Security Issues in VANETs

Submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology

in

Computer Science and Engineering and Business Systems

by

Ben Joseph Mathew

20BBS0014

and

Rupesh Rajan

20BBS0146

Under the guidance of

Dr. Prabin S. M.

School of Computer Science and Engineering,

VIT, Vellore.

May, 2024

DECLARATION

I hereby declare that the thesis entitled “Security Issues in VANETs” submitted by me, for the award of the degree of *Bachelor of Technology in Computer Science and Engineering and Business Systems* to VIT is a record of bona fide work carried out by me under the supervision of Dr. Prabin S. M.

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place	:	Vellore
Date	:	09/05/2024

Signature of the Candidate

CERTIFICATE

This is to certify that the thesis entitled "Security Issues in VANETs" submitted by **Ben Joseph Mathew, 20BBS0014 and Rupesh Rajan, 20BBS0146**, **School of Computer Science and Engineering**, **VIT**, for the award of the degree of *Bachelor of Technology in Computer Science and Engineering and Business Systems*, is a record of bona fide work carried out by them under my supervision during the period, 01. 12. 2023 to 30.04.2024, as per the VIT code of academic and research ethics.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The thesis fulfills the requirements and regulations of the University and in my opinion meets the necessary standards for submission.

Place : Vellore

Date : 09/05/2024 **Signature of the Guide**

Internal Examiner **External Examiner**

Head of the Department

Computer Science and Engineering and Business Systems

ACKNOWLEDGEMENTS

We are deeply grateful to Dr. Prabin. S. M., SCOPE, Vellore Institute of Technology, for his unwavering guidance, encouragement, and patience throughout our endeavor. Our association with her extended beyond academics, providing us with the invaluable opportunity to work alongside an intellectual expert in the field of machine learning.

We extend our heartfelt gratitude to Dr. G. Viswanathan, Chancellor of Vellore Institute of Technology, as well as Mr. Sankar Viswanathan, Dr. Sekar Viswanathan, Mr. G. V. Selvam, Vice-Presidents of VIT, Dr. V. S. Kanchana Bhanskamn, Pro-Vice-Chancellor, Dr. Partha Sarathi Mallick, Pro-Vice-Chancellor and Dr. Ramesh Babu K, Dean of the School of Computer Science and Engineering, for providing an enriching environment and inspiration throughout our course.

We express our sincere thanks to Dr. Rajkumar S., Head of Department/Professor, and all the teaching staff at our university for their enthusiastic support and timely encouragement, which greatly facilitated our successful completion of the course. Additionally, we are grateful to our parents for their unwavering support.

Special thanks go to our friends whose encouragement propelled us to undertake and complete this task. Lastly, we extend our gratitude to all those who directly or indirectly contributed to the successful completion of this project.

Place: Vellore

Date: 09/04/2024

Name of the Students

Ben Joseph Mathew (20BBS0014)
Rupesh Rajan (20BBS0146)

Executive Summary

1. Objectives: To develop a robust security protocol for Vehicular Ad-hoc Networks (VANETs) aimed at detecting and preventing Sybil attacks.

2. Motivation: Sybil attacks pose a significant threat to VANETs, compromising safety and security. Addressing this issue is crucial for ensuring the reliability and effectiveness of VANET communications.

3. Background: Existing security mechanisms in VANETs are insufficient in detecting Sybil attacks, highlighting the need for a novel approach to enhance network security.

4. Project Description: This project aims to fill the research gap by proposing a novel protocol for detecting Sybil attacks in VANETs, addressing the limitations of existing systems.

5. Technical Specification: The protocol will be designed to meet both functional and non-functional requirements, ensuring technical feasibility and compliance with relevant standards and policies.

6. Design Approach: The system architecture will incorporate advanced algorithms and communication protocols to detect and mitigate Sybil attacks effectively. Data flow diagrams, use case diagrams, and sequence diagrams will illustrate the design details.

7. Schedule: A Gantt chart will outline the project timeline, tasks, and milestones, facilitating efficient project management and progress tracking.

8. Project Demonstration: The developed protocol will undergo rigorous testing and evaluation to assess its effectiveness in detecting and preventing Sybil attacks in simulated VANET environments.

9. Summary: This project aims to address the critical security challenges posed by Sybil attacks in VANETs through the development of an innovative security protocol, contributing to the advancement of vehicular communication systems.

List of Figures

List of Abbreviations

VANETs	Vehicular Ad Hoc Network
FPR	False-positive rates
DSRC	Dedicated Short-Range Communications
C-V2X	Cellular Vehicle-to-Everything

INTRODUCTION

1.1. OBJECTIVE

The objective of the project is to create a new fog computing-based method that will increase Vehicular Ad Hoc Network (VANET) security against Sybil attacks. The main goals consist of:

Create a Decentralized System for Detecting Sybil: Develop and put into action a decentralized Sybil detection system to improve the efficiency and scalability of VANETs. Roadside Units (RSUs) and individual On-Board Units (OBUs) will become less dependent on centralized entities since this technique distributes the detection process across numerous network nodes. By utilizing consensus algorithm and distributed computing concepts, it will guarantee system resilience and robustness against hostile actions and enable effective Sybil attack detection.

Boost the effectiveness metrics of the current Sybil detection systems: Reduce processing overhead, false-positive rates (FPR), and processing delays to improve the performance metrics of Sybil detection techniques. Fog computing techniques and optimization algorithms, which distribute processing and data storage closer to the network edge, will be applied to achieve this. The suggested method will raise the overall efficacy and efficiency of Sybil detection on VANETs by reducing processing overhead and delays and optimizing false-positive rates.

Verify Scalability and Efficacy: Confirm the effectiveness and scalability of the suggested method with thorough simulations and assessments in a range of VANET scenarios. The effectiveness of the suggested strategy will be evaluated under a range of circumstances, including varying traffic densities, network topologies, and attack scenarios, using realistic VANET models and scenarios. This research intends to show how well the fog computing-based approach detects and mitigates Sybil attacks in VANETs while maintaining scalability to manage growing network sizes and traffic loads through rigorous simulation and evaluation.

1.2. MOTIVATION

In today's transportation systems, vehicular ad hoc networks, or VANETs, offer a revolutionary paradigm with the potential to transform passenger convenience, traffic efficiency, and road safety. Through real-time communication between vehicles and roadside infrastructure, Vehicle Ad Hoc Networks (VANETs) can potentially alleviate traffic congestion, lower accident rates, and facilitate the development of autonomous driving technology.

However, ongoing security risks limit the execution of this, with Sybil attacks being one especially challenging obstacle. Sybil attacks cause traffic manipulation, denial of service issues, and the spread of misleading information by having a hostile node assume numerous identities at once. These attacks risk road user safety, weaken public confidence in intelligent transportation systems, and compromise the integrity and dependability of VANETs.

The efficacy of current security techniques in VANETs, like intrusion detection systems, authentication, and encryption, in detecting and preventing Sybil attacks, has shown some limitations. Centralized methods that depend on roadside units or historical data have problems with scalability and are vulnerable to advanced adversaries' evasion strategies. Moreover, the dynamic and decentralized characteristics of VANETs make real-time detection more difficult, requiring creative and decentralized methods to deal with this critical security issue.

Inspired by the pressing need to improve VANET security and dependability, our work aims to create new methods for detecting Sybil attacks. Through the utilization of the distinct features of VANETs, such as communication dynamics and vehicle movement patterns, our goal is to develop scalable and resilient solutions that can efficiently identify and defeat Sybil attacks in real time. Our goal is to guarantee the safety and welfare of every road user while advancing intelligent transportation systems through interdisciplinary cooperation and a holistic approach.

1.3. BACKGROUND

Vehicular Ad Hoc Networks, or VANETs, have become a ground-breaking technology that might completely change the way that automobiles interact with roadside infrastructure and with one another in real time. VANETs, which have their roots in the idea of Mobile Ad Hoc Networks (MANETs), enable impromptu and dynamic communication between cars. This communication enables the sharing of traffic information, alerts for potential hazards, and other services that improve road safety, traffic efficiency, and passenger comfort.
VANETs have a lot of potential, but they also have a lot of security issues that jeopardize their dependability and integrity. The Sybil attack phenomenon, in which a hostile node falsely assumes numerous identities within the network, is one of the most well-known security dangers. These malicious nodes have the ability to modify traffic data, interfere with network operations, and endanger the VANET's overall communication integrity by taking advantage of their many identities.

Transportation security and road safety are gravely endangered by sybil attacks. Malicious nodes have the ability to spread misleading traffic information and tamper with traffic lights to create accidents, bottlenecks, and traffic congestion—thus putting the lives of drivers at risk and hindering traffic flow. Furthermore, Sybil attacks erode public confidence in intelligent transportation systems, impeding VANET technologies' broad adoption.

Numerous research projects have been undertaken in an attempt to tackle the problem of Sybil attack detection in VANETs. Conventional methods for detecting Sybil attacks frequently depend on centralized monitoring and verification systems, like Roadside Units (RSUs) or individual On-Board Units (OBUs). However, the scalability problems and single points of failure associated with these centralized techniques limit their usefulness in large-scale VANET deployments.

Researchers have been focusing more on distributed and decentralized methods for detecting Sybil attacks in the past few years. These methods use networked vehicles' collective intelligence and the dynamic nature of VANETs to detect and neutralize Sybil attacks instantly. Through the utilization of modern signal processing techniques, communication dynamics, and vehicle mobility patterns, these decentralized solutions seek to provide scalable and resilient mechanisms that may effectively defend against Sybil assaults in VANETs.

2. PROJECT DESCRIPTION AND GOALS

2.1. Survey on Existing Systems

[1] introduces the FSDV-H (Fog-based Sybil Detection in Vehicular Networks) method (an updated version of FSDV), which addresses the pressing issue of Sybil attacks in Vehicular Ad-hoc Networks (VANETs), fundamental to intelligent transportation systems. Sybil attacks involve malicious or rogue nodes creating fake traffic congestion, leading to severe consequences such as vehicle collisions and traffic rerouting.

To mitigate the limitations of existing techniques, FSDV-H presents a novel fog computing-based approach for Sybil attack detection in VANETs. FSDV-H introduces the concept of a guard node, which dynamically forms a fog layer by leveraging the onboard units (OBUs) of nearby vehicles. The fog formation adapts based on transmission range and neighboring vehicle count, facilitating efficient detection of suspicious nodes.

[2] introduces FSDV, a fog computing-based Sybil attack detection system for Vehicular Ad hoc Networks (VANETs). The system aims to mitigate the presence of rogue nodes causing Sybil attacks, which create fake traffic congestion, potentially leading to collisions. FSDV leverages onboard units (OBUs) from all vehicles to create a dynamic fog for rogue node detection, employing statistical techniques to analyze beacon messages. The system aims to reduce processing delays, overhead, and false-positive rates (FPR) associated with existing Sybil attack detection schemes. Simulation results demonstrate that FSDV achieves lower processing delays (43%), overhead (13%), and FPR (35%) at high vehicle densities compared to state-of-the-art techniques. FSDV's key contributions include the introduction of guard nodes, dynamic fog utilization, and statistical analysis for Sybil detection in VANETs, offering promising improvements in security and performance.
[3] proposes a fog-assisted system named SybilDriver to mitigate Sybil attacks in Vehicular Social Networks (VSNs) used for crowdsourcing applications. Leveraging characteristics of Vehicular Ad-hoc Networks (VANETs) and Online Social Networks (OSNs), the system employs fog computing to process data closer to vehicles and minimize network overhead. Experiments on real-world datasets demonstrate SybilDriver's effectiveness against various Sybil attack strategies, achieving higher performance than existing techniques. The system integrates proximity graphs, community detection algorithms, similarity indices, and machine learning to detect Sybils in VSNs, offering a promising solution to enhance the security of vehicular crowdsourcing applications.

[4] [5] Prior research has explored various approaches to detect Sybil attacks in VANETs, including cryptography, trust scores, past vehicle data, blockchain, and machine learning. However, existing methods often suffer from high processing delay, overhead, and false-positive rates (FPR), particularly in high-density vehicle regions.

[6] presents SecVanet, a provably secure authentication protocol designed for sending emergency events within Vehicular Ad-Hoc Networks (VANETs). It ensures data security by offering an efficient protocol validated through security analysis and proven by the Scyther tool, with lower computational complexity compared to similar schemes.

[7] proposes a machine learning-based approach for blackhole detection in VANETs, aiming to detect and mitigate blackhole attacks. It utilizes a comprehensive dataset containing normal and malicious traffic flows, evaluates various machine learning algorithms, and demonstrates their effectiveness in distinguishing between normal and malicious nodes, thereby enhancing VANET security.

[8] introduces a hierarchical Directed Acyclic Graph (DAG)-based blockchain architecture with a Proof of Reputation consensus algorithm for enhancing the security and trustworthiness of VANETs. The architecture divides the system into local chains based on geographical regions to improve scalability while maintaining a global chain for security, demonstrating exceptional prowess in enhancing network performance metrics over nonsecure protocols.

[9] presents a solution utilizing Unmanned Aerial Vehicles (UAVs) to enhance urgent alert transmission in VANETs, particularly in areas with limited terrestrial coverage. By integrating UAVs as flying relays, the solution bridges communication gaps and ensures early transmission of alert messages following car accidents, effectively improving the safety of VANETs.

[10] introduces a security-based multipath route switching protocol for quality-of-service enhancement in VANETs using the Wiedemann Car-Following Model. The protocol facilitates meticulous selection and switching between secure multi-paths from transmitter to receiver, ensuring both security and QoS requirements are met. Additionally, the protocol utilizes a robust security protocol based on Diffie-Hellman and short authentication string key agreement protocols to enhance secure data delivery in VANETs.

2.2. Research Gap

VANETs have advanced far in recent years, but research into the identification and prevention of Sybil attacks is still in its early stages. Sybil attacks, in which a malicious node assumes numerous identities at once, are a serious risk to VANET security and dependability. Denial of service attacks, traffic manipulation, and the spread of inaccurate data are just a few of the negative effects that might result from these attacks.
The majority of the information that is currently available on VANET security focuses on conventional security measures such intrusion detection systems, encryption, and authentication. Although these systems have proven successful in mitigating specific security risks, they frequently fail to identify Sybil attacks. It can be difficult to distinguish between malicious and honest nodes in VANETs due to their decentralized and dynamic character, particularly when there are no centralized authority or trustworthy historical data available. Furthermore, a lot of the current methods for detecting Sybil attacks rely on monitoring and verification from centralized sources, like Roadside Units (RSUs) or individual On-Board Units (OBUs). However, the scalability problems and single points of failure associated with these centralized techniques limit their usefulness in large-scale VANET deployments. Furthermore, clever attackers might use conventional detection approaches that rely on network signatures or historical data to evade detection.
Testing novel algorithms and techniques in real-world settings is one of the major obstacles to filling the research gap. Real-world experimentation in VANET setups is logistically difficult and financially expensive. Furthermore, current simulation programs like OMNeT++ and SUMO have limits in terms of effectively implementing complex codes and algorithms, even though simulations provide a more affordable option. This makes it more difficult to create and validate new methods for detecting Sybil attacks in real-world VANET environments.
New and decentralized methods for Sybil attack detection in VANETs are desperately needed to close this research gap. To identify and counteract Sybil attacks in real time, these strategies need to take advantage of the special qualities of VANETs, such as communication dynamics and vehicle mobility patterns. Through the development of resilient and expandable methods for Sybil attack identification, scientists can improve the security and dependability of VANETs and facilitate the extensive implementation of intelligent transportation networks.

2.3. Problem Statement

Sybil attacks pose a substantial risk to the security and dependability of VANETs, endangering road safety and transportation efficiency. The existing security procedures implemented in VANETs frequently prove inadequate in accurately identifying and minimizing Sybil attacks. These methods usually depend on centralized authorities or historical data, which can lead to problems with scalability and susceptibility to advanced evasion-tactics.

Furthermore, the ever-changing and distributed nature of VANETs makes it challenging to identify Sybil attacks in real-time. The challenge is further complicated by the presence of vehicular movement patterns, diverse traffic situations, and communication dynamics. Identifying malicious nodes among honest ones while reducing the occurrence of incorrect identifications and delays in processing continues to be a substantial obstacle.
This study aims to tackle these difficulties by creating decentralized and creative methods for detecting Sybil attacks in Vehicular Ad Hoc Networks (VANETs). The goal is to provide strong systems that can identify and prevent Sybil attacks in real-time by utilizing VANET features like vehicle movement patterns and communication dynamics. To address the complex issues of detecting Sybil attacks on VANETs, it is necessary to have interdisciplinary collaboration and adopt an integrated approach.

3. TECHNICAL SPECIFICATION

3.1. Requirements

3.2.1 Functional Requirements

The Guard Node Selection:
The system must accurately detect and recognize vehicles to function as guard nodes and effectively manage interactions between them within a predetermined range of communication.
Guard nodes should be chosen according to specific criteria, such as their capabilities or predetermined qualities.
When many guard nodes intersect, the system evaluates their capabilities to identify the most optimal one.
If it is determined that their capabilities are similar, both nodes are designated as guard nodes.

Suspicious Node Detection:
The system needs to detect suspicious nodes by utilizing predetermined parameters, such as notable disparities in speed compared to adjacent automobiles.

Nodes that raise suspicion should be marked for additional verification.

Suspicious node validation:
If the system is implemented, it should transmit challenge packets to nodes that are deemed suspicious.
The replies from suspect nodes should be validated using trusted nodes or alternative methods.

3.2.1 Non-Functional Requirements

Accuracy:
Guard Node Identification: The algorithm must accurately identify vehicles having the necessary properties to function as guard nodes, guaranteeing that only appropriate vehicles are chosen.
Suspicious Behavior Detection: The algorithm must accurately identify suspicious behavior, such as deviations in vehicle speed or communication patterns, with a high level of precision. This involves the reduction of both false positives (incorrectly classifying normal behavior as suspicious) and false negatives (failing to detect actual suspicious behavior).

Efficiency:
Computational Resources: The algorithm should efficiently utilize computational resources, optimizing memory usage and processor power to conduct large-scale simulations without consuming excessive resources.
Processing time: The algorithm should have quick processing times, regardless of the simulation's size. This guarantees quick detection and reaction to suspicious activity.

Scalability:
The algorithm must be able to handle simulations of different sizes, including various numbers of vehicles and diverse network topologies. It should consistently perform well regardless of the scale of the simulation.
Complex Networks: It must effectively manage complex network configurations, such as intersections, highways, and metropolitan areas, while maintaining optimal performance and accuracy.

Robustness:
Behavioral Variability: The algorithm should be able to withstand and adapt to changes in vehicle behavior, such as fluctuations in speed, alterations in route selection, and

modifications in interaction patterns.
Network Conditions: The system should function efficiently in various network conditions, including variations in communication signal strength or network congestion.
Environmental Factors: The algorithm must consider environmental variables that could affect the accuracy of the simulation, such as weather conditions or road surface conditions.

Security:
Data integrity refers to the algorithm's ability to safeguard simulated data from unauthorized access or modification, which might potentially undermine the accuracy and reliability of the results.
The simulation environment must be safeguarded from external threats or intervention to ensure a secure and controlled setting for experimentation and analysis.

Maintainability:
Codebase Organization: The code should include a well-structured and modular design that promotes ease of maintenance, debugging, and updates.
Documentation: The code should be accompanied by thorough documentation that offers information on functionality, usage, and implementation details. This documentation is essential for supporting continuing maintenance and development activities.
Interoperability refers to the ability of different systems or software to work together and exchange information in a seamless and efficient manner.
Integration with Other Tools: The method should seamlessly integrate with other simulation frameworks or tools, enabling seamless communication and sharing of data between different systems.

Compatibility:
It must be capable of seamlessly integrating with widely used simulation formats and standards, facilitating the ability to operate together with established simulation environments and workflows.

Usability:
User Interface: The system should provide an intuitive interface or configuration options that streamline the process of setting up, configuring, and running simulations.
Accessibility: The system should be easily usable by users with different levels of technical knowledge, offering guidance, tutorials, or assistance resources when necessary to ensure

accessibility.

3.2. Feasibility Study

3.2.1 Technical Feasibility

1. Evaluation of Technology:
Ensuring the compatibility of simulation frameworks, such as SUMO, with the necessary functionalities for algorithm implementation is crucial. Ensuring compatibility with APIs and tools that interface with simulation environments is essential for precise testing and validation.
Performing thorough simulations across several settings and situations is both demanding and essential to completely assess the algorithm's performance. Accessing high-performance computing resources and utilizing effective simulation approaches may be necessary for this.

2. Availability of Resources:
Hardware and Software Resources for Simulations: Evaluating the accessibility of hardware resources, such as computing machines and servers with the capability to execute simulations, is crucial. Moreover, the accessibility of software resources and libraries for algorithm creation is essential.
Vehicle Hardware Requirements: It is crucial to consider the hardware prerequisites for automobiles to engage in the algorithm. This involves assessing whether all vehicles, irrespective of their size or kind (e.g., cars, motorcycles), can support the required hardware and software for algorithmic execution.

3. Skill Set:
Proficiency in VANET Technologies: Proficiency in VANET technologies is necessary for developing and implementing algorithms for vehicular networks. This includes knowledge of communication protocols, security mechanisms, and simulation tools. Engineers possessing expertise in this field are crucial for tackling technological obstacles and enhancing algorithm efficiency.
Continuous Learning and Training: Due to the ever-changing nature of vehicular networks and simulation technologies, it may be important to implement continual learning and training programs to ensure that the team remains up to date with the newest innovations and best practices in the industry.

3.2.2 Economic Feasibility

Cost-Benefit Analysis: Perform a cost-benefit analysis to assess the economic feasibility of implementing the algorithm. Assess the expenses related to the development, execution, and maintenance of the algorithm in relation to the expected advantages, such as enhanced identification of sybil attacks and heightened security in VANETs.

Initial Investment: Assess the initial investment required for building the algorithm and integrating it into existing simulation frameworks or VANET platforms. Take into account costs associated with software development, the incorporation of simulation tools, and the training of workers.

Operational Costs: Calculate the continuous expenses involved in the maintenance and upgrading of the algorithm. This encompasses costs associated with software upgrades, ongoing system surveillance, and the upkeep of infrastructure. Furthermore, it is important to take into account the expenses associated with data storage, processing, and transmission that are necessary for conducting simulations.

Return on Investment (ROI): Calculate the anticipated financial gains from using the algorithm by quantifying the predicted advantages. The advantages may encompass higher security, less vulnerability to sybil attacks, and heightened dependability of VANET communications. Assess the expected profits in relation to the initial and continuous expenses in order to calculate the return on investment (ROI).

Market Analysis: Evaluate the potential and demand in the market for solutions that address security concerns in VANETs. Identify prospective users and stakeholders, including transportation authorities, vehicle manufacturers, and research organizations. Evaluate the market dynamics, regulatory landscape, and competitive climate in order to determine the economic viability of the algorithm.

Risk Assessment: Perform a comprehensive evaluation to identify potential economic risks and uncertainties linked to the deployment of algorithms. Take into account variables such as technological advancements, shifts in market demand, and regulatory obligations. Create contingency plans and mitigation methods to manage potential risks and improve the economic feasibility of the project.

3.2.3 Social Feasibility

Acceptance of VANET technologies: The acceptability and deployment of VANETs as a viable technical solution are of utmost importance. It is crucial to secure recognition and support from regulatory agencies, vehicle manufacturers, and infrastructure providers to effectively adopt VANET technology and systems.

User Acceptance: Evaluate the level of acceptance of the algorithm across different stakeholders, such as automobile manufacturers, transportation agencies, researchers, and the general public. Administer surveys, conduct interviews, or organize focus groups to assess stakeholders' opinions, worries, and choices about the implementation of security algorithms in VANETs.

Confidence and confidentiality: Resolve issues pertaining to trust and privacy among users of VANET. Guarantee that the algorithm safeguards users' privacy while efficiently identifying and reducing security risks. Develop and include measures to ensure safe transmission of information, encryption of data, and protection of user identities in VANET systems, in order to increase user trust and confidence.

Regulatory Compliance: Guarantee adherence to applicable regulations, standards, and guidelines that regulate the implementation of security algorithms in VANETs. Comply with privacy laws, data protection rules, and industry standards to reduce legal and regulatory risks. Engage in cooperation with regulatory authorities and industry stakeholders to advocate for the proper and ethical utilization of security technology in VANET environments.

3.3. System Specification

3.3.1 Hardware Specification

To accommodate the processing demands of vast volumes of satellite telemetry data in real-time, the system is underpinned by a high-performance hardware infrastructure. This includes servers equipped with the latest multi-core processors, enabling parallel data processing and machine learning tasks essential for timely anomaly detection. Data storage solutions are tailored for high-volume data archiving, supporting both the historical data necessary for model training and the real-time data for ongoing analysis. Networking equipment is selected for its capacity to facilitate rapid data transfer across the system components, ensuring that data flow from telemetry sources to analysis servers and user interfaces is seamless and efficient. Additionally, security is paramount, with advanced hardware-based security appliances deployed to protect the system against unauthorized access and ensure data integrity.

3.3.2 Software Specification

At the core of the anomaly detection system is a suite of sophisticated software tools. Machine learning frameworks such as TensorFlow or PyTorch, combined with big data processing platforms like Apache Hadoop or Spark, form the backbone of the data analysis and anomaly detection processes. These are complemented by robust database management systems that efficiently handle the storage, retrieval, and processing of vast datasets. Data visualization tools are integral to the system, offering operators intuitive interfaces through which data can be analyzed and anomalies scrutinized. Ensuring the security of these operations, a range of security software, including encryption and intrusion detection systems, is employed to safeguard data against potential cyber threats.

3.3.3 Standards and Policies

The system’s development and operation are guided by a stringent adherence to industry standards and policies, which assure its compatibility, security, and regulatory compliance. This includes alignment with data protection regulations such as the GDPR and CCPA, ensuring that telemetry data is handled with the utmost confidentiality and integrity. The system also adheres to aerospace industry standards for satellite data communication and management, such as those defined by the CCSDS and ISO 24113, which are critical for ensuring reliable and interoperable satellite operations. In the realm of software development, the system's design and implementation follow recognized standards and best practices, including those outlined by IEEE/ISO, to guarantee software quality, reliability, and maintainability. Security policies and protocols are rigorously applied, encompassing access controls, data encryption standards, and regular security assessments to mitigate cyber risks effectively and maintain system integrity.

4. PROJECT DESCRIPTION AND GOALS

4.1. System Architecture

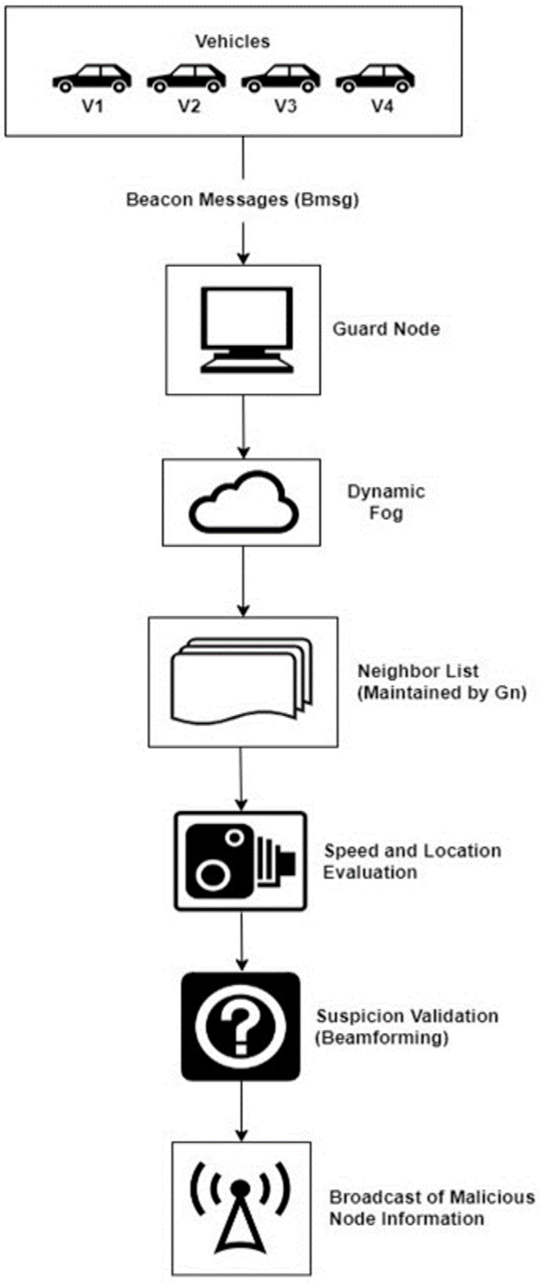

Vehicles
V1
V2
V3
V4
Beacon Messages (Bmsg)
Guard Node
Dynamic Fog
Neighbor List (Maintained by Gn)
Speed and Location Evaluation
Suspicion Validation (Beamforming)
Broadcast of Malicious Node Information

Fig 4.1: System Architecture Diagram

We are proposing a new protocol to detect and prevent Sybil attacks in VANETs. The protocol employs several mechanisms for detecting Sybil attacks in vehicular networks. Here's a breakdown and explanation of its all three steps:

1. Guard Node and Fog Formation:

a. Initial Guard Node Assignment:

   When a vehicle enters the roadways alone, it is initially designated as a guard node.

   This guard node assumes responsibility for network security within its communication range.

b. Encounter and Capability Comparison:

   When a guard node encounters another guard node within its range, they exchange information and compare their capabilities.

   This comparison involves assessing the processing power and other relevant parameters of the encountered nodes.

c. Guard Node Designation:

   If the capabilities of the encountered nodes are similar or comparable, both nodes retain their guard node status.

   This strengthens the security of the network by having multiple guard nodes actively monitoring the area.

d. Node Role Adjustment:

   If one node's capabilities significantly outweigh the other, the stronger node becomes the primary guard node.

   The weaker node transitions to an honest node, contributing to data processing within the network.

e. Formation of Fog:

   The primary guard node, along with a group of honest nodes, forms a cohesive unit known as a fog.

   This fog collectively enhances network security and facilitates efficient data exchange.

f. Continuous Data Transmission:

   Honest nodes within the fog continuously transmit their location and speed data to the primary guard node for processing.

   This data exchange enables real-time monitoring and analysis of traffic conditions within the network.

g. Neighbor List Management:

   The primary guard node maintains a neighbor list, which includes information about nearby vehicles and their statuses.

   This list is continually updated through the reception of beacon messages from new vehicles and periodic timestamp updates.

2. Speed and Density of Vehicles:

a. Node Categorization by Guard Node:

The guard node has the capability to designate surrounding nodes entering the fog into three distinct categories: honest, unknown, and suspicious.

These categorizations aid in the assessment of the trustworthiness and authenticity of the nodes within the network.

b. Criteria for Node Categorization:

Honest Nodes: Nodes confirmed by the guard node through techniques like computer vision or beamforming. These nodes are known to exist and are considered reliable.

Unknown Nodes: Nodes whose existence is acknowledged by their fellow honest nodes yet lack official validation from the guard node. Their status is yet to be fully verified.

Suspicious Nodes: Nodes exhibiting discrepancies in their reported speed and location values compared to nearby honest/unknown nodes. These discrepancies raise concerns about the authenticity of the node's identity.

c. Data Reception and Analysis:

The guard node receives speed and location values from nearby vehicles through beacon messages.

Each beacon message contains the location and speed values reported by a single node.

d. Suspicion Threshold Setting:

When a new node enters the fog, the guard node compares its speed and location values with those of nearby honest/unknown nodes.

Greenshields' model is employed to estimate the location, speed and density of vehicles in the vicinity of the new node. This model assumes an inverse correlation between vehicle speed and density.

If the difference between the reported values exceeds a dynamically set threshold, it raises suspicion of a Sybil attack.

e. Handling Suspicious Nodes:

Nodes flagged as suspicious due to significant discrepancies in their reported data are subjected to further scrutiny.

These suspicious vehicles undergo a validation phase where their identity and intentions are verified to assess the risk of a Sybil attack.

3. Validation of Suspicious Sybil Attacks:

a. Challenge Packet Transmission:

The guard node transmits a challenge packet to the suspicious node within the network.

b. Estimation of Vehicle Location (Optional):

Before sending the challenge packet, the guard node estimates the suspicious vehicle's current location based on the claimed location in its beacon message and propagation delays.

c. Validation Assistance from Honest Nodes:

   The guard node seeks assistance from nearby honest nodes to validate the presence of the new node.

   Honest nodes utilize their own sensor data or relay the challenge packet issued by the guard node.

d. Evaluation of Acknowledgment:

   If no acknowledgment packet is received from the suspicious vehicle within a set timeframe, the guard node classifies it as a potential malicious or Sybil node.

   This determination is based on the absence of response despite the challenge packet transmission.

e. Status Update:

   If an acknowledgment packet is received, the guard node revises the node's status to either honest or unknown, depending on the situation.

   If honest nodes confirm the new node's existence, the guard node updates its status to unknown.

f. Continuation of Guard Node Assessment:

   After the status update, the guard node reassesses whether the new node possesses superior capabilities compared to itself.

   If the new node is honest and more powerful, it assumes the role of the singular guard node.

   Conversely, if the new node is unknown and possesses superior capabilities, both nodes transition to guard nodes.

g. Dynamic and Cyclical Operation:

   The algorithm operates in a dynamic and cyclical manner, continually adapting and enhancing security within the network.

4.2. Design

4.2.1 Data Flow Diagram

We have used the Yourdon and Coad system of symbols to represent our DFDs.

We have included three data flow diagrams corresponding to the three stages of our protocol.

1. Guard Node and Fog Formation

The formation of a fog network begins with the initial assignment of a guard node when a vehicle enters the road alone, responsible for network security within its communication range. As guard nodes encounter each other, they compare capabilities, with similar nodes retaining guard status to strengthen security. If one node significantly surpasses another in capabilities, the stronger becomes the primary guard, while the weaker transitions to an honest node. Together, the primary guard and honest nodes form a fog, enhancing network security and facilitating data exchange. Honest nodes continuously transmit location and speed data to the primary guard for real-time analysis, while the guard node manages a neighbor list, updated with beacon messages and timestamp updates, to monitor nearby vehicles' statuses.

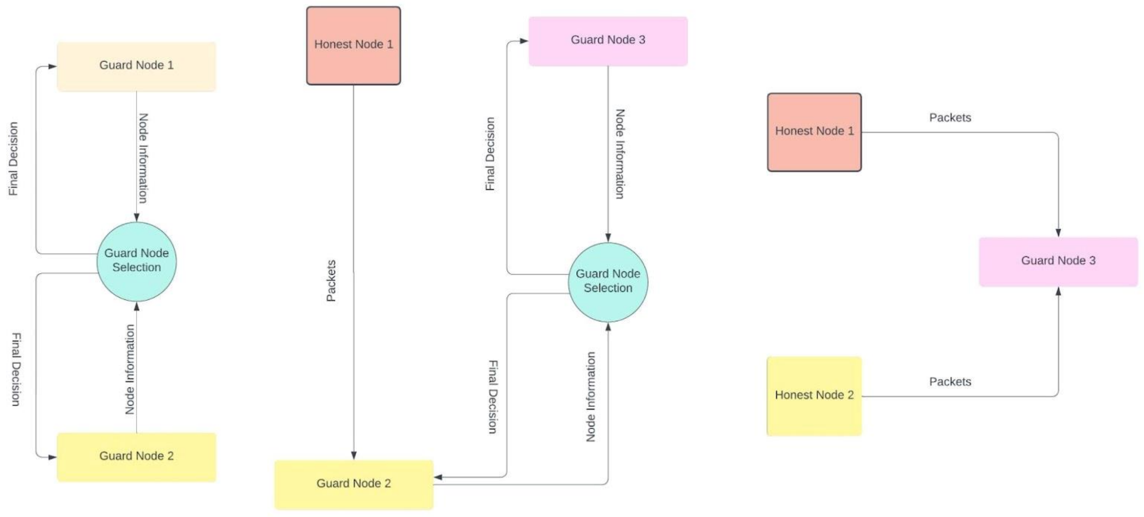


Fig 4.2: Data flow diagram for Guard Node and Fog Formation

2. Speed and Density of Vehicles

The guard node plays a pivotal role in categorizing surrounding nodes entering the fog into three distinct categories: honest, unknown, and suspicious, based on their trustworthiness and authenticity. Honest nodes are confirmed by the guard node and are considered reliable, while unknown nodes lack official validation but are acknowledged by fellow honest nodes. Suspicious nodes exhibit discrepancies in reported speed and location values, raising concerns about their authenticity. The guard node receives speed and location data from nearby vehicles through beacon messages, employing Greenshields’ model to estimate vehicle parameters. If the difference between reported values exceeds a dynamically set threshold, suspicion of a Sybil attack is raised. Suspicious nodes undergo further scrutiny to verify their identity and intentions, mitigating the risk of malicious activity within the network.

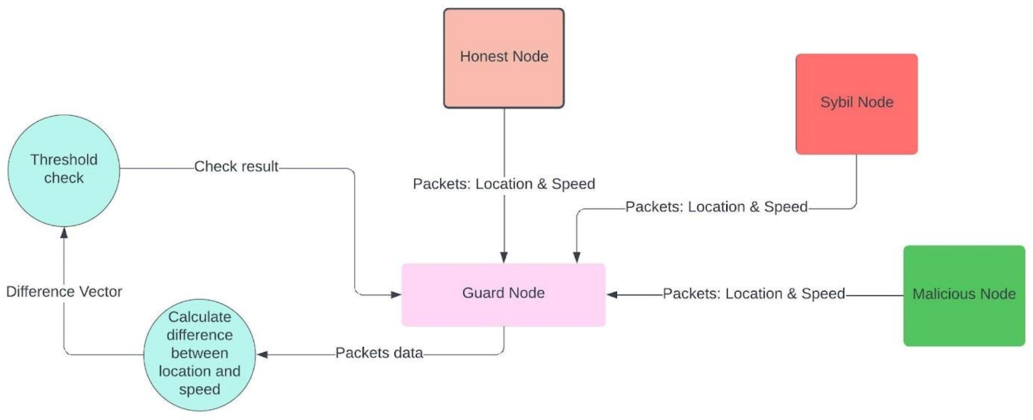

Honest Node
Sybil Node
Threshold check
Check result
Packets: Location & Speed
Packets: Location & Speed
Difference Vector
Guard Node
Packets: Location & Speed
Malicious Node
Calculate difference between location and speed
Packets data

Fig 4.3: Data flow diagram for Speed and Density of Vehicles

3. Validation of Suspicious Sybil Attacks

The challenge packet transmission involves the guard node sending a challenge packet to the suspicious node within the network. Optionally, before transmission, the guard node estimates the suspicious vehicle's location based on its claimed location and propagation delays. Additionally, the guard node seeks validation assistance from nearby honest nodes, either through their own sensor data or by relaying the challenge packet. Upon evaluating the acknowledgment, if no response is received within a specified timeframe, the guard node flags the node as potentially malicious. Conversely, if an acknowledgment is received, the guard node updates the node's status accordingly. Subsequently, the guard node reassesses the new node's capabilities, determining whether it should assume the role of the singular guard node or if both nodes should transition to guard nodes. This process operates dynamically and cyclically, continuously adapting to enhance network security.

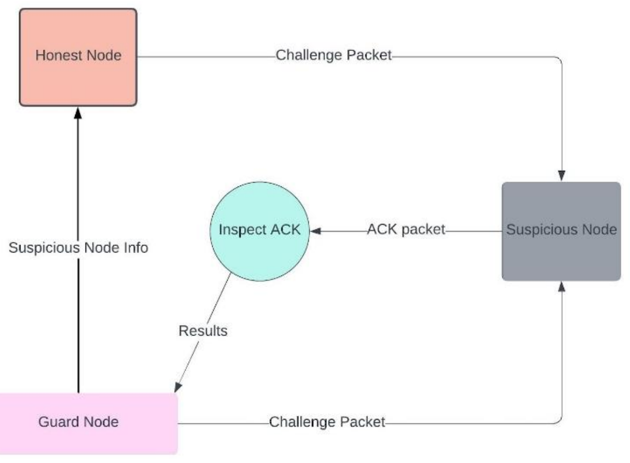


Fig 4.4: Data flow diagram for Validation of Suspicious Sybil Attacks

4.2.2 Use Case Diagram

a. Use Cases:

Formation of Fog Network: This encompasses the process of initializing and establishing a fog network, including the assignment of guard nodes, comparison of capabilities, and formation of the fog.

Node Categorization: Involves categorizing surrounding nodes into honest, unknown, and suspicious categories based on their trustworthiness and reported data.

Validation of Suspicious Nodes: Focuses on validating suspicious nodes to mitigate the risk of malicious activity within the network.

b. Actors:

Guard Node: The central actor responsible for network security and management, including assigning roles to surrounding nodes, receiving and analyzing data, and transmitting challenge packets.

Honest Nodes: Nodes confirmed by the guard node and continuously transmit data for analysis.

Unknown Nodes: Nodes acknowledged by fellow honest nodes but lack official validation from the guard node.

Suspicious Nodes: Nodes exhibiting discrepancies in reported data, requiring further scrutiny to verify their authenticity.

c. Systems:

Fog Network: The overarching system comprising guard nodes, honest nodes, and other networked vehicles.

Beacon Messaging System: Facilitates the transmission of speed and location data between vehicles within the network.

Node Categorization System: Manages the categorization of nodes into honest, unknown, and suspicious categories based on reported data.

Validation System: Responsible for validating suspicious nodes through the transmission of challenge packets and evaluation of responses.

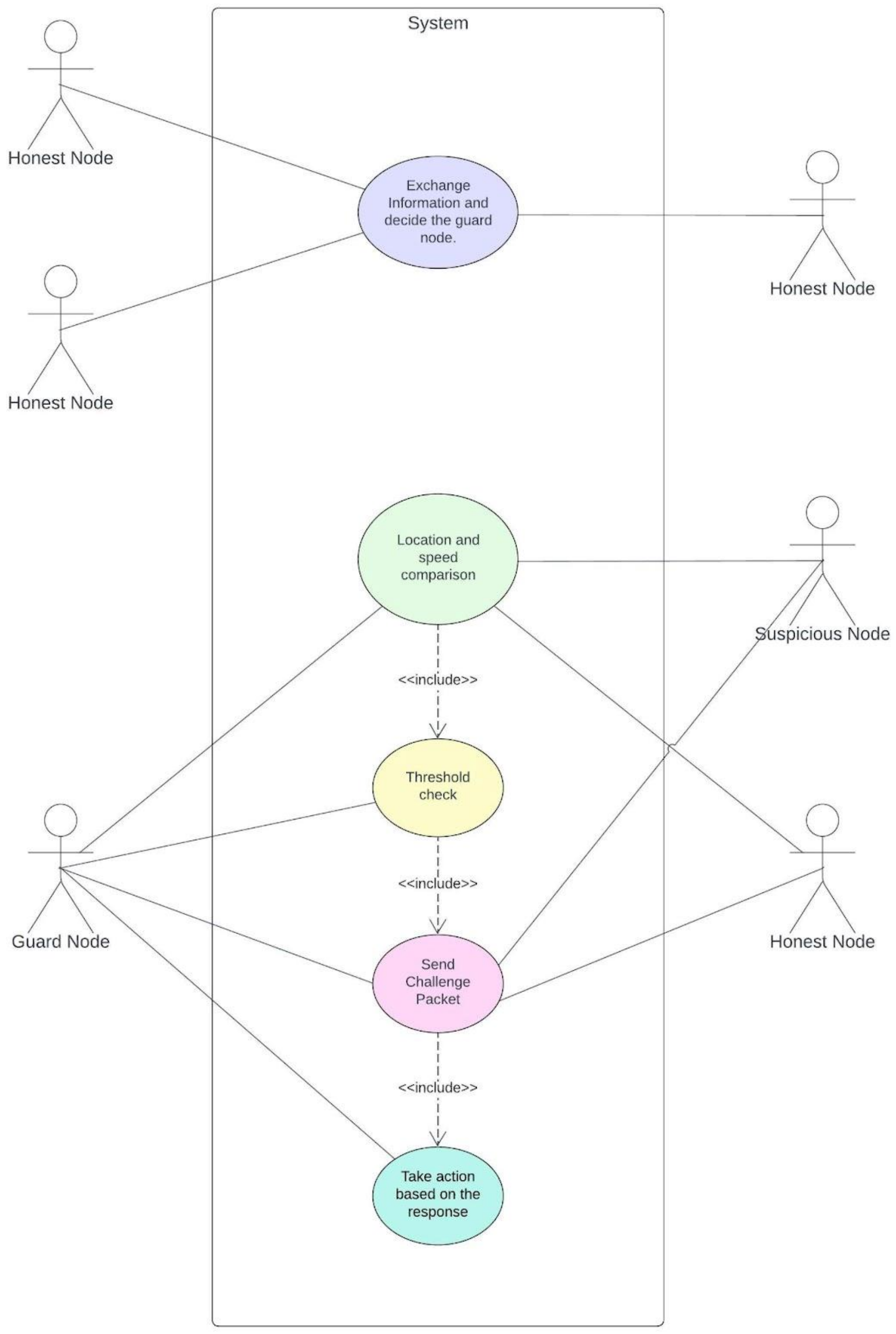
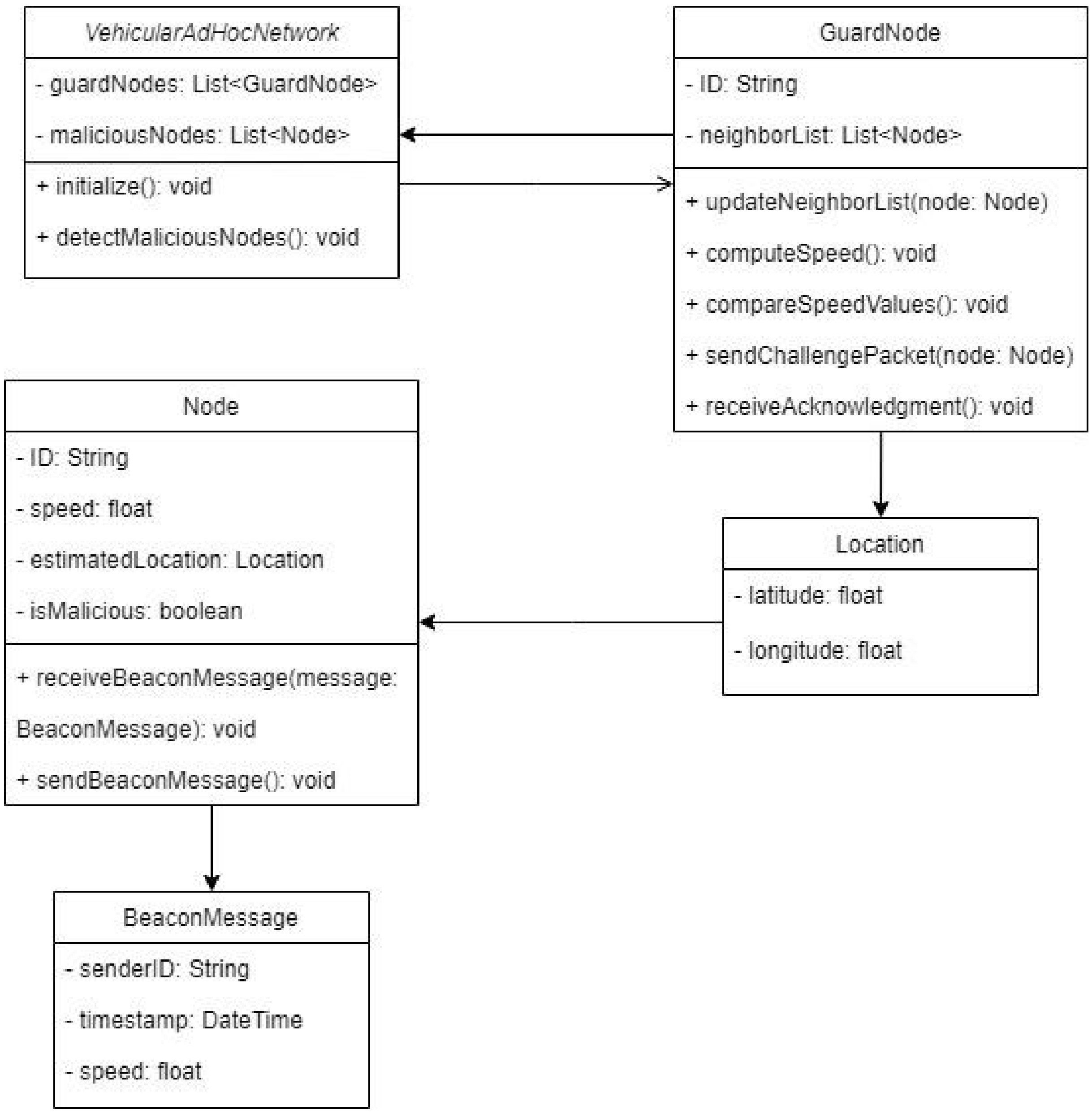


Fig 4.5: Use Case Diagram

4.2.3 Class Diagram



VehicularAdHocNetwork
- guardNodes: List<GuardNode>
- maliciousNodes: List<Node>
+ initialize(): void
+ detectMaliciousNodes(): void
GuardNode
- ID: String
- neighborList: List<Node>
+ updateNeighborList(node: Node)
+ computeSpeed(): void
+ compareSpeedValues(): void
+ sendChallengePacket(node: Node)
+ receiveAcknowledgment(): void
Node
- ID: String
- speed: float
- estimatedLocation: Location
- isMalicious: boolean
+ receiveBeaconMessage(message: BeaconMessage): void
+ sendBeaconMessage(): void
Location
- latitude: float
- longitude: float
BeaconMessage
- senderID: String
- timestamp: DateTime
- speed: float

Fig 4.6: Class Diagram

4.2.4 Sequence Diagram

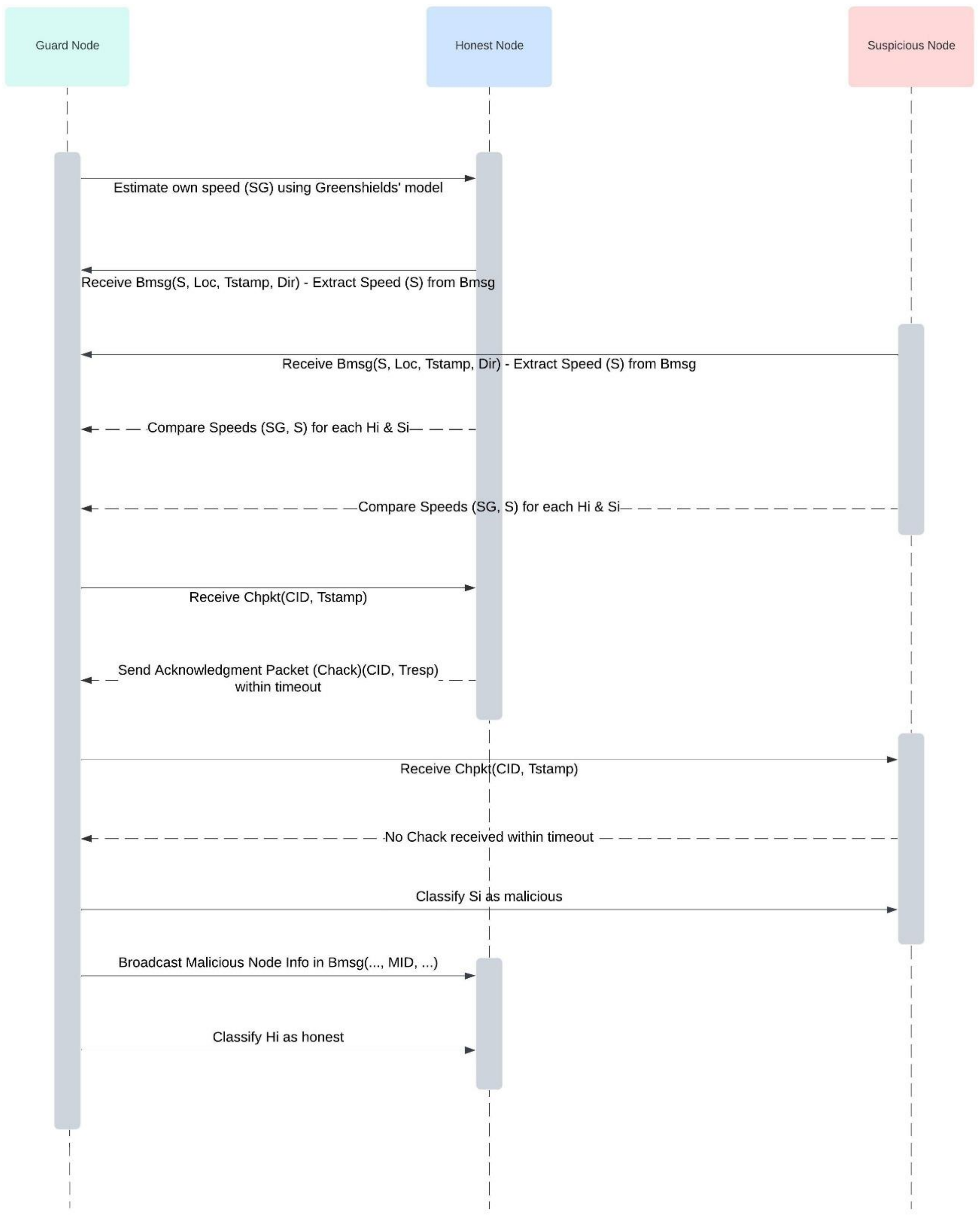

Guard Node
Honest Node
Suspicious Node
Estimate own speed (SG) using Greenshields' model
Receive Bmsg(S, Loc, Tstamp, Dir) - Extract Speed (S) from Bmsg
Receive Bmsg(S, Loc, Tstamp, Dir) - Extract Speed (S) from Bmsg
Compare Speeds (SG, S) for each Hi & Si
Compare Speeds (SG, S) for each Hi & Si
Receive Chpkt(CID, Tstamp)
Send Acknowledgment Packet (Chack)(CID, Tresp) within timeout
Receive Chpkt(CID, Tstamp)
No Chack received within timeout
Classify Si as malicious
Broadcast Malicious Node Info in Bmsg(..., MID, ...)
Classify Hi as honest

Fig 4.7: Sequence Diagram

4.3. Constraints, Alternatives and Tradeoffs

Constraints:

Technical Constraints: Our research is primarily limited by the existing Sybil detection

approaches, especially when applied to VANETs. Traditional methods face challenges in accurately identifying and mitigating Sybil attacks because of the constantly changing and wireless characteristics of the network environment. In addition, these approaches frequently result in significant processing delays, additional costs, and a high occurrence of false-positive rates (FPR), particularly in areas with a large number of vehicles.
Resource constraints: It refer to limitations in computational resources and network bandwidth that can hinder the implementation and testing of new Sybil detection systems. To successfully implement and test new methods in real-world VANET scenarios, it is essential to have adequate computing resources, network infrastructure, and vehicular data for the purposeofvalidationandassessment.

Alternatives:
Methodological Alternatives: When considering alternatives, different approaches for detecting Sybil attacks in VANETs were examined. These options consist of conventional centralized methods that depend on Roadside Units (RSUs) or individual On-Board Units (OBUs), as well as more decentralized and distributed alternatives that utilize dynamic fog computing and beamforming techniques. Every solution has unique benefits and compromises in terms of scalability, efficiency, and resistance to Sybil attacks.

Technology Alternatives: Moreover, other methodologies for identifying Sybil attacks were assessed, encompassing various machine learning algorithms for detecting anomalies, such as Support Vector Machines (SVMs), Random Forests, and Neural Networks. In addition, different communication protocols and hardware configurations, such as Dedicated Short-Range Communications (DSRC) and Cellular Vehicle-to-Everything (C-V2X), were evaluated to determine their appropriateness for detecting Sybil attacks in Vehicular Ad-Hoc Networks(VANETs).

Feedback:

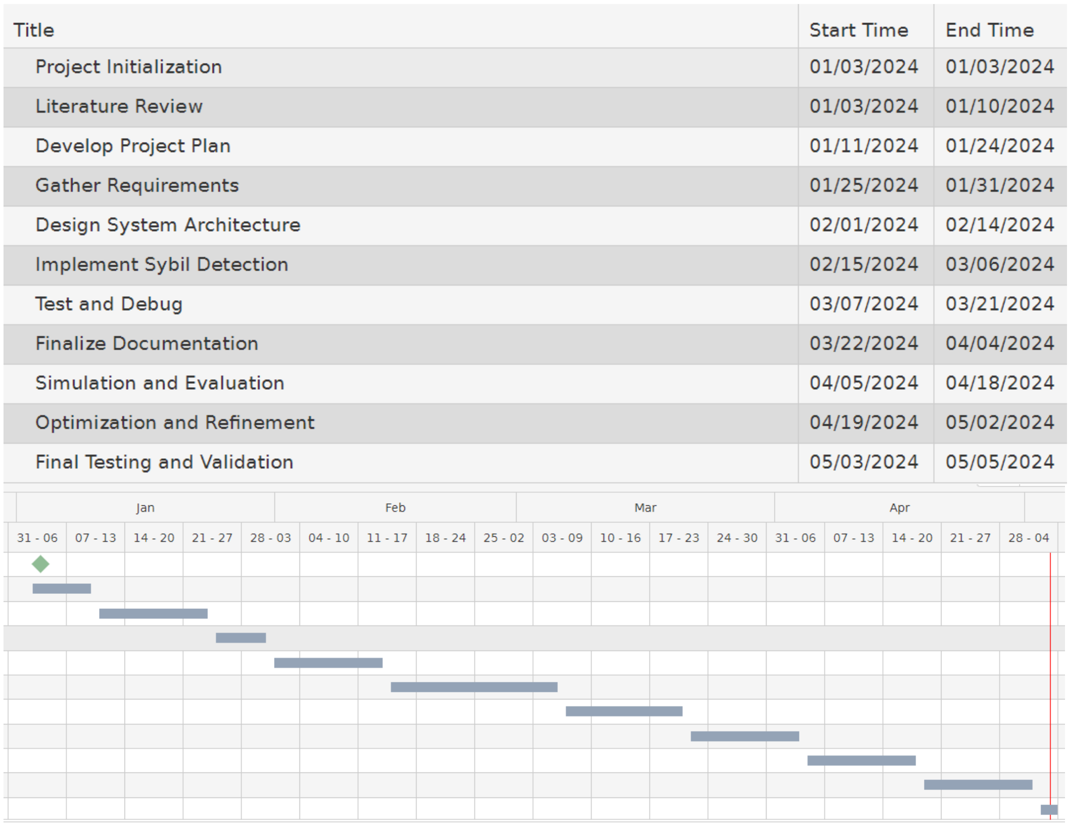
Initial Feedback: The initial input from stakeholders and domain experts emphasized the crucial need to create inventive and effective methods for detecting Sybil attacks in order to safeguard the security and reliability of VANETs. Furthermore, it was unanimously agreed that it is necessary to thoroughly validate and evaluate the proposed methods by conducting realistic simulations and field experiments. This is to showcase their effectiveness and capacity to be scaled up in various VANET scenarios.

Stakeholder Engagement:

Sustained involvement with stakeholders, such as government agencies, car manufacturers, and academia researchers, will be crucial throughout the study process. Collaboration and sharing of information will assist in confirming research results, recognizing practical difficulties, and promoting the implementation of advanced Sybil detection methods in real-world VANET deployments.

5. SCHEDULE, TASKS AND MILESTONES

5.1. Gantt Chart



Title	Start Time	End Time
Project Initialization	01/03/2024	01/03/2024
Literature Review	01/03/2024	01/10/2024
Develop Project Plan	01/11/2024	01/24/2024
Gather Requirements	01/25/2024	01/31/2024
Design System Architecture	02/01/2024	02/14/2024
Implement Sybil Detection	02/15/2024	03/06/2024
Test and Debug	03/07/2024	03/21/2024
Finalize Documentation	03/22/2024	04/04/2024
Simulation and Evaluation	04/05/2024	04/18/2024
Optimization and Refinement	04/19/2024	05/02/2024
Final Testing and Validation	05/03/2024	05/05/2024

Fig 5.1: Gantt Chart

6. PROJECT DEMONSTRATION

To test our algorithm, we needed to implement and test our protocol extensively in some simulation software like OMNeT++ (Objective Modular Network Testbed in C++) or SUMO (Simulation of Urban MObility).

Unfortunately, the task became daunting quickly. We had no idea how to set up and configure these simulators themselves let alone for our particular project. There were no online tutorials available we could refer to. This part is all about what we tried to do in the simulation part. We first downloaded a map of our desired area in a format called OpenStreetMap XML Data (.osm). This file contains detailed geographical information like roads and intersections.

We installed the SUMO suite of tools to start working on the simulation.
Then, we used netconvert, a command-line tool within the SUMO simulation suite used to convert map data from various formats, to convert the .osm file into SUMO's native network format (.net file).
Converting the .osm file into SUMO's native network format (.net file) using the netconvert tool is necessary because SUMO requires a specific network representation optimized for traffic simulation. This conversion ensures that the data is structured appropriately with relevant traffic parameters such as lane geometry and junction types, allowing for accurate simulation results. Additionally, it enables seamless integration with SUMO's suite of simulation tools, facilitating efficient analysis and visualization of traffic scenarios.

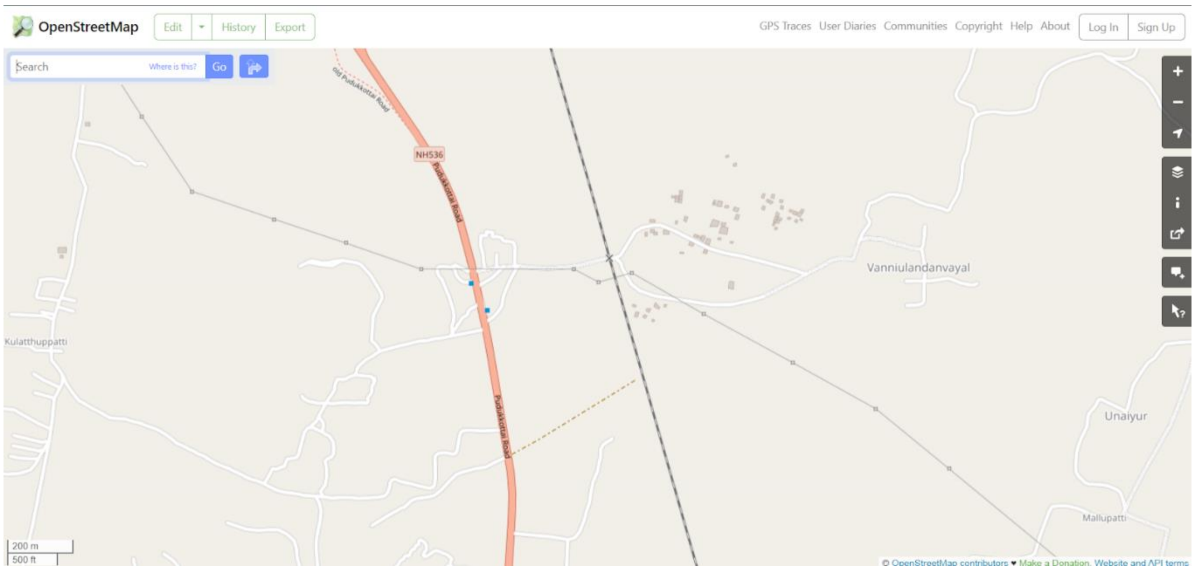
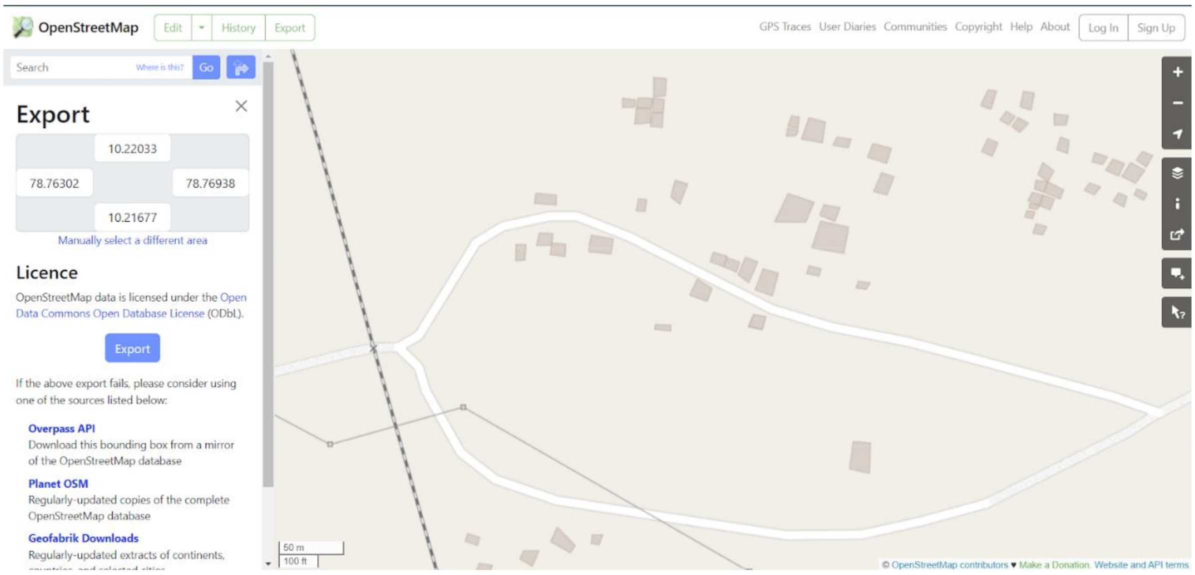


Fig 6.1: OpenStreetMap website



Export

10.22033

78.76302

78.76938

10.21677

Manually select a different area

Licence

OpenStreetMap data is licensed under the Open Data Commons Open Database License (ODbL).

Export

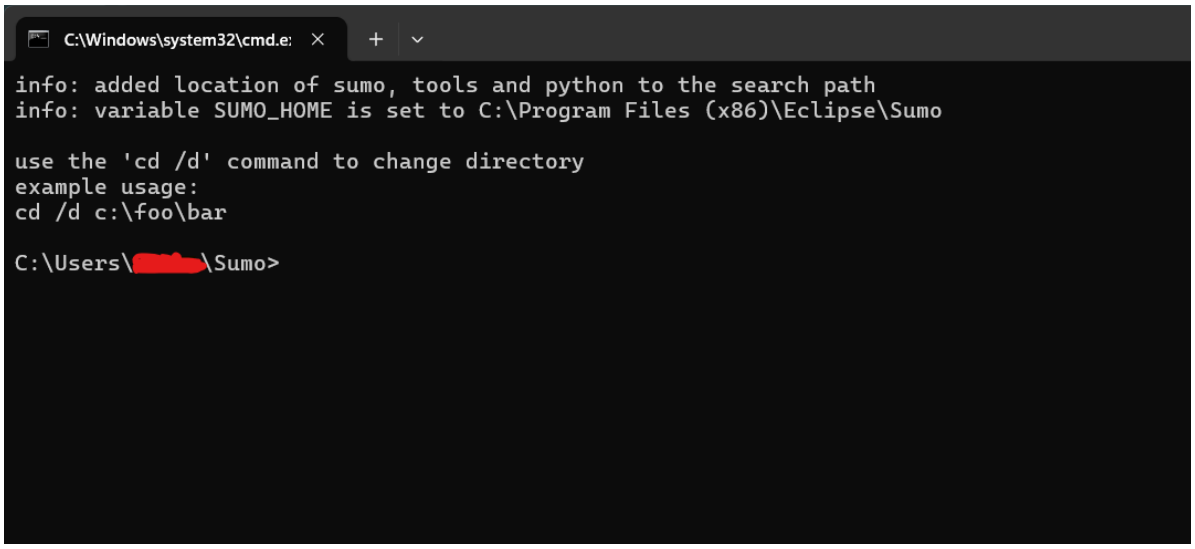
If the above export fails, please consider using one of the sources listed below:

Overpass API
Download this bounding box from a mirror of the OpenStreetMap database

Planet OSM
Regularly-updated copies of the complete OpenStreetMap database

Geofabrik Downloads
Regularly-updated extracts of continents,

Fig 6.2: OpenStreetMap XML Data (.osm) export and download

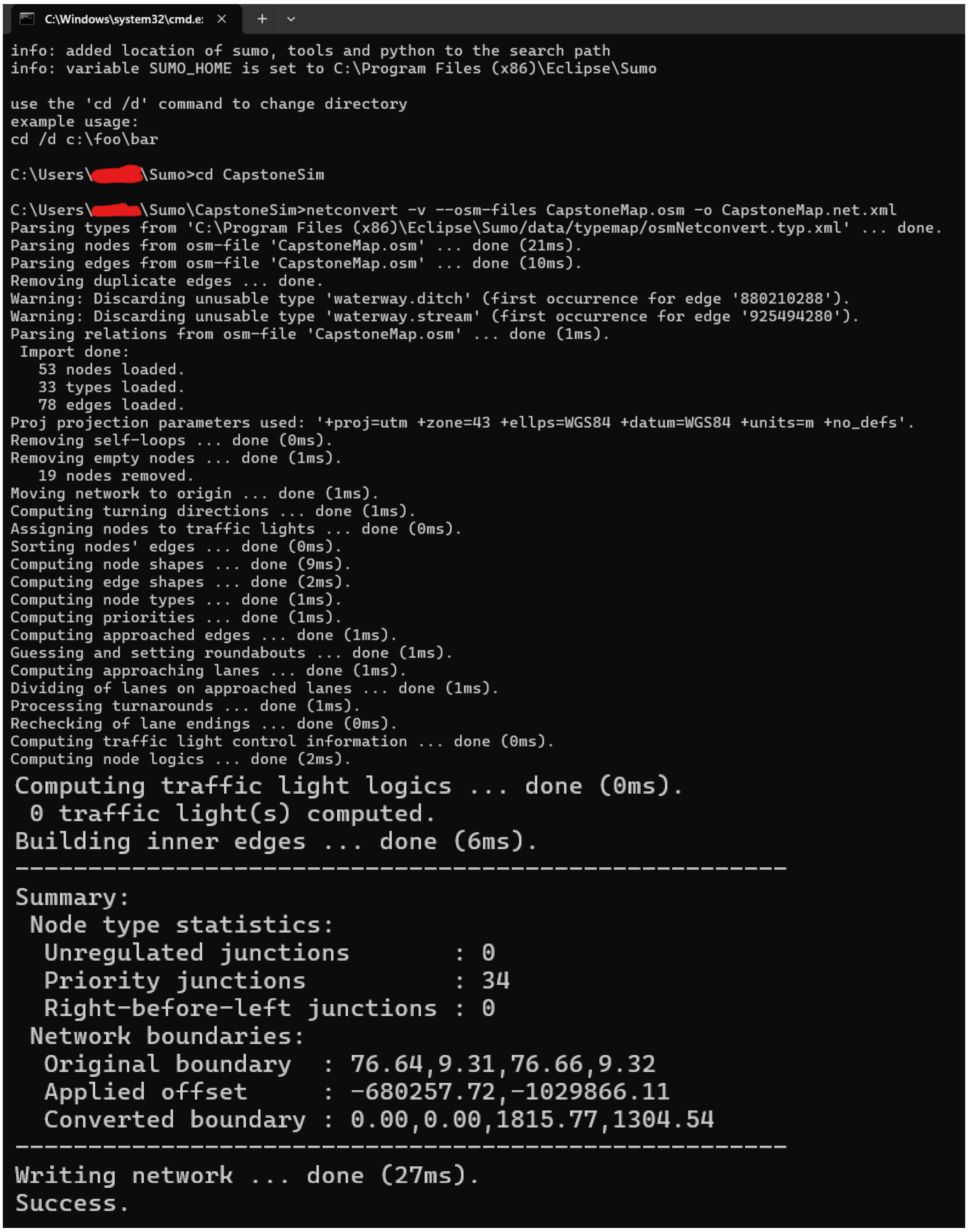


```
info: added location of sumo, tools and python to the search path
info: variable SUMO_HOME is set to C:\Program Files (x86)\Eclipse\Sumo

use the 'cd /d' command to change directory
example usage:
cd /d c:\foo\bar

C:\Users\[redacted]\Sumo>
```

Fig 6.3: SUMO Command-Line Interface

Then, we used netconvert, a command-line tool within the SUMO simulation suite used to convert map data from various formats, to convert the .osm file into SUMO's native network format (.net file).

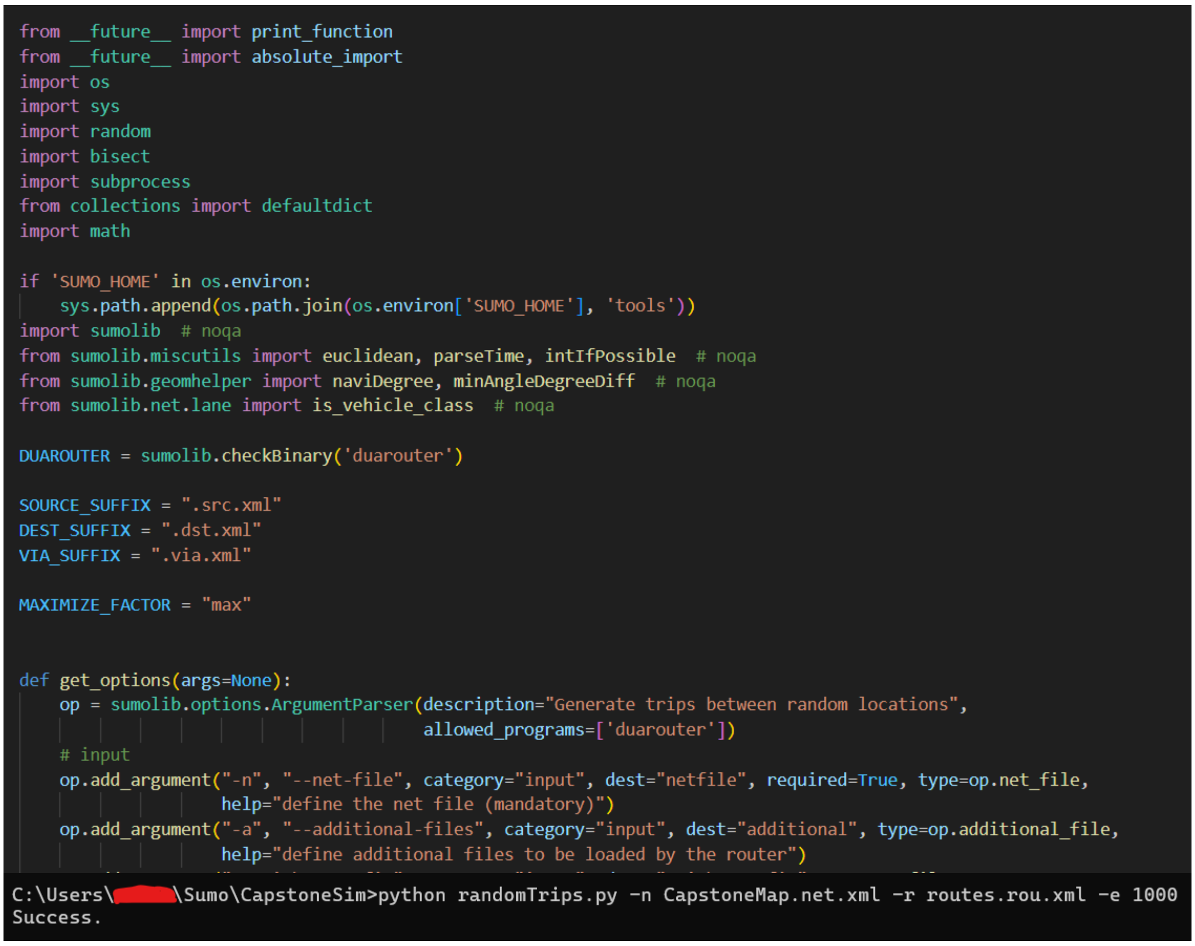
Converting the .osm file into SUMO's native network format (.net file) using the netconvert tool is necessary because SUMO requires a specific network representation optimized for traffic simulation. This conversion ensures that the data is structured appropriately with relevant traffic parameters such as lane geometry and junction types, allowing for accurate simulation results. Additionally, it enables seamless integration with SUMO's suite of simulation tools, facilitating efficient analysis and visualization of traffic scenarios.



```
info: added location of sumo, tools and python to the search path
info: variable SUMO_HOME is set to C:\Program Files (x86)\Eclipse\Sumo

use the 'cd /d' command to change directory
example usage:
cd /d c:\foo\bar

C:\Users\[redacted]\Sumo>cd CapstoneSim

C:\Users\[redacted]\Sumo\CapstoneSim>netconvert -v --osm-files CapstoneMap.osm -o CapstoneMap.net.xml
Parsing types from 'C:\Program Files (x86)\Eclipse\Sumo/data/typemap/osmNetconvert.typ.xml' ... done.
Parsing nodes from osm-file 'CapstoneMap.osm' ... done (21ms).
Parsing edges from osm-file 'CapstoneMap.osm' ... done (10ms).
Removing duplicate edges ... done.
Warning: Discarding unusable type 'waterway.ditch' (first occurrence for edge '880210288').
Warning: Discarding unusable type 'waterway.stream' (first occurrence for edge '925494280').
Parsing relations from osm-file 'CapstoneMap.osm' ... done (1ms).
 Import done:
   53 nodes loaded.
   33 types loaded.
   78 edges loaded.
Proj projection parameters used: '+proj=utm +zone=43 +ellps=WGS84 +datum=WGS84 +units=m +no_defs'.
Removing self-loops ... done (0ms).
Removing empty nodes ... done (1ms).
   19 nodes removed.
Moving network to origin ... done (1ms).
Computing turning directions ... done (1ms).
Assigning nodes to traffic lights ... done (0ms).
Sorting nodes' edges ... done (0ms).
Computing node shapes ... done (9ms).
Computing edge shapes ... done (2ms).
Computing node types ... done (1ms).
Computing priorities ... done (1ms).
Computing approached edges ... done (1ms).
Guessing and setting roundabouts ... done (1ms).
Computing approaching lanes ... done (1ms).
Dividing of lanes on approached lanes ... done (1ms).
Processing turnarounds ... done (1ms).
Rechecking of lane endings ... done (0ms).
Computing traffic light control information ... done (0ms).
Computing node logics ... done (2ms).
Computing traffic light logics ... done (0ms).
 0 traffic light(s) computed.
Building inner edges ... done (6ms).
-----------------------------------------------------
Summary:
 Node type statistics:
  Unregulated junctions       : 0
  Priority junctions          : 34
  Right-before-left junctions : 0
 Network boundaries:
  Original boundary  : 76.64,9.31,76.66,9.32
  Applied offset     : -680257.72,-1029866.11
  Converted boundary : 0.00,0.00,1815.77,1304.54
-----------------------------------------------------
Writing network ... done (27ms).
Success.
```

Fig 6.4: Converting the .osm file into SUMO's native network format (.net file) using the netconvert tool

Then, we can use the randomTrips.py script provided by SUMO to generate vehicle routes automatically based on certain criteria such as the number of vehicles, trip duration, and distribution patterns.



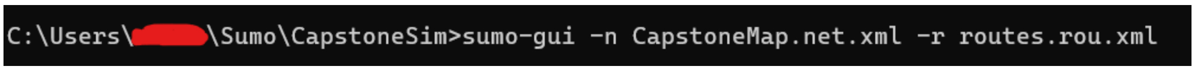
```
from __future__ import print_function
from __future__ import absolute_import
import os
import sys
import random
import bisect
import subprocess
from collections import defaultdict
import math

if 'SUMO_HOME' in os.environ:
    sys.path.append(os.path.join(os.environ['SUMO_HOME'], 'tools'))
import sumolib  # noqa
from sumolib.miscutils import euclidean, parseTime, intIfPossible  # noqa
from sumolib.geomhelper import naviDegree, minAngleDegreeDiff  # noqa
from sumolib.net.lane import is_vehicle_class  # noqa

DUAROUTER = sumolib.checkBinary('duarouter')

SOURCE_SUFFIX = ".src.xml"
DEST_SUFFIX = ".dst.xml"
VIA_SUFFIX = ".via.xml"

MAXIMIZE_FACTOR = "max"


def get_options(args=None):
    op = sumolib.options.ArgumentParser(description="Generate trips between random locations",
                                        allowed_programs=['duarouter'])
    # input
    op.add_argument("-n", "--net-file", category="input", dest="netfile", required=True, type=op.net_file,
                    help="define the net file (mandatory)")
    op.add_argument("-a", "--additional-files", category="input", dest="additional", type=op.additional_file,
                    help="define additional files to be loaded by the router")
```

```
C:\Users\[redacted]\Sumo\CapstoneSim>python randomTrips.py -n CapstoneMap.net.xml -r routes.rou.xml -e 1000
Success.
```

Fig 6.5: RandomTrips.py in action

We can customize the generation process by specifying parameters such as the total number of vehicles to generate (-e), the number of vehicles per hour (-p), the duration of the simulation (-t), and the type of distribution (-b, -u, -n) for trip start times.

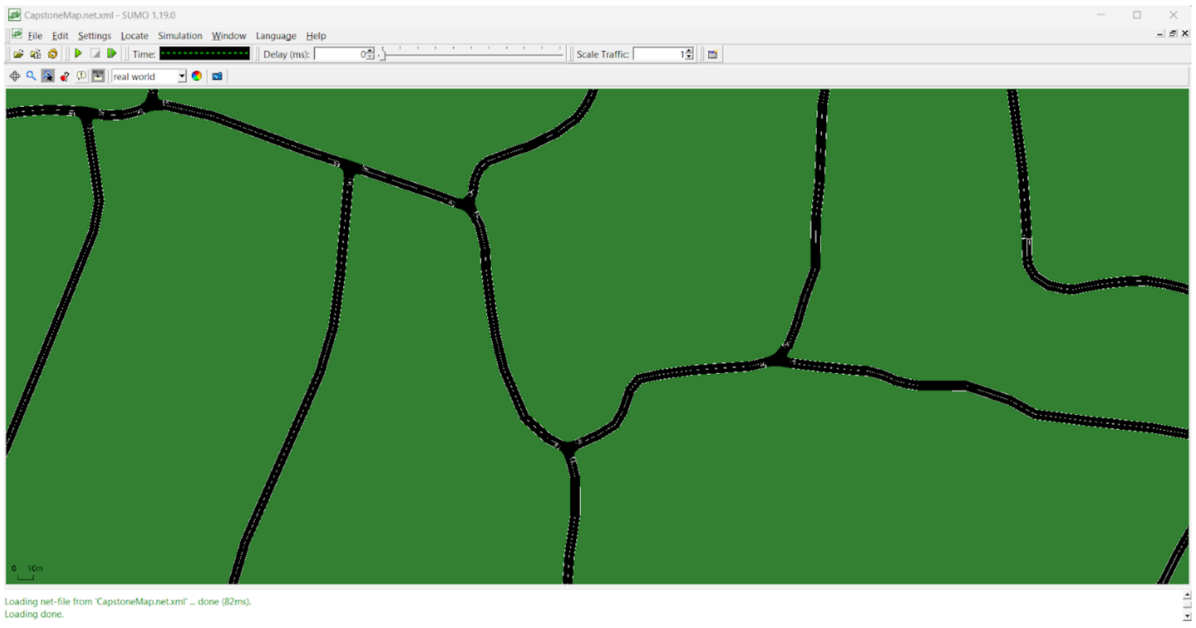
The routes.rou.xml file containing the generated routes and schedules will be created in the same directory where you executed the script. You can then use this file and the .net file as input to your SUMO simulation by running:



```
C:\Users\[redacted]\Sumo\CapstoneSim>sumo-gui -n CapstoneMap.net.xml -r routes.rou.xml
```

Fig 6.6: Opening SUMO-GUI

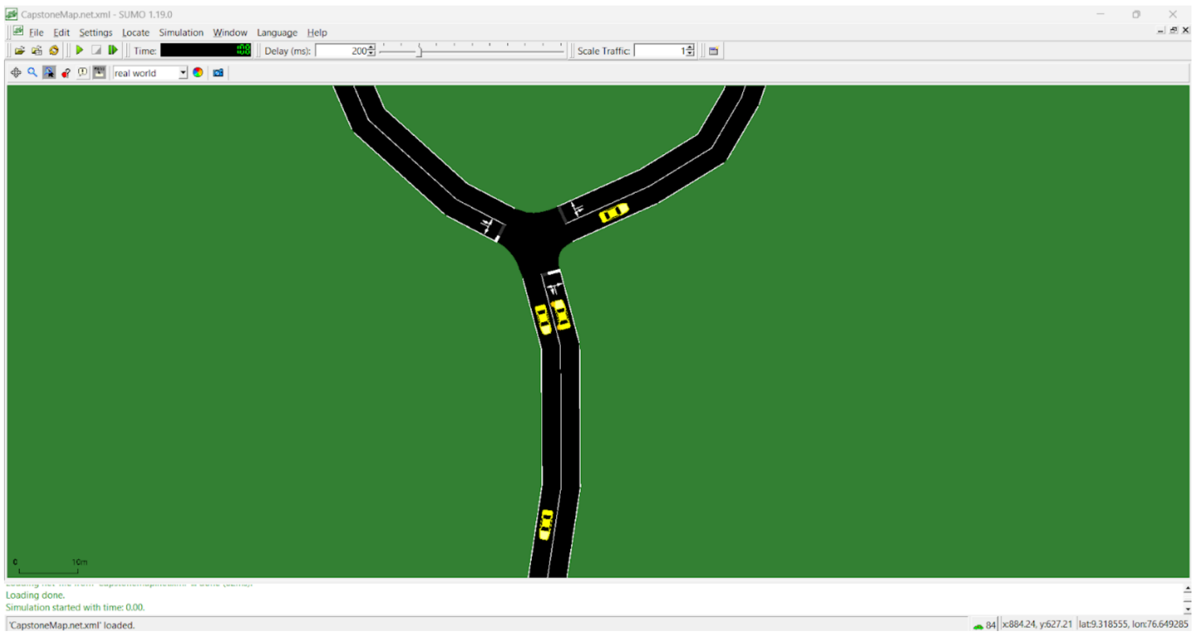
This opens the SUMO-GUI:


CapstoneMap.net.xml - SUMO 1.19.0
File Edit Settings Locate Simulation Window Language Help
Time:
Delay (ms): 0
Scale Traffic: 1
real world
0 10m

```
Loading net-file from 'CapstoneMap.net.xml' ... done (82ms).
Loading done.
```

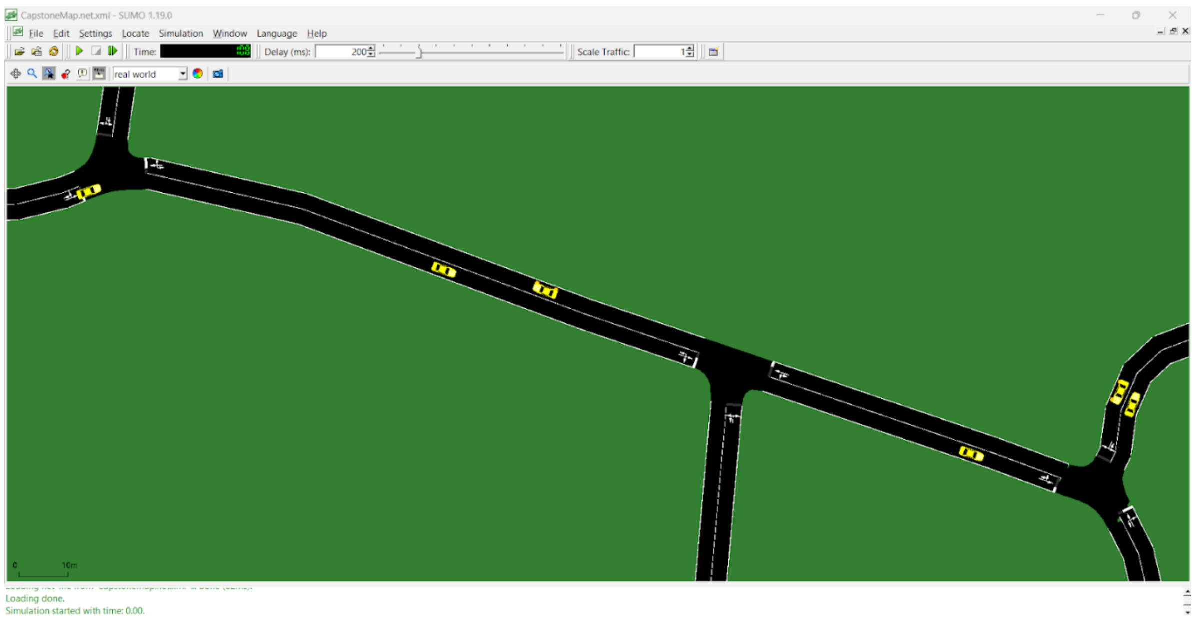
Fig 6.7: SUMO-GUI

We could simulate a VANET in this SUMO as follows:


CapstoneMap.net.xml - SUMO 1.19.0
File Edit Settings Locate Simulation Window Language Help
Time:
Delay (ms): 200
Scale Traffic: 1
real world
0 10m

```
Loading done.
Simulation started with time: 0.00.
'CapstoneMap.net.xml' loaded.    84  x:884.24, y:627.21  lat:9.318555, lon:76.649285
```

Fig 6.8: Simulating VANETs in SUMO-GUI

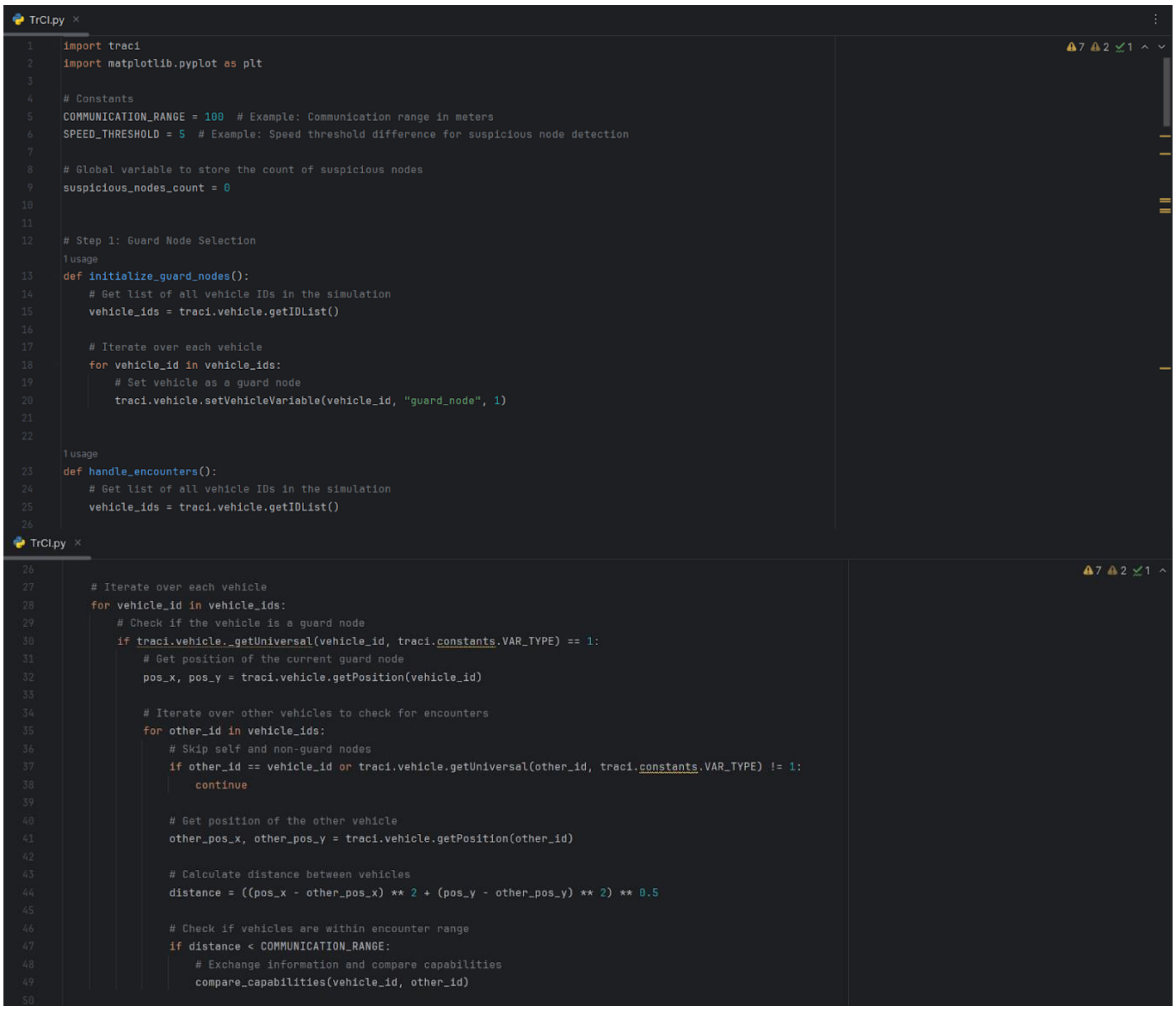


```
Loading done.
Simulation started with time: 0.00.
```

Fig 6.9: Another Simulation of VANETs in SUMO-GUI

Now, to integrate our protocol into SUMO and assess its effectiveness in detecting Sybil attacks in VANETs, we had to follow the following steps:

- Custom Application Development: Create a custom application using the SUMO API or TraCI. This application will interact with SUMO to access vehicle data, control their behavior, and detect Sybil attacks.
- TraCI Integration: Utilize TraCI to communicate with SUMO in real-time. This allows your application to monitor the simulation, send commands to vehicles, and receive sensor data for decision-making.
- Simulation Setup: Configure your simulation scenario within SUMO, defining the road network, vehicle types, and traffic patterns. Ensure the scenario includes conditions conducive to Sybil attacks, such as areas with limited visibility or high traffic density.
- Testing and Evaluation: Run the simulation with your integrated algorithm and analyze the results. Evaluate its performance in detecting Sybil attacks by measuring metrics like detection accuracy, false positive rate, and computational overhead.

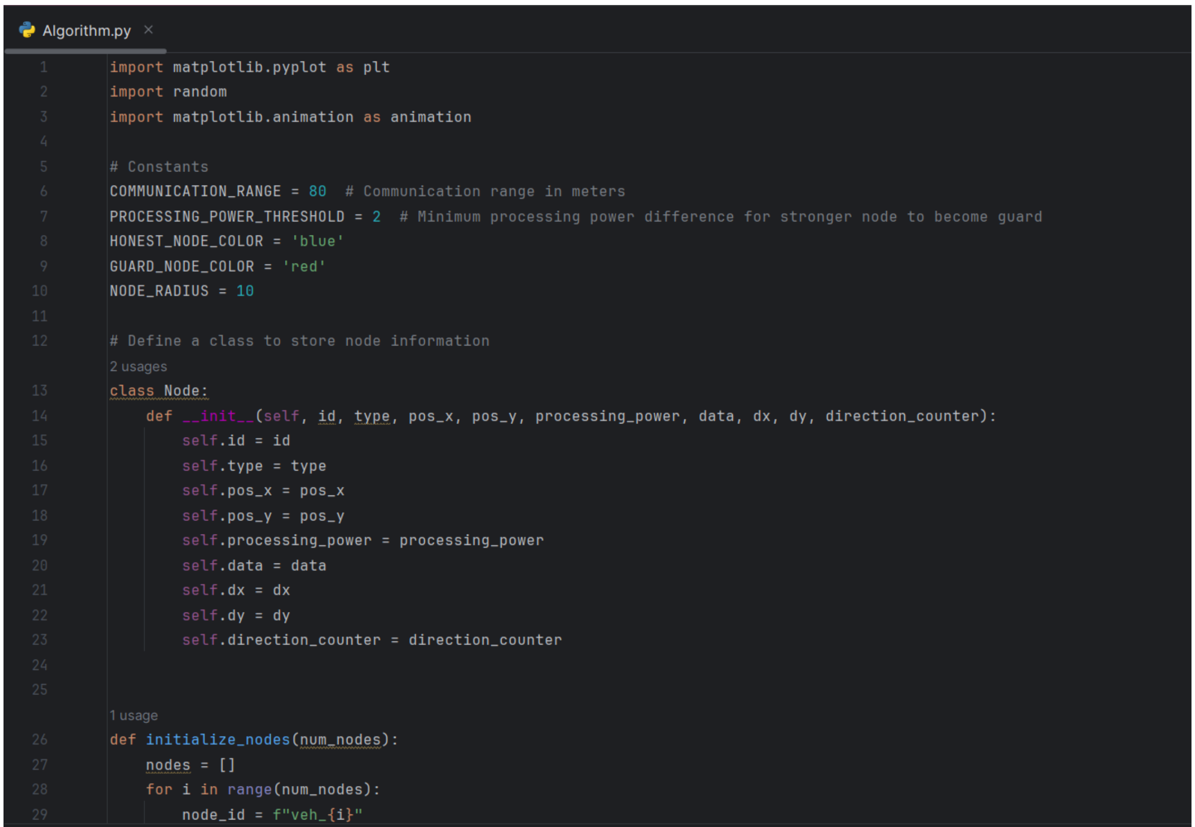


```python
import traci
import matplotlib.pyplot as plt

# Constants
COMMUNICATION_RANGE = 100  # Example: Communication range in meters
SPEED_THRESHOLD = 5  # Example: Speed threshold difference for suspicious node detection

# Global variable to store the count of suspicious nodes
suspicious_nodes_count = 0


# Step 1: Guard Node Selection
def initialize_guard_nodes():
    # Get list of all vehicle IDs in the simulation
    vehicle_ids = traci.vehicle.getIDList()

    # Iterate over each vehicle
    for vehicle_id in vehicle_ids:
        # Set vehicle as a guard node
        traci.vehicle.setVehicleVariable(vehicle_id, "guard_node", 1)


def handle_encounters():
    # Get list of all vehicle IDs in the simulation
    vehicle_ids = traci.vehicle.getIDList()

    # Iterate over each vehicle
    for vehicle_id in vehicle_ids:
        # Check if the vehicle is a guard node
        if traci.vehicle._getUniversal(vehicle_id, traci.constants.VAR_TYPE) == 1:
            # Get position of the current guard node
            pos_x, pos_y = traci.vehicle.getPosition(vehicle_id)

            # Iterate over other vehicles to check for encounters
            for other_id in vehicle_ids:
                # Skip self and non-guard nodes
                if other_id == vehicle_id or traci.vehicle.getUniversal(other_id, traci.constants.VAR_TYPE) != 1:
                    continue

                # Get position of the other vehicle
                other_pos_x, other_pos_y = traci.vehicle.getPosition(other_id)

                # Calculate distance between vehicles
                distance = ((pos_x - other_pos_x) ** 2 + (pos_y - other_pos_y) ** 2) ** 0.5

                # Check if vehicles are within encounter range
                if distance < COMMUNICATION_RANGE:
                    # Exchange information and compare capabilities
                    compare_capabilities(vehicle_id, other_id)

```

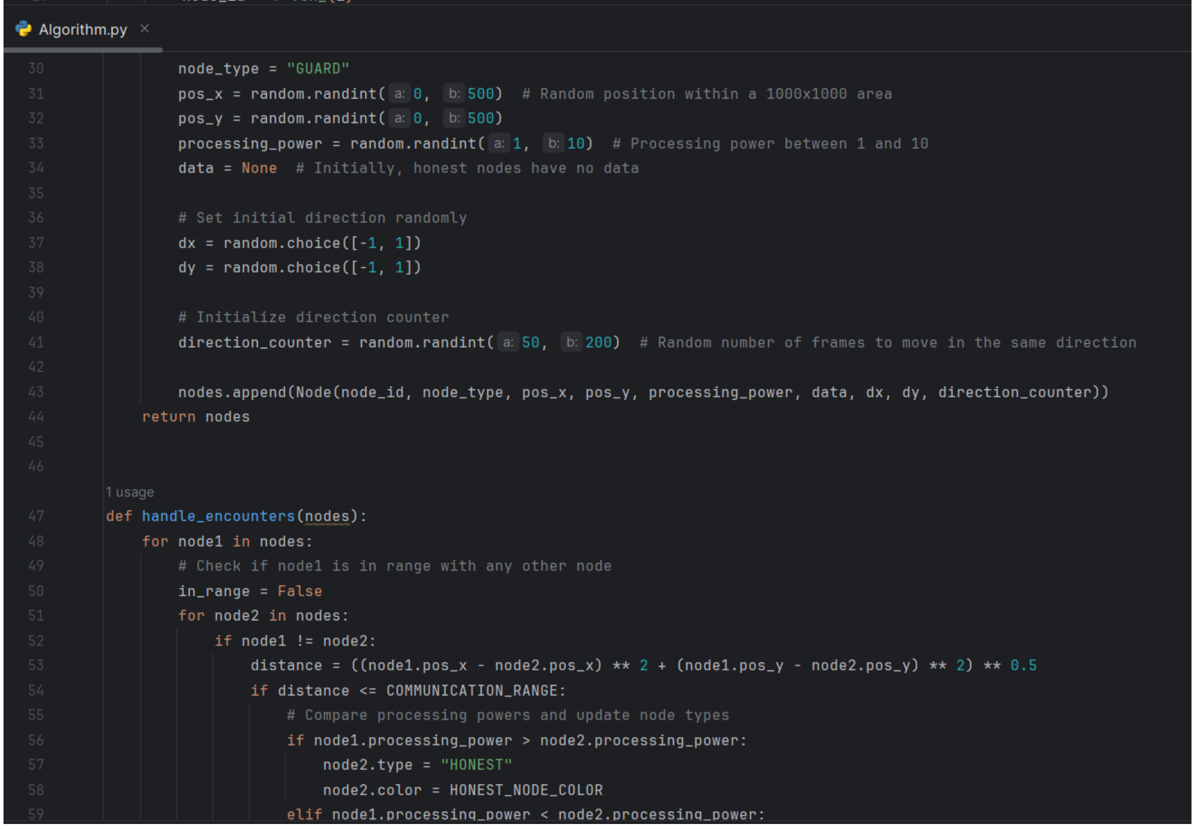
Fig 6.10: Our protocol implementation in Python using TrCI

Despite our best efforts, we faced significant challenges during the implementation process. The lack of accessible tutorials made it difficult to navigate the intricacies of integrating our protocol into SUMO. Additionally, certain protocol components, such as beacon message transmission and identifying malicious nodes, proved complex and were not directly supported by SUMO's TraCI interface. As a result, we had to develop these functionalities from scratch, leading to obstacles and errors along the way. Unfortunately, the absence of comprehensive guidance compounded our difficulties, hindering our ability to successfully integrate our protocol into SUMO.

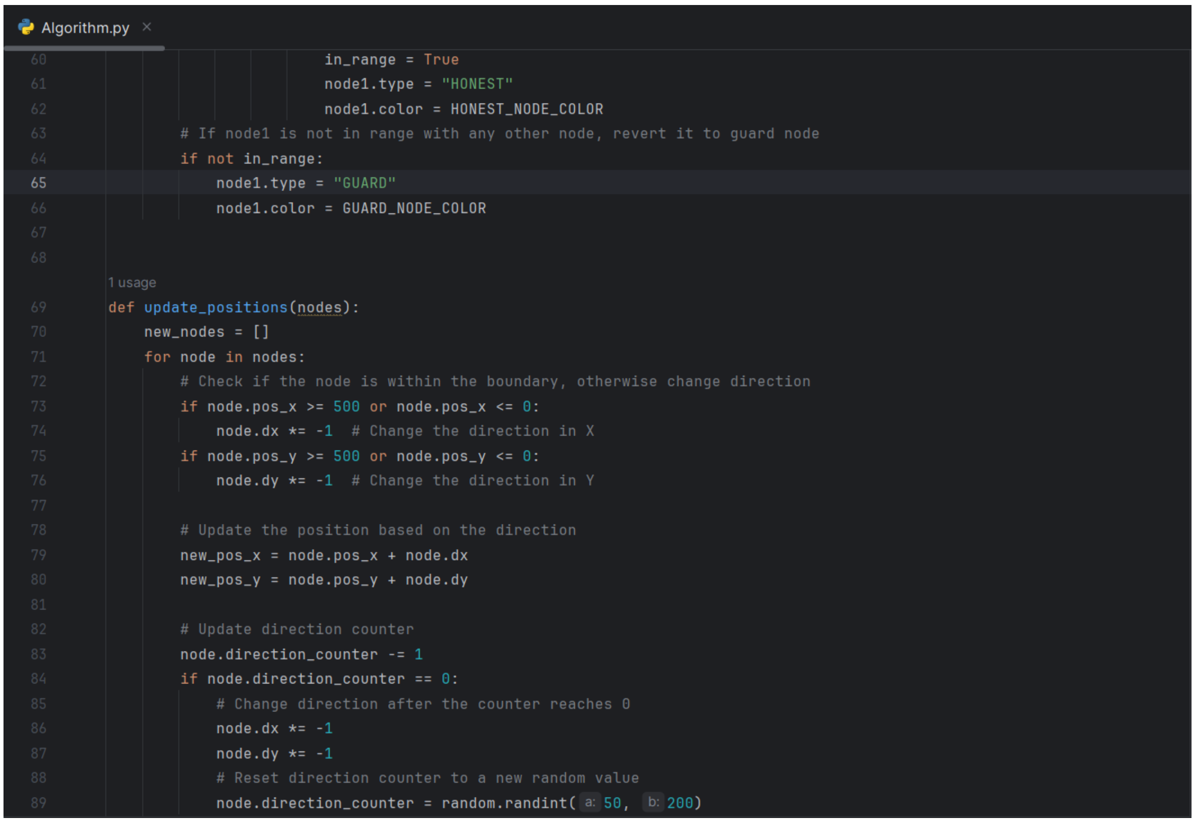
Therefore, faced with the challenges encountered during the integration of our protocol into SUMO, we opted to pursue an alternative approach. We endeavored to simulate a crucial aspect of our algorithm, namely the first step, Guard Node Selection and Fog formation through animation using Python. Despite the complexities inherent in the original implementation, this decision allowed us to focus on a segment of the algorithm deemed both significant and relatively straightforward to execute within the Python environment.



```python
import matplotlib.pyplot as plt
import random
import matplotlib.animation as animation

# Constants
COMMUNICATION_RANGE = 80  # Communication range in meters
PROCESSING_POWER_THRESHOLD = 2  # Minimum processing power difference for stronger node to become guard
HONEST_NODE_COLOR = 'blue'
GUARD_NODE_COLOR = 'red'
NODE_RADIUS = 10

# Define a class to store node information
2 usages
class Node:
    def __init__(self, id, type, pos_x, pos_y, processing_power, data, dx, dy, direction_counter):
        self.id = id
        self.type = type
        self.pos_x = pos_x
        self.pos_y = pos_y
        self.processing_power = processing_power
        self.data = data
        self.dx = dx
        self.dy = dy
        self.direction_counter = direction_counter


1 usage
def initialize_nodes(num_nodes):
    nodes = []
    for i in range(num_nodes):
        node_id = f"veh_{i}"
```



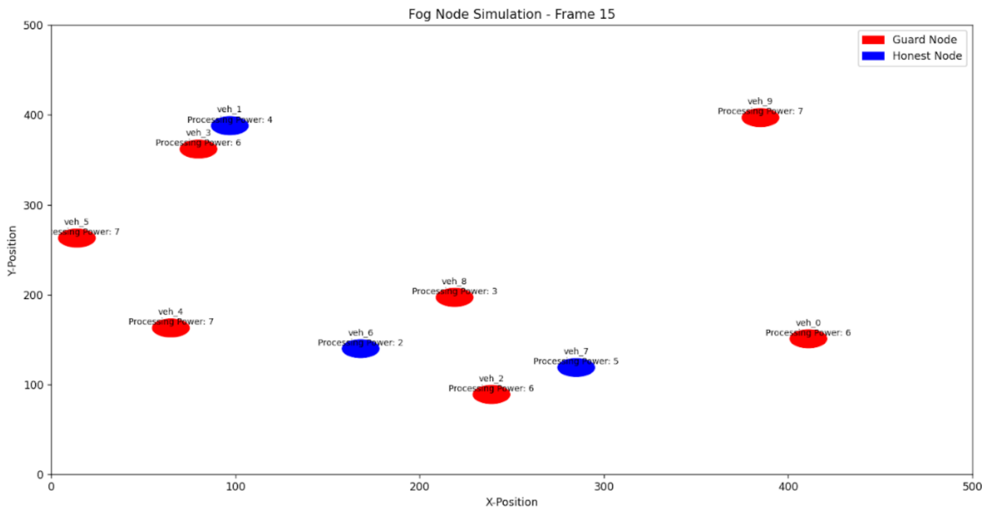
```python
        node_type = "GUARD"
        pos_x = random.randint( a: 0,  b: 500)  # Random position within a 1000x1000 area
        pos_y = random.randint( a: 0,  b: 500)
        processing_power = random.randint( a: 1,  b: 10)  # Processing power between 1 and 10
        data = None  # Initially, honest nodes have no data

        # Set initial direction randomly
        dx = random.choice([-1, 1])
        dy = random.choice([-1, 1])

        # Initialize direction counter
        direction_counter = random.randint( a: 50,  b: 200)  # Random number of frames to move in the same direction

        nodes.append(Node(node_id, node_type, pos_x, pos_y, processing_power, data, dx, dy, direction_counter))
    return nodes


1 usage
def handle_encounters(nodes):
    for node1 in nodes:
        # Check if node1 is in range with any other node
        in_range = False
        for node2 in nodes:
            if node1 != node2:
                distance = ((node1.pos_x - node2.pos_x) ** 2 + (node1.pos_y - node2.pos_y) ** 2) ** 0.5
                if distance <= COMMUNICATION_RANGE:
                    # Compare processing powers and update node types
                    if node1.processing_power > node2.processing_power:
                        node2.type = "HONEST"
                        node2.color = HONEST_NODE_COLOR
                    elif node1.processing_power < node2.processing_power:
```



```
                        in_range = True
                        node1.type = "HONEST"
                        node1.color = HONEST_NODE_COLOR
        # If node1 is not in range with any other node, revert it to guard node
        if not in_range:
            node1.type = "GUARD"
            node1.color = GUARD_NODE_COLOR


1 usage
def update_positions(nodes):
    new_nodes = []
    for node in nodes:
        # Check if the node is within the boundary, otherwise change direction
        if node.pos_x >= 500 or node.pos_x <= 0:
            node.dx *= -1  # Change the direction in X
        if node.pos_y >= 500 or node.pos_y <= 0:
            node.dy *= -1  # Change the direction in Y

        # Update the position based on the direction
        new_pos_x = node.pos_x + node.dx
        new_pos_y = node.pos_y + node.dy

        # Update direction counter
        node.direction_counter -= 1
        if node.direction_counter == 0:
            # Change direction after the counter reaches 0
            node.dx *= -1
            node.dy *= -1
            # Reset direction counter to a new random value
            node.direction_counter = random.randint( a: 50,  b: 200)
```

Fig 6.11: Our animation program

Through diligent effort and rigorous experimentation, we successfully developed a simulation model that accurately captures the essence of the initial step of our algorithm. This achievement represents a critical milestone in our ongoing efforts to evaluate the efficacy of our protocol in detecting Sybil attacks within VANETs.



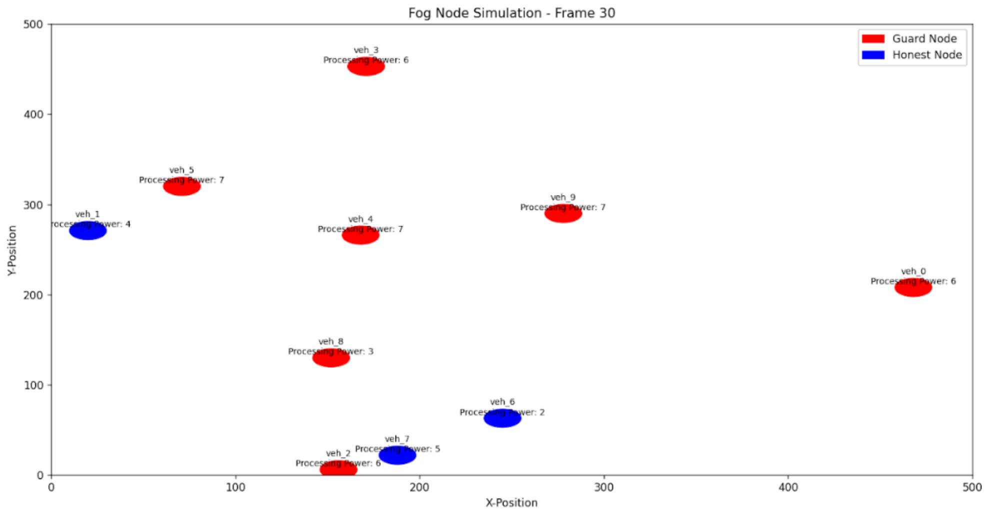


Fig 6.12: Our animation results

7. RESULT & DISCUSSION

Our project aimed to develop and apply a novel protocol to detect and prevent Sybil attacks in VANETs. The protocol consisted of three stages: Guard node selection and fog formation, Detection of Suspicious nodes and Validation of Sybil attacks. We carefully established each stage of the protocol by adhering to the key principles of algorithmic design, which include finiteness, definiteness, input, output, and effectiveness.

However, our execution of the protocol was hindered by our limited understanding of the simulation program and the absence of some features. As a response, we resorted to animation as an alternative method to replicate our methodology.

The diagrams provided clearly demonstrate that our simulation accurately represents the fundamental and crucial stage of our approach. Node selection for guarding and the generation of fog.

The formation of a fog network begins by assigning a guard node when a vehicle enters the road alone. This node is responsible for ensuring network security within its communication range. When guard nodes come across each other, they assess their skills and those that have similar attributes maintain their guard status to enhance network security. If one node significantly surpasses another in capabilities, the more powerful node takes over the function of the principal guard, while the less capable node switches to an honest node. The primary guard and the group of honest nodes combine to form a fog, which improves network security and enables smooth data exchange.

Within the domain of VANETs, which are known for their dynamic and constantly changing nature, the process of choosing and establishing guard nodes is a crucial stage in our protocol. The conspicuous lack of RSUs (Road Side Units) in our protocol is a noteworthy accomplishment stemming from our hard efforts. Through the intentional omission of RSUs in the protocol architecture, we have showcased a targeted strategy to simplify the implementation of the protocol and reduce unneeded intricacy. This strategic decision highlights our dedication to creating a protocol that is efficient and successful in dealing with the unique problems presented by VANETs. Moreover, the exclusion of RSUs is in line with our main goal of creating a protocol that can function independently in the ever-changing and decentralized environment that is typical of VANETs.

Undoubtedly, the capacity to adjust to the constantly changing structure inherent to VANETs is crucial for guaranteeing the effectiveness and durability of any protocol in real-life situations. Although our animated simulation accurately depicts the important process of guard node selection and fog creation, we are eager to subject our protocol to additional examination and validation. By exploring alternate simulation software platforms like OMNeT++ and SUMO, we may gain useful insights and verify the effectiveness of our protocol in many scenarios. Although we have faced difficulties, we are determined to improve and verify our protocol to handle the changing complexities of VANET systems.

8. SUMMARY

Our project aims to develop and implement a novel protocol for detecting and preventing Sybil attacks in VANETs. Beginning with a comprehensive literature survey of existing systems, we identified key research gaps and formulated our project objectives. Addressing the lack of effective solutions for Sybil attack mitigation in VANETs, our protocol seeks to enhance network security and reliability.

In our technical specification, we meticulously outlined the requirements for our protocol, encompassing both functional and non-functional aspects to guarantee its efficacy and robustness in addressing Sybil attacks within VANETs. Furthermore, we conducted comprehensive feasibility studies, evaluating the technical, economic, and social aspects of our project to ascertain its viability and potential impact. This assessment guided our decisions and strategies throughout the development process. Additionally, we defined precise hardware and software specifications, along with relevant standards and policies, to provide clear guidance for the development and implementation phases, ensuring adherence to industry best practices and regulatory requirements.

In crafting our design approach, we prioritized the development of a system architecture that could seamlessly adapt to the dynamic environment of VANETs while ensuring scalability and flexibility. We complemented this architectural framework with detailed design diagrams, including data flow, use case, class, and sequence diagrams, to provide a comprehensive visualization of system functionality and interactions. Additionally, we meticulously assessed constraints, alternatives, and tradeoffs to guide decision-making processes and optimize system performance, ensuring alignment with project objectives and stakeholder requirements.

In order to effectively manage the project and ensure its timely completion, we developed a comprehensive Gantt chart that delineates tasks, milestones, and associated timelines. This schedule serves as a roadmap for project execution, enabling effective coordination of activities and resources to meet project objectives within the specified timeframe.

For project demonstration, we endeavored to implement our protocol within simulation environments such as OMNeT++ and SUMO, with the ultimate goal of conducting real-world testing to validate its effectiveness. However, due to limited resources and unfamiliarity with these platforms, our attempts encountered challenges. As a workaround, we opted to animate the algorithm steps, yielding promising results. While our initial efforts were fruitful, there is room for further refinement and improvement in future iterations of the project.

9. References

[1] A. Borah and A. Paranjothi. (2023, October). "Sybil Attack Detection in VANETs using Fog Computing and Beamforming." IEEE 14th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON), New York, NY, USA, pp. 0626-0631. Available: https://ieeexplore-ieee-org.egateway.vit.ac.in/document/10316102

[2] Paranjothi and M. S. Khan. (2023, September). "Enhancing Security in VANETs with Sybil Attack Detection using Fog Computing." IEEE 98th Vehicular Technology Conference (VTC2023-Fall), Hong Kong, Hong Kong, pp. 1-6. Available: https://ieeexplore-ieee-org.egateway.vit.ac.in/document/10333491

[3] F. Concone, F. De Vita, A. Pratap, D. Bruneo, G. Lo Re, S. K. Das. (2022, January). "A fog-assisted system to defend against Sybils in vehicular crowdsourcing." Pervasive and Mobile Computing, vol. 83, pp. 101612. Available: https://www-sciencedirect-com.egateway.vit.ac.in/science/article/pii/S1574119222000505?via%3Dihub

[4] M. Baza et al. (2022, Jan.-Feb.). "Detecting Sybil Attacks Using Proofs of Work and Location in VANETs." IEEE Transactions on Dependable and Secure Computing, vol. 19, no. 1, pp. 39-53. Available: https://ieeexplore-ieee-org.egateway.vit.ac.in/document/9091099

[5] Y. Zhong, B. Yang, Y. Li, H. Yang, X. Li, Y. Zhang. (2023, August). "Tackling Sybil Attacks in Intelligent connected vehicles: A Review of Machine Learning and Deep Learning Techniques." 8th International Conference on Computational Intelligence and Applications (ICCIA), Haikou, China, pp. 8-12. Available: https://ieeexplore-ieee-org.egateway.vit.ac.in/document/10387880

[6] S. A. Mousavi, M. S. Sirjani, S. J. B. Z. Razavi and M. Nikooghadam. (2023, December). "SecVanet: provably secure authentication protocol for sending emergency events in VANET." 14th International Conference on Information and Knowledge Technology (IKT), Isfahan, Iran, Islamic Republic of, pp. 86-91. Available: https://ieeexplore-ieee-org.egateway.vit.ac.in/document/10433027

[7] T. N. Canh and X. HoangVan. (2023, December). "Machine Learning-Based Malicious Vehicle Detection for Security Threats and Attacks in Vehicle Ad-Hoc Network (VANET) Communications." RIVF International Conference on Computing and Communication Technologies (RIVF), Hanoi, Vietnam, pp. 206-211. Available: https://ieeexplore-ieee-org.egateway.vit.ac.in/document/10471804

[8] Z. Dong, H. Wu, Z. Li, D. Mi, O. Popoola and L. Zhang. (2023, December). "Trustworthy VANET: Hierarchical DAG-Based Blockchain Solution with Proof of Reputation Consensus Algorithm." IEEE International Conference on Blockchain (Blockchain), Danzhou, China, pp. 127-132. Available: https://ieeexplore-ieee-org.egateway.vit.ac.in/document/10411480

[9] L. Bouchrit, S. Zairi, I. C. Msadaa, A. Dhraief and K. Drira. (2023, December). "Flying to the Rescue: UAV-Assisted Urgent Alert Transmission in VANET." IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), Paris, France, pp. 1-6. Available: https://ieeexplore-ieee-org.egateway.vit.ac.in/document/10477830

[10] M. Saleh. (2023, December). "Security-based Multipath Route Switching Protocol for Quality-of-Service Enhancement in VANETs using Wiedemann Car-Following Model." IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), Paris, France, pp. 1-6. Available: https://ieeexplore-ieee-org.egateway.vit.ac.in/document/10477782

APPENDIX A – SAMPLE CODE

```
import matplotlib.pyplot as plt
import random
import matplotlib.animation as animation

# Constants
COMMUNICATION_RANGE = 80  # Communication range in meters
PROCESSING_POWER_THRESHOLD = 2  # Minimum processing power difference for
stronger node to become guard
HONEST_NODE_COLOR = 'blue'
GUARD_NODE_COLOR = 'red'
NODE_RADIUS = 10

# Define a class to store node information
class Node:
   def __init__(self, id, type, pos_x, pos_y, processing_power, data, dx,
dy, direction_counter):
       self.id = id
       self.type = type
       self.pos_x = pos_x
       self.pos_y = pos_y
       self.processing_power = processing_power
       self.data = data
       self.dx = dx
       self.dy = dy
       self.direction_counter = direction_counter


def initialize_nodes(num_nodes):
   nodes = []
   for i in range(num_nodes):
       node_id = f"veh_{i}"
       node_type = "GUARD"
       pos_x = random.randint(0, 500)  # Random position within a
1000x1000 area
       pos_y = random.randint(0, 500)
       processing_power = random.randint(1, 10)  # Processing power
between 1 and 10
       data = None  # Initially, honest nodes have no data

       # Set initial direction randomly
```

```
        dx = random.choice([-1, 1])
        dy = random.choice([-1, 1])

        # Initialize direction counter
        direction_counter = random.randint(50, 200)  # Random number of
frames to move in the same direction

        nodes.append(Node(node_id, node_type, pos_x, pos_y,
processing_power, data, dx, dy, direction_counter))
    return nodes


def handle_encounters(nodes):
    for node1 in nodes:
        # Check if node1 is in range with any other node
        in_range = False
        for node2 in nodes:
            if node1 != node2:
                distance = ((node1.pos_x - node2.pos_x) ** 2 + (node1.pos_y
- node2.pos_y) ** 2) ** 0.5
                if distance <= COMMUNICATION_RANGE:
                    # Compare processing powers and update node types
                    if node1.processing_power > node2.processing_power:
                        node2.type = "HONEST"
                        node2.color = HONEST_NODE_COLOR
                    elif node1.processing_power < node2.processing_power:
                        in_range = True
                        node1.type = "HONEST"
                        node1.color = HONEST_NODE_COLOR
        # If node1 is not in range with any other node, revert it to guard
node
        if not in_range:
            node1.type = "GUARD"
            node1.color = GUARD_NODE_COLOR


def update_positions(nodes):
    new_nodes = []
    for node in nodes:
        # Check if the node is within the boundary, otherwise change
direction
        if node.pos_x >= 500 or node.pos_x <= 0:
            node.dx *= -1  # Change the direction in X
        if node.pos_y >= 500 or node.pos_y <= 0:
            node.dy *= -1  # Change the direction in Y

        # Update the position based on the direction
        new_pos_x = node.pos_x + node.dx
        new_pos_y = node.pos_y + node.dy

        # Update direction counter
        node.direction_counter -= 1
        if node.direction_counter == 0:
            # Change direction after the counter reaches 0
            node.dx *= -1
            node.dy *= -1
            # Reset direction counter to a new random value
            node.direction_counter = random.randint(50, 200)

        new_node = Node(node.id, node.type, new_pos_x, new_pos_y,
node.processing_power, node.data, node.dx, node.dy,
                       node.direction_counter)
        new_nodes.append(new_node)
    return new_nodes
```

```
def draw_node(node, ax):
   circle = plt.Circle((node.pos_x, node.pos_y), NODE_RADIUS,
                       color=HONEST_NODE_COLOR if node.type == "HONEST"
else GUARD_NODE_COLOR)
   text = plt.Text(node.pos_x, node.pos_y + NODE_RADIUS + 2,
                   f"{node.id}\n{node.data}" if node.data else
f"{node.id}\nProcessing Power: {node.processing_power}",
                   ha='center', va='center', fontsize=8)
   ax.add_patch(circle)
   ax.add_artist(text)


def visualize_nodes(nodes, frame_num):
   plt.cla()  # Clear previous plot
   for node in nodes:
       draw_node(node, plt.gca())
   plt.xlabel("X-Position")
   plt.ylabel("Y-Position")
   plt.title(f"Fog Node Simulation - Frame {frame_num}")
   plt.xlim(0, 500)  # Set limits for X-axis
   plt.ylim(0, 500)  # Set limits for Y-axis

   guard_patch = plt.Rectangle((0, 0), NODE_RADIUS, NODE_RADIUS,
color=GUARD_NODE_COLOR)
   honest_patch = plt.Rectangle((0, 0), NODE_RADIUS, NODE_RADIUS,
color=HONEST_NODE_COLOR)
   legend_labels = ["Guard Node", "Honest Node"]
   plt.legend([guard_patch, honest_patch], legend_labels, loc='upper
right')

def animate(frame_num):
   global nodes  # Update global nodes list
   nodes = update_positions(nodes)  # Update positions (returns new list)
   handle_encounters(nodes)  # Handle encounters between nodes
   visualize_nodes(nodes, frame_num)  # Update visualization


if __name__ == "__main__":
   # Number of nodes to simulate
   num_nodes = 10

   # Initialize nodes
   nodes = initialize_nodes(num_nodes)

   # Animation setup
   fig, ax = plt.subplots()
   ani = animation.FuncAnimation(fig, animate, frames=100, interval=20)  #
Animate for 100 frames with 50ms interval
   plt.show()
```