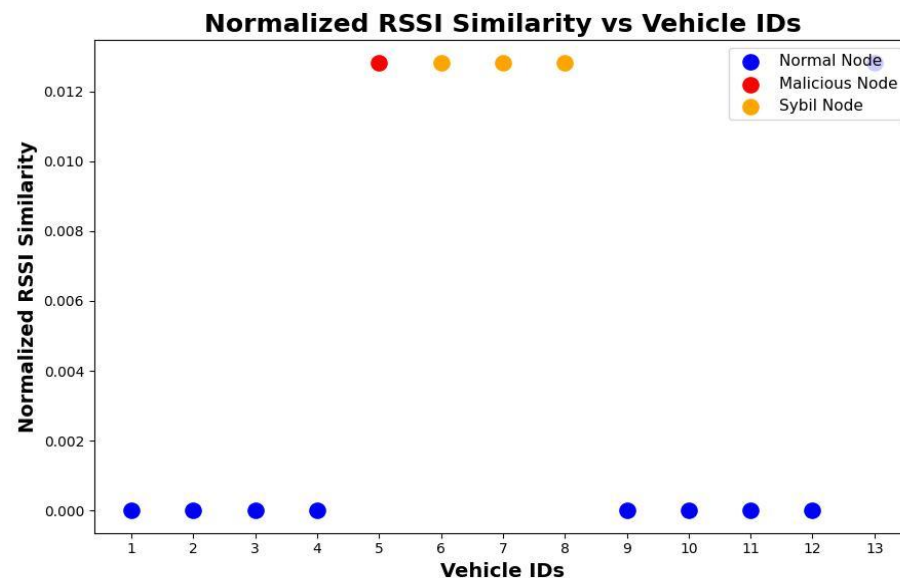
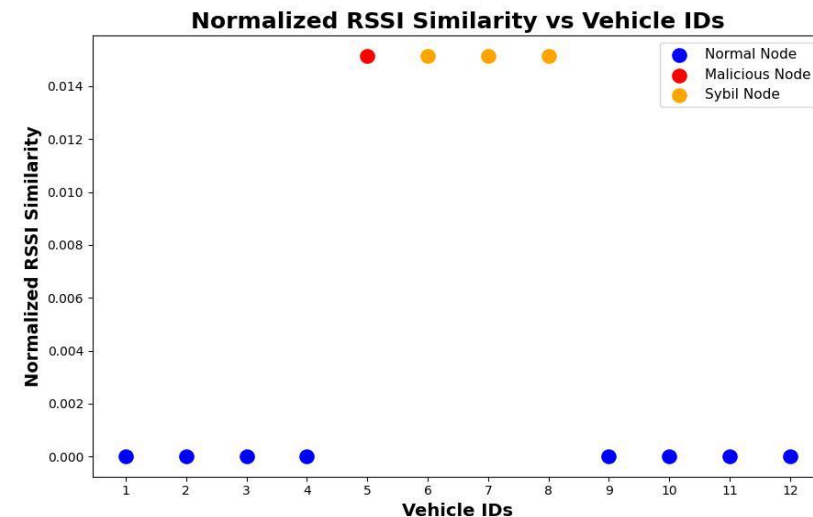
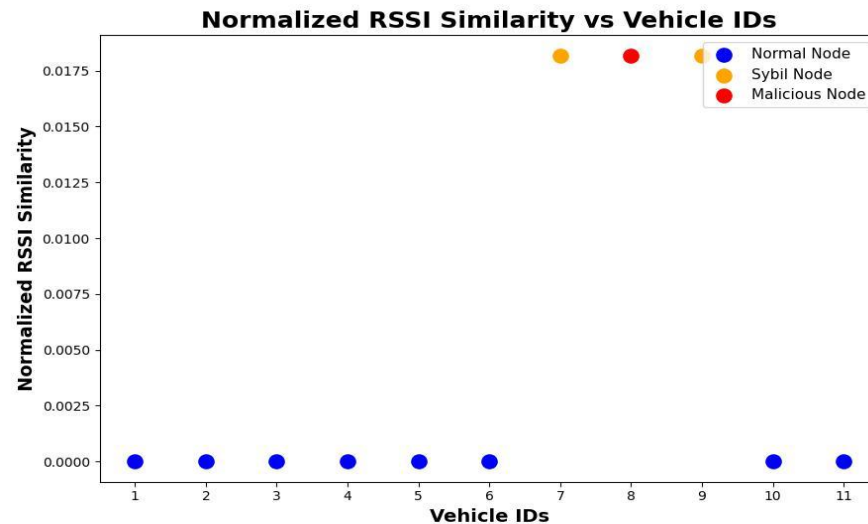
Pair	RSSI Similarity	Similarity Score
2-1	0.000	0
3-1	0.000	0
3-2	0.000	0
8-1	0.000	0
8-2	0.000	0
8-3	0.000	0
9-1	0.000	0
9-2	0.000	0
9-3	0.000	0
9-8	0.100	1
1-4	0.000	0
1-6	0.000	0
1-7	0.000	0
2-1	0.000	0
2-4	0.000	0
2-6	0.000	0
2-7	0.000	0
3-1	0.000	0
3-2	0.000	0
3-4	0.048	1
3-6	0.000	0
3-7	0.000	0
4-6	0.000	0
4-7	0.000	0

We label the similarity score based on features

Feature visualization: 5 ,7 and 9 vehicles



Feature visualization: 11, 12 and 13 vehicles



Classifier training

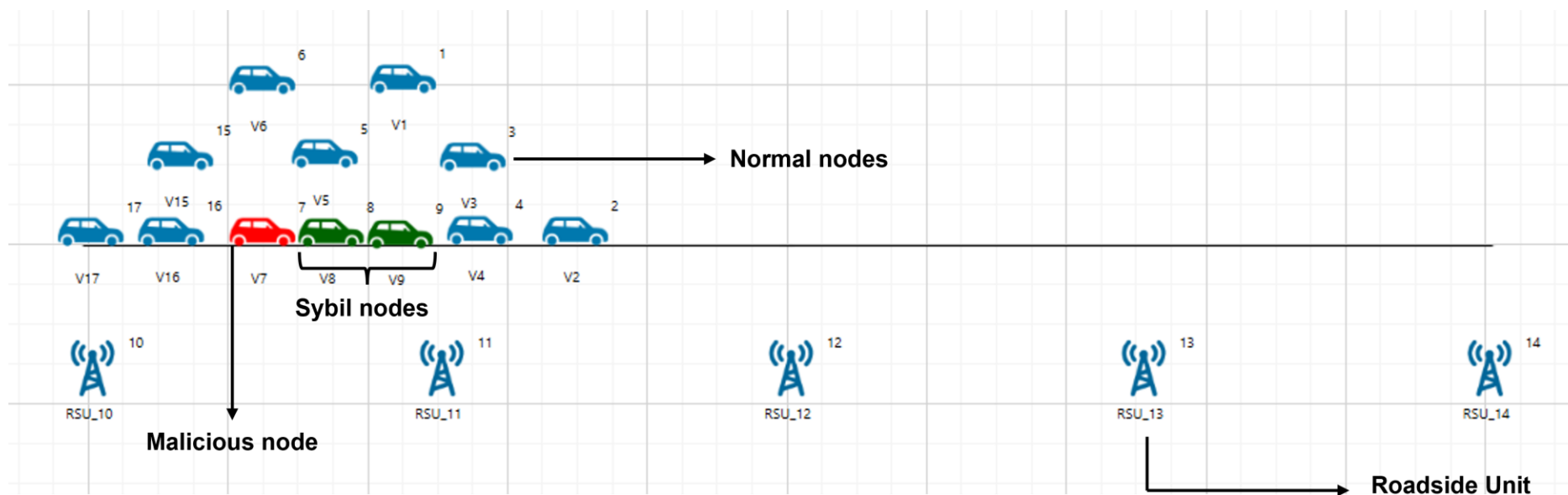
Features data was used to train the following classifiers:

- K-Nearest Neighbor
- Random Forest
- XGBoost Classifier
- Decision Tree

Inference

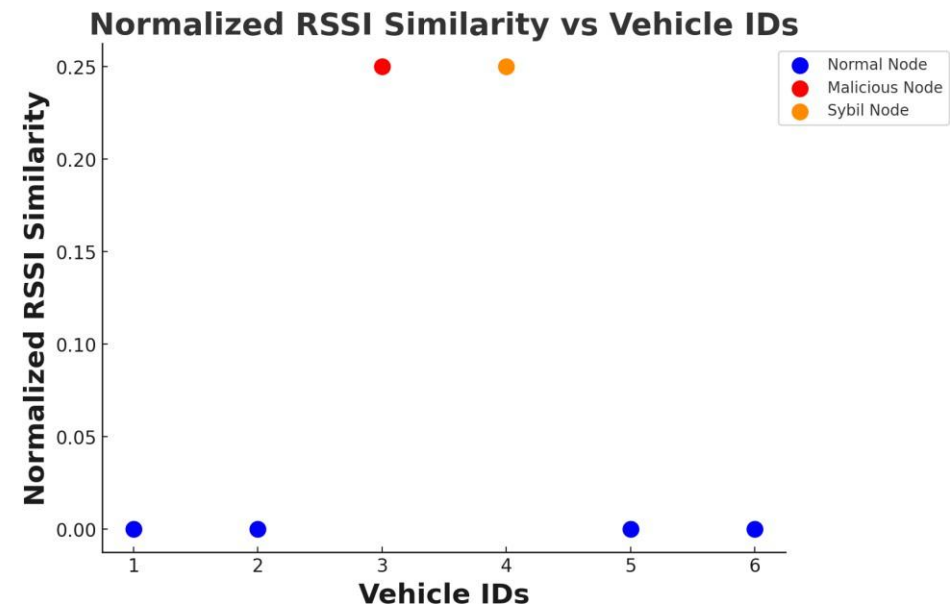
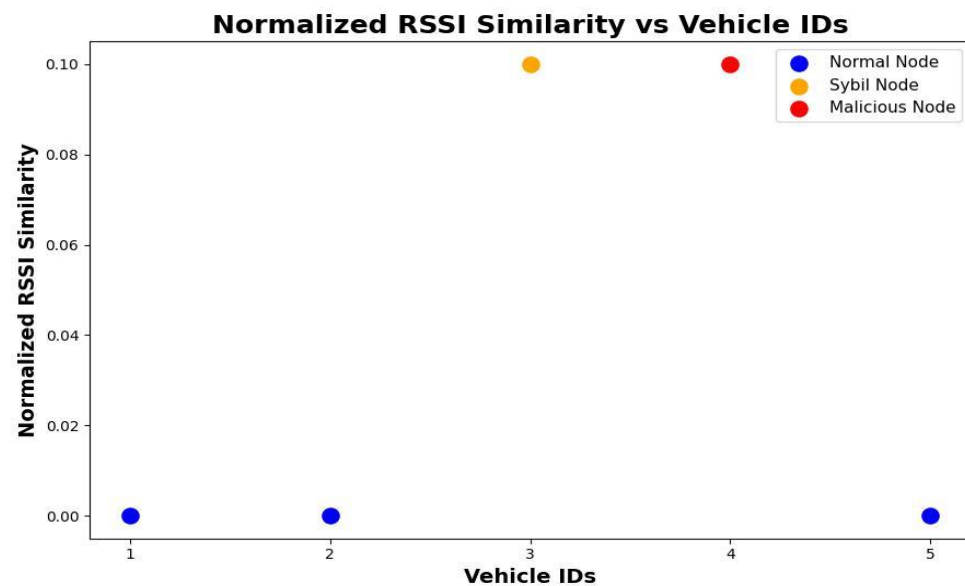
Inference and Test Scenarios

- Created 6 new scenarios with different node counts (5, 6, 9, 12, 12, 14).
- Sybil node count: 1, 1, 1, 3, 3, 3.
- Normalized the data using a python script as explained in slide 8.

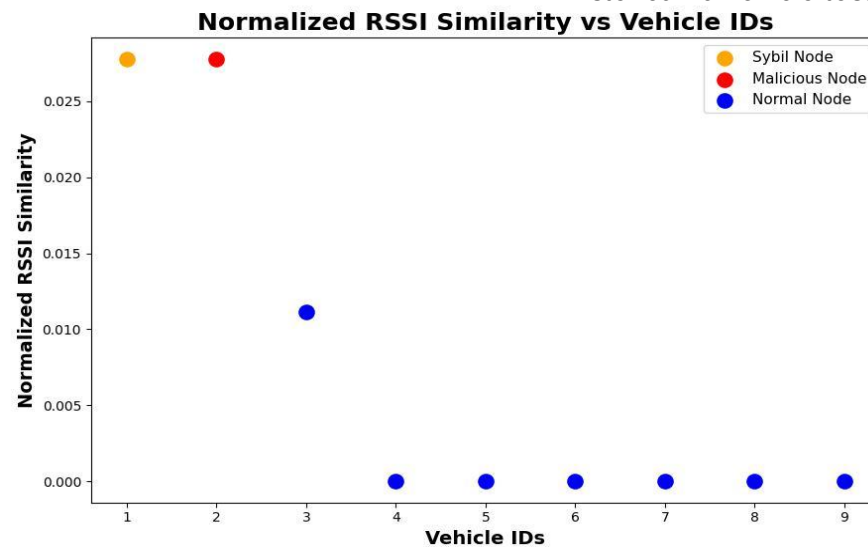


The network topology in VANETs includes 1 malicious node (sybil attacker in red) and 2 illusion (sybil node in orange) and 3 normal vehicular (in blue)

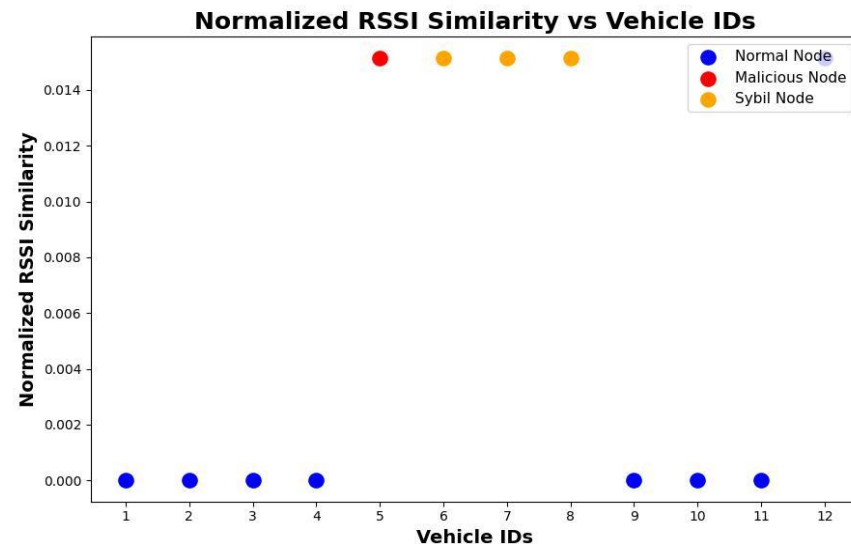
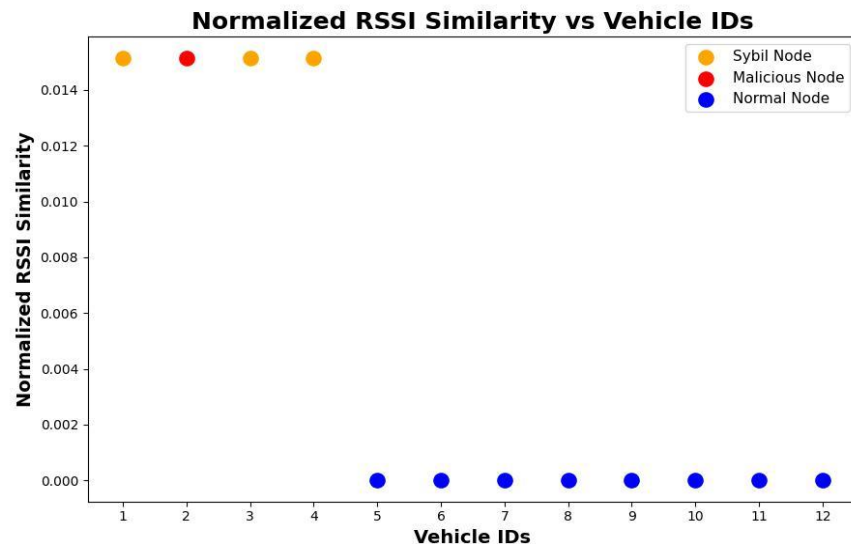
Feature visualization: 5, 6 and 9 vehicles



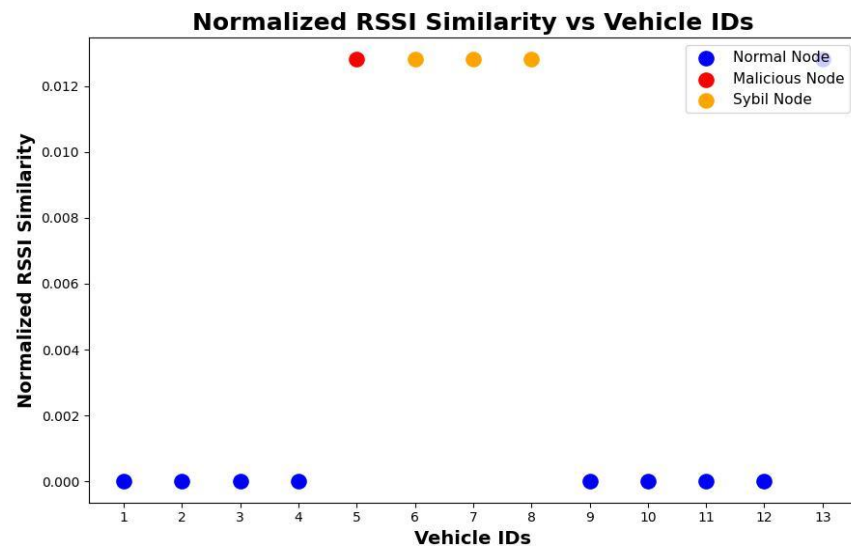
Historical: 6-vehicle test scenario no longer in the active dataset



Feature visualization: 12, 12 and 14 vehicles



Historical: 12-second test scenario removed from the active dataset



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

Confusion Matrix for Random-Forest Classifier

Predicted Class	True Class	
	Positive	Negative
Positive	176	0
Negative	6	8

Confusion Matrix for Gradient-Boost Classifier

Predicted Class	True Class	
	Positive	Negative
Positive	172	4
Negative	0	14

Confusion Matrix for K-Nearest Classifier

Predicted Class	True Class	
	Positive	Negative
Positive	171	5
Negative	0	14

Confusion Matrix for Decision-Tree Classifier

Predicted Class	True Class	
	Positive	Negative
Positive	176	0
Negative	6	8

Comparison and future work

Classifier	True Positives	True Negatives	False Positives	False Negatives	Accuracy	Precision	Recall	F1 Score
Random Forest	8	176	0	6	0.97	1.00	0.57	0.73
KNN	14	171	5	0	0.97	0.74	1.00	0.85
xgBoost	14	172	4	0	0.98	0.78	1.00	0.88
Decision Tree	8	176	0	6	0.97	1.00	0.57	0.73

Key Observations

- Accuracy: All classifiers perform well, with accuracy ranging from 97% to 98%.
- Precision: Varies from 74% (KNN) to 100% (Random Forest / Decision Tree).
- Recall: Varies from 57% (Random Forest / Decision Tree) to 100% (KNN / xgBoost).
- F1 Scores: Range from 0.73 to 0.88; xgBoost has the best score.

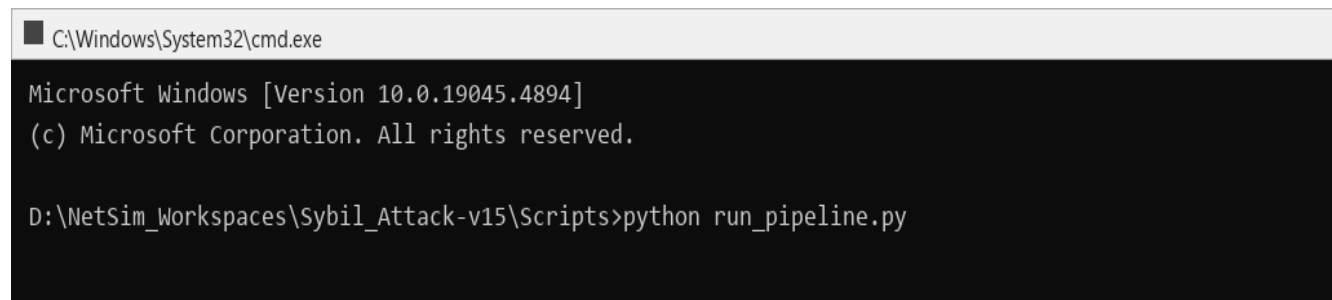
Future Work

- Test with larger networks to assess scalability.
- Experiment with ensemble methods to potentially improve overall performance.
- Impact of different VANET scenarios (e.g., urban vs. highway) on classifier performance.
- Explore ways to improve precision

Appendix: How-to-Guide

How to run the pipeline

- The whole pipeline (feature extraction, ground-truth labeling, train/test merge, classifier training, evaluation, visualization) is one script: `run_pipeline.py`.
- Prerequisite: NetSim must have already produced each scenario's `IEEE_802_11_Radio_Measurements_Log.csv` (i.e. the simulations have been run at least once).
- Open the command prompt and navigate to the Scripts folder.
- Run: `python run_pipeline.py`
- It runs all 6 stages in order and stops immediately, naming the failed stage, if any step fails.
- Outputs land in Results/Predictions, Results/Confusion-Matrix and Results/Feature-Viz, matching the Scripts subfolder that produced them.



```

C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.4894]
(c) Microsoft Corporation. All rights reserved.

D:\NetSim_Workspaces\Sybil_Attack-v15\Scripts>python run_pipeline.py
  
```

How scenarios.py works

- What it does: automatically finds every experiment folder that has NetSim data in it - no one has to type scenario names into a script by hand.
- What it can't do: it can't tell which vehicle you set up as the attacker (malicious/Sybil node) - that's a one-time detail you have to enter yourself, per experiment.
- Where to add it: open Scripts/scenarios.py and add one line with the folder name and the malicious/Sybil vehicle numbers.
- Why this is safe: an experiment folder that hasn't been added yet is still processed, but left out of training and testing - so nothing gets labeled wrong by accident.