Hybrid Machine Learning-based Intrusion Detection System Framework for Vehicular Ad hoc Networks

Erfan Amirzadeh Shams

Submitted to the
Institute of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Applied Mathematics and Computer Science

Eastern Mediterranean University
August 2022
Gazimağusa, North Cyprus

Approval of the Institute of Graduate Studies and Research

Prof. Dr. Ali Hakan Ulusoy
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Doctor of Philosophy in Applied Mathematics and Computer Science.

Prof. Dr. Nazim Mahmudov
Chair, Department of Mathematics

We certify that we have read this thesis and that in our opinion it is fully adequate in scope and quality as a thesis for the degree of Doctor of Philosophy in Applied Mathematics and Computer Science.

Prof. Dr. Ali Hakan Ulusoy
Co-Supervisor

Prof. Dr. Ahmet Rizaner
Supervisor

Examining Committee

1. Prof. Dr. Rashad Aliyev
2. Prof. Dr. Aydın Çetin
3. Prof. Dr. Ahmet Rizaner
4. Assoc. Prof. Dr. Nazife Dimililer
5. Asst. Prof. Dr. Müge Saadetoğlu

ABSTRACT

Autonomous transportation has an intense impact on the way of our day-to-day commute or traveling. The benefit of a sustainable and flexible system combined with safety and comfort is the next step of the transportation evolution. Connectedness is an inseparable part of this evolution, and it is possible through Intelligent Transportation Systems applications. These applications use a network system known as Vehicular Ad hoc Network (VANET). This system comes with its own benefits and drawbacks. In this research, we study one of the most acute issues of VANET which is security. The safety that is provided by VANET can be easily compromised by malicious users, hence the need for an Intrusion Detection System (IDS). An IDS is a computer program that detects the presence of malicious users and acts accordingly.

We designed two IDS models that can cooperate in a hybrid manner for detecting and eliminating attackers in VANET. Therefore, we generated relevant network data for training the core of the IDS using the widely utilized Network Simulator 2, 3 (ns-2, ns-3), and Simulation of Urban Mobility (SUMO) tools. We employed the Support Vector Machine classifier to design a lightweight Trust-aware binary IDS (TSIDS) with only three inputs to detect packet dropping and delaying attacks, as well as a multi-class IDS using Convolutional Neural Networks (CNN) with a novel feature extraction method (CAFECNN). The TSIDS uses a modified version of the promiscuous mode that only collects relevant packets for monitoring the neighboring nodes or vehicles. The CAFECNN model takes advantage of the interconnected Roadside Units to collect network flow information and use the said data for detecting both passive and active types of attacks. We evaluated both models using real-time

simulations and, in both cases, we observed an immediate improvement in the performance of the network in presence of intruders.

Keywords: Intrusion Detection System, Vehicular Ad hoc Network, Support Vector Machine, Convolutional Neural Network.

ÖZ

Otonom ulaşım, günlük işe gidip gelme veya seyahat etme şeklimiz üzerinde yoğun bir etkiye sahiptir. Güvenlik ve konforla birleştirilmiş sürdürülebilir ve esnek bir sistemin faydası, ulaşım evriminin bir sonraki adımıdır. Bağlılık bu evrimin ayrılmaz bir parçasıdır ve Akıllı Ulaşım Sistemleri uygulamaları ile mümkündür. Bu uygulamalar, Taşıtlar Arası Ağ (VANET) olarak bilinen bir ağ sistemi kullanır. Bu sistem kendi avantajları ve dezavantajları ile birlikte gelir. Bu araştırmada, VANET'in en önemli sorunlarından biri olan güvenlik konusunu inceledik. VANET'ler tarafından sağlanan güvenlik, kötü niyetli kullanıcılar tarafından kolayca tehlikeye atılabilir, bu nedenle bir Saldırı Tespit Sistemine (IDS) ihtiyaç duyulur. IDS, kötü niyetli kullanıcıların varlığını tespit eden ve buna göre hareket eden bir bilgisayar programıdır.

Bu çalışmada VANET sistemlerinde gerçekleşen saldırıları tespit etmek ve ortadan kaldırmak için hibrit olarak çalışan iki IDS modeli tasarladık. Bu amaçla, önerilen IDS için gerekli olan verileri bilgisayar simülasyonu ile oluşturmak için bu amaçla yaygın olarak kullanılan ns-2, ns-3 ve SUMO kullanılmıştır. Paket bırakma ve geciktirme tipi saldırıları algılayabilmek için Destek Vektör Makinesi sınıflandırıcısı kullanılarak yalnızca üç girişe sahip Güvene duyarlı ikili IDS (TSIDS) önerdik. Ayrıca yeni bir özellik çıkartma metoduna sahip Evrişimli Sinir Ağları (CNN) kullanan çok sınıflı bir IDS (CAFECNN) tasarlanmıştır. TSIDS, komşu düğümleri veya araçları izlemek için yalnızca ilgili paketleri toplayan karışık modun değiştirilmiş bir sürümünü kullanır. CAFECNN modeli, ağ akış bilgilerini toplamak ve söz konusu bilgileri hem pasif hem de aktif saldırı türlerini tespit etmek için kullanmak için birbirine bağlı Yol Kenarı

Birimlerinden yararlanır. Her iki modeli de gerçek zamanlı simülasyonlar kullanarak değerlendirdik ve her iki durumda da izinsiz giriş yapan araçların varlığında ağın performansında önemli iyileşmeler gözlemledik.

Anahtar Kelimeler: Saldırı Tespit Sistemi, Araçlar Arası Ağ, Destek Vektör Makinesi, Evrişimli Sinir Ağı.

Dedicated to all seekers of knowledge who came before us
and the next generation who keep pursuing the truth.

ACKNOWLEDGMENTS

This study is the result of not only hours, days, or years of research and experimentation, but the result of the tireless support, encouragement, and supervision of Prof. Dr. Ahmet Rizaner and Prof. Dr. Ali Hakan Ulusoy. I am forever indebted to them for believing in me when I did not.

I feel privileged to learn from the best of the professors and instructors in the department of mathematics, to whom I owe an invaluable debt of knowledge that enables me to see the world with a broader scope and a finer margin.

I would like to also express my gratitude for the constant support and uplift from Assoc. Prof. Dr. Nazife Dimililer, the director of the School of Computing and Technology.

Finally, I will never forget the kindness and support of my colleagues and managers at the EMU PDRAM and the Institute of Graduate Studies and Research for providing a friendly atmosphere for me to focus on my studies.

TABLE OF CONTENTS

ABSTRACT iii

ÖZ v

DEDICATION vii

ACKNOWLEDGMENTS viii

LIST OF TABLES xii

LIST OF FIGURES xiii

LIST OF SYMBOLS AND ABBREVIATIONS xvi

1 INTRODUCTION 1

1.1 Preliminaries 1

1.1.1 Vehicular Ad Hoc Network 1

1.1.2 Intrusion Detection System 2

1.1.2.1 Data Gathering Module 3

1.1.2.2 Detection Module 3

1.1.2.3 Response Module 7

1.1.3 Neural Network-based Classification 8

1.2 Research Queries 10

1.3 Contributions 10

1.4 Outline 11

2 RELATED WORK 13

2.1 General-Purpose IDS Models 13

2.2 VANET Specific IDS Models 18

3 PROBLEM DEFINITION 29

3.1 Information and Communication Security Threats 29

3.1.1 Network-based Intrusion 29
3.1.2 Host-based Intrusion 31
3.2 VANET Mobility Traces 32
3.3 VANET IDS Dataset Generation 33
3.4 Computational Complexity 34
4 SUPPORT VECTOR MACHINE-BASED BINARY IDS 36
4.1 Packet Header Analysis 37
4.2 SVM Intrusion Detection Module 39
4.3 Trust Value Table 41
4.4 Performance Analysis 44
4.4.1 SVM Classification Performance 45
4.4.2 Trust Value Analysis 47
4.4.3 Computational Complexity of SVM 48
4.5 TSIDS Simulation Result 49
5 CONTEXT-AWARE FEATURE EXTRACTION-BASED CNN MULTI-CLASS IDS 55
5.1 CAFECNN System Model Design 55
5.1.1 Dataset Structure 56
5.1.1.1 NSL-KDD Dataset 56
5.1.1.2 CICIDS2017 Dataset 57
5.1.1.3 ADFA Datasets 58
5.1.2 Feature Extraction 59
5.1.2.1 Feature Extraction for NSL-KDD and CICIDS2017 60
5.1.2.2 Feature Extraction for ADFA-LD 62
5.1.2.3 Feature Extraction for ADFA-WD 63

5.1.3 Feature Selection 64
5.1.4 Data Augmentation 66
5.2 CAFECNN IDS Design and Training 70
5.2.1 Model Fine-Tuning 72
5.3 Performance Analysis of CAFECNN 74
5.4 New VANET IDS Dataset 85
5.4.1 Simulation Setup 86
5.4.2 Network Flow Data Collection 88
5.5 Proposed IDS for the CVI Dataset 91
5.5.1 Preprocessing 91
5.5.2 Training and Fine-Tuning 91
5.5.3 CAFECNN Intrusion Detection Module 92
5.5.4 Trust Value Analysis 96
5.5.5 Computational Complexity of CAFECNN 98
5.6 CAFECNN Simulation Result 99
6 CONCLUSION AND FUTURE STUDY 104
6.1 Conclusion 104
6.2 Future Study 105
REFERENCES 106

LIST OF TABLES

Table 4.1: Simulation Parameters (ns-2, SUMO). 40

Table 4.2: Performance of the trained SVM in terms of Precision, Recall, and F-Score in urban scenarios. 46

Table 4.3: Performance of the trained SVM in terms of Precision, Recall, and F-Score in the highway scenario with 50 Vehicles and 5 Attackers. 47

Table 5.1: Original dataset size. 57

Table 5.2: Dataset feature information regarding data types. 57

Table 5.3: Eliminated features from the NSL-KDD and CICIDS2017 datasets after ERT features selection. 66

Table 5.4: Final dataset class distribution after the complete preprocessing steps. ... 69

Table 5.5: CNN structure for each dataset with and without fine-tuning. 74

Table 5.6: Accuracy and classification time of the proposed IDSs before and after fine-tuning. 80

Table 5.7: Simulation Parameters (ns-3, SUMO). 87

Table 5.8: CAFECNN features for VANET IDS 89

Table 5.9: New dataset class distribution 90

Table 5.10: CNN structure for CVI dataset with and without fine-tuning 92

Table 5.11: Accuracy and BAC of train and test data for the CVI dataset 93

LIST OF FIGURES

Figure 3.1: Vehicular mobility simulation area using SUMO 33

Figure 4.1: TSIDS block diagram. 42

Figure 4.2: Urban (top) and highway (bottom) mobility models in SUMO. 44

Figure 4.3: Initial trust value performance against PDR with 30 nodes and 4 attackers. 48

Figure 4.4: Initial trust value performance against EED with 30 nodes and 4 attackers. 48

Figure 4.5: TSIDS performance analysis for PDR in 10 vehicles scenario. 50

Figure 4.6: TSIDS performance analysis for PDR in 30 vehicles scenario. 50

Figure 4.7: TSIDS performance analysis for PDR in 50 vehicles scenario. 51

Figure 4.8: TSIDS performance analysis for EED in 10 vehicles scenario. 51

Figure 4.9: TSIDS performance analysis for EED in 30 vehicles scenario. 52

Figure 4.10: TSIDS performance analysis for EED in 50 vehicles scenario. 52

Figure 4.11: TSIDS performance analysis for PDR in highway vs. urban scenarios with 50 vehicles and 5 attackers. 53

Figure 4.12: TSIDS performance analysis for EED in highway vs. urban scenarios with 50 vehicles and 5 attackers. 53

Figure 4.13: PDR performance comparison with 76 vehicles and 8 attackers. 54

Figure 4.14: EED performance comparison with 76 vehicles and 8 attackers. 54

Figure 5.1: An example of data augmentation process on an arbitrary sample with 9 features. 68

Figure 5.2: Visual example of the final preprocessing output for NSL-KDD, CICIDS2017, ADFA-LD, and ADFA-WD datasets. 68

Figure 5.3: Preprocessing block diagram for HIDS and NIDS datasets. 69
Figure 5.4: Workflow of the proposed CAFECNN IDS. 73
Figure 5.5: Accuracy of the CAFECNN on KDDTest+ with different preprocessing steps. Including NP, FE, FS, and DA preprocessing steps separately, or together. ... 78
Figure 5.6: Classification time per sample for the CAFECNN on KDDTest+ with different preprocessing steps. Including NP, FE, FS, and DA preprocessing steps separately, or together. 78
Figure 5.7: Performance of the proposed CAFECNN IDS for NSL-KDD, ADFA-LD, and ADFA-WD datasets in terms of Accuracy, WPR, WRC, and WFS. 81
Figure 5.8: Performance comparison of the proposed CAFECNN IDS with the other models for the NSL-KDD Test+ dataset regarding the multiclass classification Accuracy. Including results from RNN [25], SCAD-RNN [26], CNN [28], multi-CNN [29], and other methods that are mentioned in [29]. 82
Figure 5.9: Performance comparison of the proposed CAFECNN IDS with the other models for the CICIDS2017 dataset regarding the multiclass classification Accuracy. Including results from IG models [30], Cu(LSTM-CNN)* [32], DL-IDS* [33], CNN+LSTM11* [34], and DeepCoin* [35]. 83
Figure 5.10: Performance comparison of the proposed CAFECNN IDS with the MLP-based model [43] regarding the ADFA-LD dataset in terms of Accuracy, WPR, WRC, and WFS. 84
Figure 5.11: Performance comparison of the proposed CAFECNN IDS with the MLP-based model [43] regarding the ADFA-WD dataset in terms of Accuracy, WPR, WRC, and WFS. 85
Figure 5.12: Simulation map and RSU placement. 88

Figure 5.13: Generated samples as the result of CAFECNN preprocessing for the CVI dataset. 91
Figure 5.14: Confusion matrix of CVI test data for CAFECNN, RFC, DTC, MLP, and SVM models. 94
Figure 5.15: Performance comparison of the proposed CAFECNN IDS with the RFC, DTC, SVM, and MLP-based models using the CVI dataset in terms of Accuracy, WPR, WRC, and WFS. 95
Figure 5.16: Performance comparison of the proposed CAFECNN IDS with the RFC, DTC, SVM, and MLP-based models using the CVI dataset in terms of BAC, MPR, MRC, and MFS. 96
Figure 5.17: Trust value limit performance against PDR with 50 nodes and 5 attackers. 97
Figure 5.18: Trust value limit performance against EED with 50 nodes and 5 attackers. 98
Figure 5.19: CAFECNN performance analysis for PDR in 50 vehicles and 10 connections scenario. 100
Figure 5.20: CAFECNN performance analysis for EED in 50 vehicles and 10 connections scenario. 101
Figure 5.21: CAFECNN performance analysis for PDR in 50 vehicles and 20 connections scenario. 102
Figure 5.22: CAFECNN performance analysis for EED in 50 vehicles and 20 connections scenario. 102
Figure 5.23: Performance impact of attacks on the VANET in terms of PDR. 103
Figure 5.24: Performance impact of attacks on the VANET in terms of EED. 103

LIST OF SYMBOLS AND ABBREVIATIONS

$\|dc_i\|$	The number of times dc_i is called in the trace
$\|S_i\|$	The number of times the i^{th} system call appeared in the trace
$\|T\|$	The length of the system call trace
$\lfloor N_p \rfloor$	Integer part of N_p
C_p	Number of pixels for categorical features
da	The number of times dc_i is followed by dc_{j-1}
dc_i	The i^{th} DLL call
df_i	The number of times dc_i appears in every 5 consecutive DLL calls
D_i	The 12 pair feature vector corresponding to the i^{th} DLL
$d_{i,j}$	The j^{th} element of D_i
dm_i	The number of unique dc_i+*memory_address* calls
d_t	Trust value decay
f	The number of features
$frac(N_p)$	Fraction part of N_p
g_t	Trust value gain
ma	Memory Address (*memory_address*)
m_c	Maximum value of a feature in the dataset
ms	Millisecond
n	The number of classes
N_p	Number of pixels for numerical features
p	The number of pixels
q	The number of samples
q_i	The number of samples in class i

S_i	The i^{th} system call
SP_i	The i^{th} system call ratio in the final feature vector
T	System call trace
v	The number of support vectors
x_c	The original value of a feature
$\hat{y}$	Classifier output
µs	Microsecond
ADFA-LD	Australian Defence Force Academy Linux Dataset
ADFA-WD	Australian Defence Force Academy Windows Dataset
A-IDS	Anomaly-based Intrusion Detection System
ANFIS	Neuro-Fuzzy Inference System
AODV	Ad hoc On-Demand Distance Vector
AOMDV	Ad hoc On-Demand Multiple Path Distance Vector
API	Application Programming Interface
AWID	Aegean WiFi Intrusion Dataset
BAC	Balanced Accuracy
BN	Bayes Network
BSM	Basic Safety Message
CAFECNN	Context-Aware Feature Extraction-based CNN
CBR	Constant Bit Rate
CH	Cluster Head
CIA	Confidentiality, Integrity, Availability
CICIDS2017	Canadian Institute for Cybersecurity IDS 2017
CNN	Convolutional Neural Network
Conv	Convolution

CVI	CAFECNN VANET IDS
DA	Data Augmentation
DARPA	Defense Advanced Research Projects Agency
DCDIV	Distributed Collaborative intrusion Detection system based on Invariant
DDoS	Distributed Denial of Service
DL	Deep Learning
DLL	Dynamic Link Library
DoS	Denial of Service
DTC	Decision Tree Classifier
EED	End-to-End Delay
EHA	Elephant Herd Algorithm
ERT	Extremely Randomized Trees
FE	Feature Extraction
FN	False Negative
FP	False Positive
FS	Feature Selection
FSVM	Fuzzy Support Vector Machine
GA	Genetic Algorithm
GPS	Global Positioning System
HIDS	Host-based Intrusion Detection System
IDS	Intrusion Detection System
IG	Information Gain
IoT	Internet of Things
IP	Internet Protocol

IPS	Intrusion Prevention System
ITS	Intelligent Transportation Systems
KDD	Knowledge Discovery and Data Mining
KH	Krill Herd
kNN	k Nearest Neighbor
LASSO	Least Absolute Shrinkage Selection Operator
LDA	Linear Discriminant Analysis
LSSVM-IDS	Least Square SVM-based IDS
LSTM	Long Short-Term Memory
MANET	Mobile Ad hoc Network
MFS	Macro F-Score
ML	Machine Learning
MLP	Multi-Layer Perceptron
MOVE	Mobility model generator for Vehicular networks
MPR	Macro Precision
MRC	Macro Recall
NB	Naïve Bayes
NIDS	Network-based Intrusion Detection System
NP	No Preprocessing
ns-2	Network Simulator 2
ns-3	Network Simulator 3
NSL-KDD	The modified version of KDD dataset
OBU	Onboard Unit
PCA	Principal Component Analysis
PDC	Packet Drop Count

PDR	Packet Delivery Ratio
PFI	Packet Forward Interval
PMBASR	Packet Marking Based on Adapted Stream Region
PML-CIDS	Privacy-preserving Machine Learning based Collaborative IDS
PTD	Packet Transfer Delay
ReLU	Rectified Linear Unit
RFC	Random Forest Classifier
RL	Reinforcement Learning
RNN	Recurrent Neural Network
RREP	Route Reply
RREQ	Route Request
RSU	Roadside Unit
RT	Random Tree
S-IDS	Signature-based Intrusion Detection System
SKB	Select K-Best
SUMO	Simulation of Urban Mobility
SVM	Support Vector Machine
TfT	Tit-for-Tat
TN	True Negative
TP	True Positive
TSIDS	Trust-aware SVM-based Intrusion Detection System
TVT	Trust Value Table
U2R	User-to-Root
UNSW-NB15	University of New South Wales-NB 2015
V2I	Vehicle to Infrastructure

V2V	Vehicle to Vehicle
V2X	Vehicle to Everything
VANET	Vehicular Ad hoc Network
WAVE	Wireless Access in Vehicular Environments
WFS	Weighted F-Score
WPR	Weighted Precision
WRC	Weighted Recall

Chapter 1

INTRODUCTION

1.1 Preliminaries

1.1.1 Vehicular Ad Hoc Network

In our technologically advanced world where every device is capable of network-based communication, it is time for our day-to-day transportation vessels to take advantage of this infrastructure as well. Autonomous vehicles are equipped with several specialized sensors for monitoring the traffic and their whereabouts for assisting the driver in experiencing safer and more comfortable transportation. Vehicular Ad hoc Network (VANET) is a cooperative communication system for information exchange between vehicles, known as Vehicle-to-Vehicle (V2V), between vehicles and existing infrastructures, referred to as Vehicle-to-Infrastructure (V2I), or between the vehicle and any other device that is known as Vehicle-to-Everything (V2X). The abovementioned communications use Intelligent Transportation Systems (ITS) applications. This is a bedrock for many safety and comfort applications that come with today's Internet of Things (IoT), as well as all the risks and securities that are either shared by other IoT devices or are specific to VANET. For instance, Cooperative Adaptive Cruise Control systems are one of the safety applications of VANETs that use wireless communication for collecting traffic and environment information for better decision-making in driving. However, the wireless nature of this communication creates potential security flaws in the system which substantially reduces the efficiency

of the communication. Issues such as wireless channel congestion or falsified information.

While network communication security is not a new topic, its progress is an ongoing challenge that is changing with new threats and vulnerabilities emerging together with new types of communication technologies. VANET is a subcategory of Mobile Ad hoc Network (MANET) with unique properties such as the specialized IEEE 802.11p standard that is known as Wireless Access in Vehicular Environments (WAVE), and much higher mobility and hence more dynamic topology.

1.1.2 Intrusion Detection System

The Confidentiality, Integrity, and Availability (CIA) triad is one of the most common frameworks for cybersecurity insurance. Malicious users often try to gain unauthorized access to data (confidentiality), tamper with the data in the communication channel (integrity), or render a service provider inaccessible (availability). An Intrusion Detection System (IDS) is a tool for protecting the CIA of a network or computing device [1]. An IDS detects the presence of an attacker after it affected the system by either recognizing the signature of the attack or by detecting anomalous behavior that is a deviation from normal behavior. It should not be confused with Intrusion Prevention System (IPS) where the purpose of the mechanism is to stop the malicious user before gaining access to the system or disrupting the normal process. An example of an IPS is user authentication that allows a trusted authority to confirm authorized access to the system.

Even though an IPS might sound like a better solution since it does not allow the intrusion in the first place, it may fail at certain points. There are times when a legitimate user's device is compromised and the malicious user takes advantage of that

to start a cyber-attack. This is where IDS complements the IPS by detecting the intrusion from the compromised device or user. As mentioned, an IDS either uses the signature of an attack which is known as Signature-based IDS (S-IDS), or it detects anomalies, which is known as Anomaly-based IDS (A-IDS). In either case, an IDS can be divided into at least two or three components namely the data gathering module, detection module, and response module. The detection and response module can be combined into one module as the detection engine, however, in this study, we use the three-component IDS model.

1.1.2.1 Data Gathering Module

The data gathering module is responsible for parsing relevant data and extracting required features for the detection module. For example, in a network attack scenario, an IDS can find signatures or anomalies by receiving network flow information such as the number of delivered or lost packets, network delay, *etc*. One of the most common ways to gather data for network intrusion detection is to use programs such as pcap to capture network traffic and parse them for further feature extraction.

A network packet is a segment of a whole message and consists of two major parts which are the packet header and the payload. The packet header contains information such as source, destination, and other required data regarding the packet. The payload is the actual message data that is meant to be delivered to the destination. Packet headers are often an important source for extracting features that can be used in the detection module.

1.1.2.2 Detection Module

The detection module which is the core of the IDS uses simple or complex classification techniques to detect intrusions. The classification algorithm analyzes the provided data from the gathering module through a certain set of rules or functions to

label the user interaction as malicious or normal. For instance, outlier detection is one of the primary applications of data mining [1] that can be adopted for this purpose. The definition of an outlier is very close to what we call an anomaly which is the basis of A-IDS. For example, a simple threshold-based method can be designed to detect anomalous behavior when the number of dropped packets and the packet delivery time is higher than the average of the normal observation. As mentioned before, other than detecting anomalies, a detection module can be programmed as S-IDS. There are more discussions regarding this topic in [2] and [3]. We also discussed the general approaches to creating detection modules in [4], where we mentioned that the detection engine in any IDS is either anomaly-based, designed to detect intrusions in the system by monitoring signs of deviation from normal behavior; or otherwise, is designed to be signature-based and rely on previous observation of attack data to detect known types of intrusions. To implement any type of IDS, it is necessary to prepare a dataset that includes features that best describe the normal behavior or an intrusion method, or both. The aforementioned datasets can be simulated or gathered through system monitoring techniques.

One of the most well-known network intrusion datasets is the Knowledge Discovery and Data Mining (KDD) CUP 99, also known as KDD-99 [5]. This dataset was used for the third international KDD tools competition. However, this dataset has flaws such as redundant data in both training and testing datasets. This issue can lead to bias towards those classes with redundant samples. A detailed analysis of the dataset can be found in [6]. The authors of the mentioned article proposed modifications to this dataset to fix the existing issues. The modified dataset is known as NSL-KDD and is available at [7]. Entries of this dataset are obtained from the tcpdump data of a UNIX host network traffic, conducted by MIT Lincoln Laboratory, and sponsored by Defense

Advanced Research Projects Agency (DARPA) as part of an off-line IDS evaluation program [8]. The gathered data includes 41 network-related features such as network protocol, source and destination data transfer size, number of connections to the host, etc. The value of these features may differ under different circumstances such as normal operation or Denial of Service (DoS) attack. This is especially helpful in Machine Learning (ML) techniques for data classification where a certain feature uniquely outweighs others for identifying a certain class. However, the boundaries between classes are not always clear, which makes it difficult for the classifier to simply rely on the raw features.

Even though NSL-KDD has been improved compared to its previous version, researchers have debunked its relevance for the current attack protocols [9]. To create contemporary datasets for intrusion detection, researchers have proposed new methods for generating relevant features for new types of intrusions. One of the more recent datasets is the Canadian Institute for Cybersecurity IDS 2017 (CICIDS2017) [10]. This dataset was released in 2017 and contains relatively newer types of attacks. It is generated by capturing the network traffic between an attack-network and a victim-network. The networks are on separate infrastructures with the most common operating systems installed. The former network launches network attacks on the latter which then captures the incoming traffic using pcap. The capture period is done in five days. The first day (Monday) has only normal traffic whereas the rest of the days include 6 types of network attacks. The dataset has more than 80 features, where 78 of them are selected for machine learning purpose and it is available in the MachineLearningCSV file from the original dataset repository.

Other than the mentioned datasets above, two of the most recent open datasets are presented by the Australian Defence Force Academy (ADFA) which includes Linux Dataset (ADFA-LD) [11], [12] and Windows Dataset (ADFA-WD) [12]. These datasets concentrate more on the realism of attack scenarios and system vulnerability; hence, the operating systems that are used for the data gathering honeypots are fully patched to conform to the latest security standards. ADFA datasets are favored in anomaly-based IDS designs, however, we took the multiclass approach in our research. Another major difference between the above datasets is that NSL-KDD and CICIDS2017 are suitable for Network-based Intrusion Detection System (NIDS) whereas ADFA datasets are made for Host-based Intrusion Detection System (HIDS). The reason for this is that NSL-KDD and CICIDS2017 dataset samples are generated by a combination of packet-based and flow-based data gathering methods whilst ADFA samples are generated by recording system calls from a single process on the host system. The major difference between HIDS and NIDS, aside from the data gathering methods, is the deployment place. NIDSs can be deployed on network gateways that are connected to many other hosts or clients regardless of their operating system or firmware. This allows them to monitor the security of the network flow coming from various sources. On the other hand, an HIDS is deployed on a single host at a time and only monitor the very host that it was installed on. The operating system of the host defines the data gathering method which in turn demands a more specialized design for the detection module. Hence, we can say that NIDS has greater scalability in terms of implementation, whilst HIDS is better at protecting against system-specific vulnerabilities.

A more detailed discussion on the dataset structure is presented in Section 5.1.1. Other noteworthy datasets that are commonly used as benchmarks are Kyoto 2006+ [13],

University of New South Wales-NB 2015 (UNSW-NB15) [14], and Aegean WiFi Intrusion Dataset (AWID) [15]. The mentioned datasets are all network-based and since we only wanted to use one benchmark for our NIDS, we decided to use the most commonly used dataset which is NSL-KDD. In the end, the main purpose of the detection module is to produce the most accurate output in response to the behavior of the users.

The output of the detection module is critical to the performance of an IDS. There are common metrics such as Precision and Recall for evaluating an IDS performance. The said metrics inform us about the confidence of the output from the detection module. At this point, the IDS can take immediate action based on the output of the detection module or it can use a specialized response module to make further decisions.

1.1.2.3 Response Module

Also known as the decision-making engine, is the final part of a three-component IDS. There are various methods for programming this module that can also depend on the type of the detection module. In its simplest form in an A-IDS this module just eliminates an intruder upon receiving alerts from the detection module. It can also be designed to take different actions based on the signature of the attack in an S-IDS.

We must take into consideration that detection modules are often imperfect and there are chances of misclassifications. A response module can help address this issue by creating tolerance for false positives. One of the effective solutions is to use a reputation or trust-based system. In this system, which we used in our study, the response module designates a trust value for each network member and updates this value based on the output of the detection module. This way, each time a user acts maliciously will lose its trustworthiness and for each cooperative behavior, it gains

trust. When the trust value or score of a user hits the minimum value the response module takes appropriate action to mitigate the effect of the malicious behavior. This method also requires a cap for the trust value to prevent high trust score exploitation. This will ensure that a small amount of false classification rate would not lead to the elimination of legitimate users.

Now that we know what we are expecting from an IDS we can decide which classification method we want to use for our detection module. As mentioned before, we can use a variety of tools and algorithms for this purpose, however, ML algorithms, especially neural networks evolution have proven to be an excellent tool for various classification and prediction tasks.

1.1.3 Neural Network-based Classification

Neural networks, also known as artificial neural networks, are special types of ML algorithms that mimic biological neural networks. This allows them to learn from examples using the weighted connection between the connected units or neurons. Hence, unlike traditional programming where we need to provide the functions and algorithm, the neural network can adjust the rules by itself by using learning rules. This reduces extensive programming or calculations unnecessary while also increasing the response rate.

The most popular neural network application is the classification task where the network learns to understand the difference between two or more classes in a dataset. After the learning process is done, the trained model can classify any input of the same structure within the boundary of the learned classes. The classification task can be either in binary form with only two classes, or it can be multi-class with n number of classes where n is a positive integer. Some of the neural network methods are

inherently binary and they require multiple models for classifying more than two classes. Support Vector Machine (SVM) is one of the most famous examples of binary classifiers. On the other hand, classifiers such as multi-layered Deep Learning (DL) models can perform multi-class classification without the need for multiple trained models. One of the most famous classifiers for image processing is the Convolutional Neural Network (CNN) which can be trained to classify hundreds of different classes in one model [16]. It should be noted that each one of the mentioned classifiers has its benefits for different applications.

In this study, we decided to use SVM for a lightweight binary IDS that can be easily executed by all vehicles and use the CNN classifier for a cloud-based multi-class IDS that is suitable for running on infrastructures with broader data access.

After establishing the structure of our IDS framework, we must decide where we want to implement the system. One of the issues in VANETs is partial dependence on infrastructures. Even though VANET does not need any infrastructure to function, it can still benefit from infrastructures such as Roadside Unit (RSU). For instance, a vehicle may require authenticating through an RSU to be able to be part of a VANET, or because of the limitation in communication range and signal degradation, vehicles depend on RSUs for signal amplification. Nevertheless, infrastructures such as RSUs can be utilized for other purposes such as an all-seeing IDS.

RSUs in a VANET are connected together and also have better access to high bandwidth connections. This property allows RSUs to share a high amount of traffic information from different flows that can be used for monitoring purposes such as IDS.

The IDS can also be installed on the Onboard Unit (OBU) of the vehicles to eliminate the requirement for the RSUs. Although vehicles are more constrained in terms of processing power and energy, lightweight solutions can easily be used without any major impact on the rest of the system.

1.2 Research Queries

By investigating the abovementioned issues and the available research in this field we would like to address the following questions in this study:

- How can we mitigate network attacks in VANET without significantly over sourcing the already limited capabilities of mobile devices such as the OBU of the vehicles?
- Is it possible to detect both active and passive cyber-attacks using a single IDS model?
- Is it possible to collect relevant data from network packets with zero overhead to detect the mentioned types of cyber-attacks?
- Are the existing benchmark datasets enough for creating or evaluating IDS models, and is it necessary to create more sophisticated test data as well as real-time simulation to evaluate an IDS?

We are going to have a more in-depth discussion regarding the existing problems in Chapter 3 with emphasis on types of cybersecurity attacks, dataset availability, and research gaps in this field.

1.3 Contributions

In this study, we aim to propose a hybrid IDS that is functional in both situations whether the network is formed only through vehicles or includes RSUs. In either case, vehicles can benefit from a lightweight IDS that monitors other neighbors for detecting

malicious behavior. Additionally, in presence of RSUs, the network can benefit from a stronger multi-class IDS that performs better in detecting passive attacks. It also allows the system to detect the type of attack based on its signature. The main contributions of this study are as follows:

- Create an efficient SVM-based IDS that can detect delaying and dropping types of attack in VANET using a small number of features. The proposed model is published in the journal of Computers & Security [17].
- Design a multi-class IDS using our proposed context-aware feature extraction and CNN that can identify passive types of attacks in addition to active types without additional network overhead. The original feature extraction method is published in the journal of Neural Computing and Applications [4].
- Define a simple yet effective response module that eliminates malicious users using reputation-based decision-making.
- Generate a new set of network flow-based datasets for VANET IDS using simulation tools. This dataset also includes more complex test data for IDS evaluation.

1.4 Outline

The rest of this research is organized and presented in six different chapters. The related work by other researchers in this field is presented in Chapter 2. In Chapter 3 we discuss the problem of maintaining security in VANETs and different types of intrusions. Chapter 4 is our proposed SVM-based IDS method that is created and evaluated using computer simulation, and in Chapter 5 we present our new feature extraction method that helps improve multi-class IDS performance using CNN. We also discuss its performance using existing IDS datasets as well as our simulated flow-based dataset. The simulation result for evaluating the real-time performance analysis

of the network is available at the end of their corresponding chapters, and finally, we have the conclusion and future study in Chapter 6.

Chapter 2

RELATED WORK

Intrusion detection algorithms have a long history of research and only in the past few years, VANETs gained popularity in this area. As Tidjon *et al.* [18] mentioned in their cross-domain IDS survey, the growth of the interconnected information exchange and transfer systems such as the IoT expanded the domain of security attacks as well. They investigated IDS models in general as a security application in several domains such as industrial transportation and healthcare areas. A part of the current study also investigates more general-purpose intrusion detections in network and host-based systems. The first section of this chapter is focused on the said types of IDS methods and is available in our previous study [4]. The second part of this chapter investigates recent studies in the domain of VANETs.

2.1 General-Purpose IDS Models

There are numerous publications regarding the intrusion detection topic over the past decade. Many of them are focused on creating solutions that are based on the KDD dataset while others may choose to evaluate their method with other IDS datasets. We are going to review the most recent and the most relevant works in this section.

Looking into the structure of an IDS, normally the data gathering module is the first part of the system. Hence, based on the gathered information, the IDS can be categorized as NIDS or HIDS. We previously discussed the said categories along with introductory information on NSL-KDD and ADFA datasets in Section 1.1.2.2; more

information on the structure of the mentioned datasets, as well as their features, is available in Section 5.1. Next, based on the detection module, we can categorize IDSs as binary or multiclass classifiers. In this section, we investigate previously studied methods of IDSs based on the above categories.

Supervised learning is the most common method used for training new IDSs; however, unsupervised ML as the one proposed by Choi *et al.* [19] is another effective way of generalizing intrusion data without the need for labeled data. In the said paper, the researchers employed autoencoders for training a binary classifier IDS using 85% of the NSL-KDD dataset for training and the rest of the data for testing. They also created three subsets from the training data with different label ratios for further analysis.

In a study by Kaur *et al.* [20], a binary classifier IDS is proposed by hybridizing the Firefly algorithm over K-Means. This study also considered NSL-KDD for evaluating the IDS. Another study in this category is done by Latah and Toker [21] which included several ML methods that were benchmarked against the NSL-KDD dataset.

SVM is inherently made for binary classification [22]; however, it is possible to use techniques such as the voting strategy with multiple SVMs for multiclass applications; Alabdallah and Awad [23] proposed a weighed SVM for intrusion detection. They combined the NSL-KDD train+ and test+ sets and then used cross-validation to evaluate their proposed IDS. Similarly, Ambusaidi and Nanda [24] proposed an SVM-based IDS with filter-based feature selection. They used the One-Vs-All technique for multiclass intrusion detection. The proposed Least Square SVM-based IDS (LSSVM-IDS) was evaluated using 10-fold cross-validation on the training data.

Yin *et al.* [25] proposed a new IDS by using Recurrent Neural Networks (RNN) and evaluated it using NSL-KDD for both binary and five-category classifications. In their study, they achieved 83.28% accuracy in binary and 81.29% in multiclass IDS on the NSL-KDD test+ set. In a study by Singh *et al.* [26], the researchers investigated the effect of the hierarchical representation of data for binary and multiclass IDS to reduce the false alarm ratio. The authors used RNN for training the IDS; however, they claim that RNN can be replaced with other fit and predict methods. They evaluated their proposed methods on both binary and multiclass labels.

Blanco *et al.* [27] proposed a CNN-based multi-class IDS with the Genetic Algorithm (GA) for feature selection. They evaluated the GA for feature selection on NSL-KDD and UNSW which improved the performance of the classifier. The authors did not include User-to-Root (U2R) attack in their evaluation because of its small sample size. Ding and Zhai [28] proposed a multiclass IDS using multi-stage features in CNN to improve their results. In their study, the best performance was achieved on the KDDTest+ dataset with 80.13% accuracy. In a more recent publication by Li *et al.* [29] the authors used data clustering to design a multiclass IDS based on multiple CNN models. They summarized the performance of each CNN model separately and combined it in their multi-CNN IDS. The study shows a significant improvement in the multi-CNN fusion method in comparison to the CNN models individually. Also, their testing result on binary and multiclass classification shows a higher accuracy on the binary IDS.

CICIDS2017 is another popular dataset for flow-based IDS design and evaluation. As explained by Kurniabudi *et al.* [30] this dataset represents a more recent real-world network traffic. In the mentioned paper, the authors used Information Gain (IG) for

feature selection on the CICIDS2017 dataset to examine the effect of feature selection on the performance of the IDS. They used several well-known classification models with a different number of features to evaluate their method. The result of their study shows different classifiers react differently to feature selection using IG.

Chiba *et al.* [31] used a hybrid deep learning approach for their proposed IDS. Their method involves Improved GA and the Simulated Annealing Algorithm as a hybrid model for classification. They evaluated their method on three different datasets using binary classification. The preprocessing steps used in the author's work include feature selection and class imbalance reduction.

Another type of hybrid deep learning model proposed by Malik *et al.* [32] employs Long-Short-Term Memory (LSTM) in combination with CNN to train an IDS with high accuracy. In this approach, the authors performed the training for both classifiers individually but used the combined test result as the final performance of the IDS. In their experiment, the authors only used 3 attacks plus the normal class for training the network. Similarly, Sun *et al.* [33] presented a hybrid CNN-LSTM IDS model that uses time-division and traffic segmentation as part of the preprocessing steps. They also demonstrated the effect of class weights on improving the performance of the IDS. Zhang *et al.* [34] have also used a CNN-LSTM hybrid model for training their experimental IDS. Unlike some hybrid models that need to train two networks for the classifier, the authors of this paper used a hierarchical model that enables concurrent training. The authors evaluated their IDS with both binary and multiclass classifiers with successful results.

Ferrag *et al.* [35] proposed an energy exchange framework for smart grids using blockchain and deep learning algorithms. The authors used RNN with truncated backpropagation through time for training the classifier. They evaluated the detection performance of the proposed method on 3 different datasets including CICIDS2017.

As can be observed from the above studies, CNN and LSTM are very popular in designing IDS models in the most recent studies. A more in-depth study of multiclass NIDS methods can be found in a study by Elmasry *et al.* [36].

Regarding the ADFA Linux and Windows datasets, they are both popular in A-IDSs. Anomaly detection is effective against zero-day attacks. However, it suffers from a higher false positive ratio in comparison to S-IDSs. The studies that are focused on HIDS based on ADFA datasets are mostly focused on the ADFA-LD and a few are focused on ADFA-WD while even fewer publications considered both ADFA datasets together.

Lv *et al.* [37] introduced a sequence-to-sequence system call prediction model that is intended to provide extra information for the classifiers. The authors used RNN for binary classification over the ADFA-LD dataset. Another binary classifier for the ADFA-LD is proposed by Serpen and Aghaei [38] using Principal Component Analysis (PCA) feature extraction and the k-Nearest Neighbor (kNN) algorithm. Vijayanand *et al.* [39] investigated the effect of the GA and Mutual Information-based hybrid feature selection on SVM-based HIDS using the ADFA-LD dataset. Tran *et al.* proposed a minimalistic CNN-based anomaly detection for host-based IDS [40]. The proposed system is assessed on the ADFA-LD dataset. Even though the performance of the system is not promising, the authors aimed to introduce CNN as an extended

usage for the said classifier and did not try to improve the performance by using a more complicated architecture.

A host-based anomaly detection system study by Shin and Kim [41] depicts comparison results among different ML algorithms on the ADFA-LD dataset. The authors used *n*-gram [42] for feature extraction and improved the Sequence Time Delay Embedding algorithm. The authors verified the result of their study using simulation tools with SVM, Logistic Regression, and kNN algorithms.

Khater *et al.* [43] proposed a lightweight perceptron-based HIDS by using *n*-gram feature extraction for ADFA-LD and ADFA-WD. The use of *n*-gram helps to maintain the relative order of the system calls which is a meaningful feature in each trace.

In a comprehensive survey by Mahdavifar and Ghorbani [44] the authors reviewed deep learning applications in cybersecurity. The authors explained and discussed publications since 2013 with a brief introduction to each mentioned deep learning algorithm.

2.2 VANET Specific IDS Models

The idea of the autonomous vehicular environment has been trending in the past few years and there are plenty of research papers published to address various aspects of this topic. In this section, we review the most recent publications regarding the security problems in VANET, especially the ones that cover the intrusion detection topic.

Sharma and Kaul [45] investigated the latest threats and challenges in VANET in a comprehensive survey. Additionally, they proposed an IDS framework that is aimed at detecting zero-day attacks by using a proactive honeypot-based system [46]. This

framework must have both S-IDS and a honeypot-based system. The honeypot nodes (either vehicles or RSUs) collect data from new attackers and create a new IDS rule if it does not exist. The authors did not evaluate their proposed method but mentioned the possible challenges such as honeypot node selection, the density of the said nodes, and the choice of existing IDS that goes along with the honeypot-based system.

Misbehavior in VANET can be defined by either describing a malicious or a faulty node behavior. van der Heijden *et al.* [47] focused specifically on misbehavior detection in VANET by surveying existing mechanisms. They analyzed the said mechanisms based on various qualitative aspects such as the impact of the system on computational resources and the privacy of the user. Additionally, the authors provided a brief insight on the open issues in misbehavior detection.

Hasan *et al.* [48] focused their research overview on intrusion detection in V2X communication. In this research, the authors reviewed the possible threats in the V2X environment and summarized existing solutions for a series of known attacks in this area.

In a more recent survey, Hbaieb *et al.* [49] focused on trust-based security systems for VANETs. The authors analyzed both classical and emerging trust schemes. They mainly evaluated the trust schemes based on their scalability, time complexity, communication overhead, privacy, robustness, and dynamicity. None of the reviewed models fully satisfied all criteria, and there is always a tradeoff in some places for the sake of performance or simplicity.

Aside from the older studies that are summarized in the above surveys and reviews, the topic of intrusion detection in VANET is still an ongoing popular subject that is actively studied by researchers. We reviewed the most recent and relevant proposed security methods for VANET since 2018. The rest of this chapter is organized by reviewing the proposed methods suggested by other authors in chronological order.

2018:

Zhang and Zhu [50] proposed a collaborative IDS that aims to also keep the privacy of the users intact. The IDS consists of a preprocessing engine for intrusion detection and a privacy-preserving engine based on ML. This method is an anomaly-based detection that uses vehicle behavior as well as communication data. The authors used the network data from NSL-KDD for evaluating the performance of the proposed Privacy-preserving Machine Learning based Collaborative IDS (PML-CIDS) model.

Subba *et al.* [51] designed a framework for VANET for clustering and intrusion detection using game theory and neural networks. The IDS uses a two-player non-cooperative game based on the Nash equilibrium while the clustering algorithm is created to ensure scalability in presence of greater traffic density. The Cluster Heads (CHs) also maintain a trust value for the cluster members based on their true or false alarms ratio. The framework is evaluated by VANET simulation using Network Simulator 3 (ns-3) [52] and Simulation of Urban Mobility(SUMO) [53] tools.

Shames *et al.* [22] proposed an SVM-based intrusion detection in VANET that uses trust score for isolating malicious vehicles from the network. In this method, every node that is participating in packet forwarding uses a modified version of the promiscuous mode to monitor the next hop in the route. The core of the IDS uses SVM

to detect attackers and consequently update their trust values. The trust value is shared with the other neighbors and is used in the route discovery process to help avoid malicious nodes. The dataset and evaluation of the proposed methodology are carried out in Network Simulator 2 (ns-2) [54] and SUMO simulation tools.

Security systems that rely on CHs in VANETs to do the heavy lifting of intrusion detection can exhaust the resources of the CHs and introduce a delay in the network. For this very reason, Sharma and Kaul [55] suggested a clustering system where multiple CHs share the responsibilities. They used a hybrid fuzzy-based decision-making system for selecting CHs. The authors also proposed an ML-based IDS using SVM for protection against malicious behavior. The performance analysis is conducted using NetSim, SUMO, and MATLAB.

2019:

A binary Distributed DoS (DDoS) classifier is proposed by Gao *et al.* [56] where the detection module is based on Random Forest Classifier (RFC). The algorithm consists of a standard data collection module that parses packets for feature extraction. The parsed data will be sent to the RFC-based detection module for classification. The model is evaluated using a combination of NSL-KDD and UNSW-NB15 [14].

Makhlouf and Guizani [57] designed a security scheme for the Ad-hoc On Demand Multiple Path Distance Vector (AOMDV) routing protocol that has a detection model as well as a disjuncture method for Route Reply (RREP) messages. In this method, the Route Request (RREQ) messages are not broadcasted instead it uses RBin bit to indicate whether a node has already accepted an RREQ or not. The RREP messages are also hashed to increase the integrity of the packets. Misbehavior detection is based

on several vehicle-related information such as the change in speed. The authors evaluated this protocol using ns-2 in presence of a blackhole attack.

A trust-based system for detecting DDoS attacks in VANETs is proposed by Poongodi *et al.* [58] using the GA. This method takes into account various statistics such as residual energy and successful transmission rate at a given time for updating the trust values of each vehicle. The trust value is used for cluster formation and CH selection as well. ns-2 is used for simulating DDoS attack in VANET and evaluating the effectiveness of the proposed algorithm.

Nie *et al.* [59] implemented an anomaly detection system using a combination of CNN and Reinforcement Learning (RL). The input for the first phase aims to estimate the traffic based on the special and temporal features of the vehicles. After the traffic matrix is estimated, the RL model identifies deviation from the expected normal behavior. This information is then used to identify malicious users in the network. The proposed method is evaluated under DDoS attack simulation using the Low Orbit Ion Cannon and traffic model using the Open Shortest Path First algorithm.

2020:

Tripathi and Sharma [60] proposed a trust-based IDS where the trusted authorities that are also known as observer nodes assess the trustworthiness of other nodes. The observers use metrics such as the number of dropped or modified packets to calculate the 1-bit trust value of other nodes. This makes the transmission overhead for the trust values negligible. The authors evaluated their method in presence of a blackhole attack using ns-2.

Tit-for-Tat (TfT) is a game theory-based strategy that is used by Mostefa *et al.* [61] to mitigate network intrusion in VANET. In this method, the nodes fit into three categories, safelist, watchlist, and blocklist. The first time a node is detected as malicious it will be put on the watchlist and if the malicious behavior is repeated once more, it will be put on the blocklist and consequently isolated from the network. The authors used ns-2 and SUMO to simulate two types of active attacks, namely blackhole and jellyfish attacks for evaluating their method.

Zhou et al. [62] proposed a distributed collaborative IDS against DoS and spoofing attacks in VANET. In this method, the clusters are formed by selecting a trusted node as the CH based on their reputation value. The algorithm collects invariants such as traffic and communication flow, for anomaly detection using Distributed Collaborative Intrusion Detection system based on Invariant (DCDIV). The simulation part is conducted using SUMO, TraCI [63], and OMNET++ [64] tools for evaluating the effectiveness of this method. The validation of the method is done by the authors using ns-2 simulation in a 3-junction road map.

Kolandaisamy *et al.* [65] used a method known as packet marking for detecting DDoS attacks in VANETs named as Packet Marking based on the Adapted Stream Region (PMBASR). In this method, traffic data is collected by RSUs are sent to the IDS for analysis and detection.

In order to create a more realistic approach to IDS in VANETs, Rahal *et al.* [66] generated a real-world wireless communication between two vehicles. The dataset is generated in three different environments. The VDoS-LRS includes three types of DoS attacks that are generated during the communication between the two vehicles. The

authors then assessed the dataset against five different ML models such as SVM, RFC, Naïve Bayes (NB), kNN, and Decision Tree Classifier (DTC). All mentioned classifiers performed well in both binary and multi-class classifications. RFC and DTC are the top performers in terms of average accuracy in all results.

Bensaber *et al.* [67] proposed a fuzzy logic-based model known as the Adaptive Neuro-Fuzzy Inference System (ANFIS) for predicting the security index of vehicles. This method is a combination of neural networks and fuzzy logic in one solution. The authors created the IDS dataset using OMNET++, VEINS [68], and SUMO simulation tools.

A spline-based intrusion detection method for VANET is proposed by Schmidt *et al.* [69]. In this method, the spline-based classification is combined with clustering for the core of the IDS. The authors reduced the overfitting problem of complex splines by using knot flow classification. For evaluation, they used the NSL-KDD dataset divided in an 80-20 train-test ratio in binary form. The features are also reduced from 41 to 3 by using the Principal Component Analysis (PCA) method.

Ghaleb *et al.* [70] proposed an on-demand IDS that enables vehicles to share their locally trained classifier with other neighbors. The sharing process starts when a neighbor requests it, and upon receiving the IDS it will be evaluated against local test data. The classifier is based on the random forest algorithm which in the end will be used in an ensemble form. The ensemble is weighted based on the performance of the local and imported datasets that were evaluated on the local test dataset earlier. They assessed the performance of the IDS using the NSL-KDD dataset in comparison to other IDS models.

2021:

Alsarhan *et al.* [71] also used the Bayesian learner with the combination of a rule-based security filter, and the Dempster-Shafer adder for detecting attacks on safety message delivery. The authors evaluated their proposed IDS using simulation tools. The performance of the network is assessed against the KDD dataset as the authors claimed, however, there is no mention of which version of the dataset is used.

Alsarhan *et al.* [71] proposed another ML-based IDS using SVM. In this study, the researchers used GA, Particle Swarm Optimization, and Ant Colony Optimization for optimizing the model. The authors used the NSL-KDD dataset which is an infrastructure-based dataset to evaluate their method. The results from 10-fold cross-validation show that the GA provided the best performance.

Information and identity attacks are a common concern in the IoT which can be related to VANETs as well. Stepien and Poniszewska-Maranda [72] have proposed an algorithm that can detect Sybil and Bogus attacks [73], [74] using the behavior and footprint of VANET users. This algorithm is an improvement over the previously existing Bogus & Sybil Trust Level & Timestamp algorithm [75].

Velayudhan *et al.* [76] have integrated the chaotic map into the Elephant Herd Algorithm (EHA) optimization algorithm to improve the classification ability of their proposed deep neural network IDS for VANETs. Their proposed method also includes cluster formation and CH selection prior to intrusion detection. This study also includes an encryption method called MD5-ECC for the approved CH communication with the cloud server. The evaluation is done on a publicly available dataset, however, there is no mention of the dataset name.

Raja *et al.* [77] provided a solution for the software-defined network in VANET that focuses on security and energy efficiency. The proposed method is defined in two levels. It starts by using RSU-based group authentication for authenticating vehicles that are joining the network and an ML-based IDS for detecting malicious nodes. They evaluated the proposed IDS using the NSL-KDD dataset.

A hybrid IDS is formed by using multiple steps for detecting misbehavior in the system. An example of such a system is presented by Bangui *et al.* [78] where the authors used RFC and coresets clustering in a hybrid manner. The RFC-based module tries to detect an attack based on the signature of an attack. However, if no signatures were detected, the anomaly detection-based module examines the data for any anomalies. When an attack signature or anomalous behavior is detected by the IDS, the information will be sent to the decision module to take proper action. The authors used CICIDS2017 to evaluate their method.

Gonçalves *et al.* [79] proposed a hierarchical IDS using ML methods. This IDS is trained to identify flooding DoS and fabrication attacks by collecting special and time related information. The information is collected only from the nodes in the same cluster. The cluster information is extracted from the previously simulated dataset that was generated by the authors [80] using SUMO and ns-3. The authors filtered the data by platooning which is placing the vehicles that drive in the same path in the same group or cluster. The authors evaluated several ML algorithms in binary mode classification.

Sedjelmaci *et al.* [81] proposed a hierarchical cooperative game-based framework for protection against network intrusion. The hierarchy consists of head and secondary

agents that cooperate to detect attackers in VANET. The detection is done by the secondary agent while the decision at the time of intrusion detection is done by the head agent. The framework adds a small overhead to the communication that is an acceptable tradeoff for security. SUMO and ns-3 are used for simulating and evaluating the proposed hierarchical method.

Shu *et al.* [82] designed and presented a collaborative IDS using deep learning. In this method, the CH is in charge of performing intrusion detection. The IDS introduces a marginal overhead which is acceptable. The authors used the KDD99 dataset for their experimental results.

2022:

Sharma and Jaekel [83] took advantage of Basic Safety Message (BSM) data for detecting misbehavior in VANETs. Specifically, the authors targeted the position falsification act that can cause serious problems in traffic management. Since BSM includes positional information as well it can be maliciously used to mislead other users. The authors of this paper proposed an ML solution based on four different ML algorithms. The input of the IDS is a combination of two consecutive BSM messages to improve the detection. The authors used the Vehicular Reference Misbehavior (VeReMi) [84] dataset to evaluate their method.

Reputation-based IDS is a concept that allows the response module to act based on the reputation value of a user. Najafi *et al.* [85] proposed a decentralized system where the malicious user is detected using a reputation system that is based on the Bayesian theorem and watchdog system. The watchdog system assesses the reputation value of a neighboring user by overhearing their network communication. Therefore, it can

detect if the surveyed user is dropping or forwarding the received packet. The watchdog collects this information and uses the Bayes theorem to predict the probability of malicious behavior. The authors used MATLAB for simulating and evaluating their method.

Lavanya and Kannan [86] designed and evaluated a clustering technique for VANET routing based on fuzzy logic with an addition of a lightweight Fuzzy SVM (FSVM) based IDS for anomaly detection. The main parameters for CH selection are based on the velocity, ID, direction, and location of the nodes. For the FSVM optimization, the authors used the Krill Herd (KH) optimization algorithm. The ns-3 is used for investigating the results, where the proposed method shows improvement in the network stability and performance compared to other methods.

Chapter 3

PROBLEM DEFINITION

3.1 Information and Communication Security Threats

The threat of cyber-attacks existed in the early days when home computers were gaining popularity. These threats became more widespread with the introduction of the Internet to households. Nowadays, we can categorize cyber-attacks as network-based or host-based. The first group takes advantage of the security holes in network protocols or architectures, while the latter target the vulnerabilities of the operating system or firmware of a device.

In Chapter 1 we had a small discussion on the vulnerability of VANETs because of their architecture and communication medium. In this chapter, we are going to discuss both network-based and host-based attacks briefly. We also investigate the research gap and information availability in this regard.

3.1.1 Network-based Intrusion

Based on telemetry information gathered in 2021 by Bitdefender [87] the most common attack on IoT devices is DoS, followed by overflow attack. DoS attack is a type of network-based intrusion where the attacker targets the availability of a network or service. For example, in a UDP flooding DoS attack, the malicious user floods the network channel by sending a large amount of junk data to one or more devices. This will cause the network bandwidth to be filled and prevent other legitimate users from communicating with each other or with a service provider. There are several types of

DoS attacks, and they can be executed using several infected devices, which in this case it is known as DDoS attack. Nevertheless, the network-based DoS attacks have one thing in common, which is affecting the flow of network data in a damaging manner.

Here we are going to have a brief introduction to two types of DoS attacks that we used in our simulations:

- *Blackhole attack*: in this type of active attack the intruder indicates that it has network routes available to a requested destination from other nodes. However, when they receive packets from other nodes, instead of forwarding them to the next hop or destination they will drop the packets without informing the source nodes about it.
- *Greyhole attack* is another active attack remarkably similar to the blackhole attack, however, instead of always dropping the incoming packets, the attack will drop the packets randomly. This makes the detection of this kind of attack more difficult compared to blackhole attack.
- *Delaying attack*: similar to the above attacks, it is an active type of attack where the malicious user must be first selected as part of a network route and then executes the attack by permanently or randomly delaying the packet forward process.

Other than DoS attacks, we investigate the passive type attacks where the malicious user does not directly affect the flow of the data in the network. In this type of intrusion, the attacker often targets the confidentiality of the data or a service. It is done by eavesdropping or redirecting the messages. One of the most severe attacks in wireless networks is known as wormhole attack:

- *Wormhole attack* is a passive type of network attack that is executed by at least two malicious nodes placing themselves in strategic locations within direct reach of each other [45]. These nodes advertise to have the shortest pass through their established channel to have a higher chance of being selected for routing messages. They can also use a secondary wireless device with higher transmission power and range for this purpose. The wormhole attack can be simply used for eavesdropping, or it can be used to execute other types of attacks such as replay attack [88] or impersonation attack [89].

As we mentioned in the introduction and related work sections, there are various research and publications that are dedicated to network-based intrusion detection. Many network-based datasets are available as a result of these studies that are either collected using flow-based or packet-based data collection [90]. However, almost all of them are collected from infrastructure-based networks, and efforts such as [66] while a great contribution, are limited. Other issues concern the privacy of the collected data [50] or the type of intrusions in the datasets where most of the attacks are active types. This is why in this study we focused on data collection method that does not violate the privacy of the users and also includes both active and passive types of attacks.

3.1.2 Host-based Intrusion

Referring to the mentioned report from Bitdefender, the overflow attack is the second most common attack in IoT. This type of intrusion targets vulnerabilities in the code execution of the device firmware or operating system, hence, the name host-based intrusion. All IoT devices including the OBU of the vehicles need to have proprietary or standard firmware for running their applications. Attackers analyze the firmware

and find vulnerabilities in the codes that allow them to harm the system. Mullen and Meany investigated the overflow attack in IoT systems and its effect [91]. Overflow attack is one of the many host-based intrusions that exist in both IoT and non-IoT spaces and one of the best solutions against these kinds of cyber-attacks is using A-IDS or S-IDS.

Similar to NIDSs, for designing and implementing a proper HIDS, we need appropriate data. Investigating HIDS datasets is not the scope of this study, however, dataset structures such as the ADFA-LD for Linux and ADFA-WD for Windows are good examples of host-based data collection that can be sued for both A-IDS and S-IDS. Also, since our simulation tools are limited in simulating OBU firmware we used the abovementioned datasets for evaluating our proposed multi-class IDS.

3.2 VANET Mobility Traces

Numerous attempts have been made for generating vehicular mobility traces for various applications. Be it for traffic management programs or VANET simulations. The available traces are either created using simulation software or using Global Positioning System (GPS) data that are gathered from certain vehicles. Mobility traces such as Rome [92], San Francisco [93], and Beijing [94] are gathered using GPS data from taxi drivers, whilst traces such as Cologne [95] are created using simulation software.

However, there are shortcomings in many of the cases above that are explained by Celes *et al.* [96]. The authors explained the issues in terms of gaps in the trajectories in the traces. This can be simply explained as the lack of coordinated data for a certain vehicle during mobility. This can cause an unrealistic mobility pattern which in turn

makes the simulation lack its realism. The mentioned paper introduced a way to improve the traces by using a reference system and calibration methods to fill the gaps. While the proposed method has helped improve the existing traces, in this research we aim to create a more recent trace that considers more issues than just trajectory gaps.

In a realistic scenario, the movement of a vehicle is not entirely predictable. Variables such as speed, turning points, and takeovers need to be all considered. That is why in this research we are using the most recent version of SUMO [97] software for creating a realistic mobility trace that includes both public and private vehicles in the city of Famagusta. This way, we can solve the issue of some of the existing datasets that are limited to only taxi drivers' data or low trajectory updates due to old software. Figure 3.1 shows the generated map for our VANET simulation using SUMO.

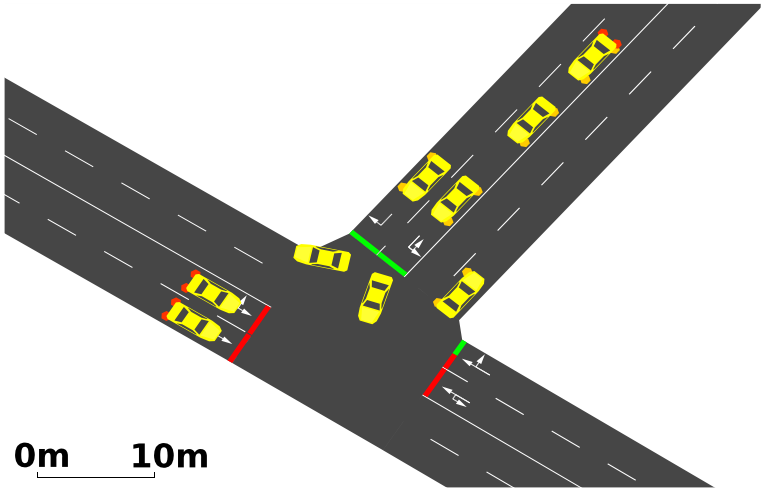

Figure 3.1: Vehicular mobility simulation area using SUMO

3.3 VANET IDS Dataset Generation

The main target of this study is to create an intrusion detection system that can negate the effect of network attacks in VANETs. After creating our mobility trace, we used the ns-3 to create the network data for normal and abnormal behavior. For the abnormal

behavior, we chose the blackhole and wormhole attacks for active and passive types of intrusions, respectively. We also aim to avoid creating any overhead for the network. Therefore, we needed to find a solution that is both accessible and does not require extra data transmission. Network simulation tools such as OMNET++, ns-2, ns-3 are the most common tools for evaluating IDSs in literature. We used ns-2 in our previous studies for evaluating SVM-based IDS in MANET [98] and VANET [17]. However, the ns-2 is no longer actively maintained, and we decided to use the ns-3 for our current studies.

The ns-3 tool provides a variety of tools for extracting simulated network statistics in various scenarios. For instance, it is possible to record the network traces in pcap format which is a popular format for professional network diagnostics. It is also a format that Sharafaldin *et al.* [10] used for extracting new features for the CICIDS2017 dataset. However, one of the challenges of evaluating an IDS is to do it in real-time during the simulation, hence, we used one of the most useful modules of ns-3 which is flow-monitor. We provided the details of our initial feature extraction using the flow-monitor module in Section 5.4.2.

In the coming chapters, we introduce our proposed solutions to the mentioned issues in this chapter. We start with the SVM-based binary IDS for VANET followed by the multi-class CNN-based IDS solution for both network-based and host-based IDSs.

3.4 Computational Complexity

ML algorithms such as SVM and CNN have solid architectural differences and basic similarities. This means different computational requirements as well. Many IoT devices are heavily constrained by computational power and energy consumption.

Therefore, it is crucial to understand what is the computational complexity which is also usually calculated as the time complexity of algorithms that we use in the IDS. To understand the computational complexity of each algorithm we use the Big-*O* notation to describe the impact of each algorithm at the end of the corresponding performance analysis of each model.

Chapter 4

SUPPORT VECTOR MACHINE-BASED BINARY IDS

Vehicles may have more resources compared to other small portable devices that often exist in the MANET environment, nonetheless, it is still constrained by their limited computational power and energy. The processing constraint can be addressed with edge and cloud computing, however, that also adds another security risk. As mentioned, an IDS can be programmed to alert the user in presence of intrusion without detecting the type of the attack and it is still particularly useful.

SVM is one of the best binary classifiers because of its unique classification technique using support vectors. The support vectors combined with the kernel functions provide a huge advantage for separating the two classes using a hyperplane while remaining computationally efficient with a lower number of input features. Therefore, we decided to use this method for an IDS that allows every vehicle to monitor other participating users during a routing process to ensure reliability. The remainder of this chapter is the result of the study that we have published previously in [22].

The main purpose of this method is to design and implement an IDS that does not interfere with the normal packet routing process of VANETs while maintaining high performance. In other words, the proposed system should not create any packet overhead. Furthermore, every active node or vehicle in the network route should carry on the detection process with high accuracy and speed.

In the proposed Trust-aware SVM-based IDS (TSIDS) model every vehicle gathers network information only if necessary through a modified promiscuous mode and uses the SVM-based IDS to detect malicious users and consequently update the designated trust value of that user. For a better understanding of our IDS design, we consider a three-components cross-layer based IDS [1] to discuss every mechanism separately and in order of execution.

4.1 Packet Header Analysis

The first component of the TSIDS collects relevant data for further processing in the detection system. For data gathering, we use promiscuous mode with altered behavior for more efficiency and security to capture and analyze packet headers.

In general, every network node only accepts and processes data packets that are broadcasted or addressed to them and ignores the rest. Promiscuous mode changes this state and enables the node to capture data packets in its reception range. Although it is also a method for some malicious activities [99], in this study, we modify this method to be effective in faster data collection for the IDS and ignore the packet payload to prevent any confidentiality issues. By default, this causes a lot of processing power and energy source to be spent when receiving numerous packets and is especially inefficient in crowded areas with higher-than-normal network traffic. To avoid this situation, our designed data gathering module only activates during packet routing and only captures the packets which have their own address labeled as the previous hop in the header. This allows every node or vehicle simply monitor the next hop and detect any misbehavior when it is affecting packet delivery on their own route. This mechanism works in parallel with normal packet routing and will not queue any packets, thus avoiding bottlenecks due to capturing packets, and since it acts highly

selective, there will not be unnecessary processing power consumption on irrelevant data.

To find a pattern that effectively addresses the presence of malicious nodes we consider three parameters that affect the throughput of the network. Every network packet header contains the address of the previous hop, also known as the forwarding hop, as well as the address of the next hop. By considering the mentioned parameters and the time of arrival and departure of each packet, we selected relevant information for the data gathering module. The selected parameters are packet drop rate, packet transfer delay, and packet transfer interval; all considered in one second period throughout the network activities, and each of the parameters is collected by the initial source node or an intermediate hop from their next hop if it is not the destination.

The first parameter to monitor is Packet Drop Count (PDC), which is the number of packets that are dropped at the next hop. An intruder or a selfish node can intentionally drop random packets to interrupt the consistency of the network, or to save their own resources. This affects packet transmission by lowering the successful packet delivery, which consequently interrupts data reception at legitimate nodes. Every node that sends a packet to a non-destination node keeps track of its own sent packets and counts any packet that the next hop does not forward. This mechanism enables packet drop tracking regardless of the transport protocol, *e.g.*, connectionless protocols.

The second parameter is described as Packet Transfer Delay (PTD), which is a variable of the time measured between the transmission delay from one hop to the next, together with process time and transmission delay of the same packet from the latter hop that is captured by the previous hop. For instance, Node-A is transferring a packet to Node-

B, which is its next hop and expects Node-B to forward it to the next hop or destination. PTD calculation starts with recording the departure time when Node-A sends out the packet to Node-B. When Node-B receives the packet, it starts processing and then forwards it to the next hop or destination. At this time if Node-A is still in the coverage area of Node-B, it captures the packet and calculates PTD based on the previously recorded departure time. This parameter distinguishes between a genuine node with low process time and low PTD in comparison to a node that might be an intruder or selfish with unreasonably higher values. The data collection module gathers the average PTD in one second period and is measured in seconds.

The third parameter on the list is represented as Packet Forward Interval (PFI). It is the time interval in seconds between two adjacent packets that are forwarded by the next hop. When a sequence of packets is being sent from a node, usually the forwarding interval between two adjacent packets is close to a stable number. However, when there is a packet drop or a malicious queueing delay, PFI starts to fluctuate, depicting an unwanted behavior in the network. This value is also collected as average in every one-second interval. This concludes all three parameters in the gathering module. The above parameters are then used as the feed of the SVM in the detection module.

4.2 SVM Intrusion Detection Module

After collecting the previously defined statistics for intrusion detection, they are used in the detection module for additional analysis. SVM is a reliable ML tool for various non-linear classification scenarios [100]. This makes it a suitable method for identifying network intruders by providing applicable parameters as the input of the machine, and its effectiveness is proven in different environments before [98]. SVMs are supervised ML tools; therefore, to train one with a high detection ratio for our

issue, we need to run a few samples of VANET scenarios with the presence and absence of network intruders.

The simulations for SVM training and testing are performed by including 30 vehicles in the area, which start at a random starting point of a road and move within the defined roads with the presence of traffic lights; at each junction turn, vehicles have an equal chance to choose their next destination. Simulation parameters are included in Table 4.1. We generate two types of setups with the only difference being the presence and absence of attacker with normal and malicious outputs respectively, and an extra setup at a slightly loaded network with no intruder that is classified as normal output. The load level is not high and still keeps the network performance close to optimum; its purpose is for preventing a biased classification during slight congestions. More details about the simulation tools and parameters are available in Section 4.4.

Table 4.1: Simulation Parameters (ns-2, SUMO).

Parameters	**Values**
Simulation Area	Urban: 1000 × 1000 m^2 Highway: 2500 × 70 m^2
Simulation Time	1000 s
Number of Vehicles	10 / 30 / 50 / 76
Vehicle placement / Movement	Random Starting Points with Random Junction Turns
Vehicle Speed	Urban: 0 Km/h to 50 Km/h Highway: (80 / 100 / 120 Km/h) ± 20%
Mac Protocol	IEEE 802.11p
Routing Protocol	Ad hoc On-Demand Distance Vector (AODV)
Network Traffic Model	Constant Bit Rate (CBR)
Network Traffic Rate	64 Kbps
Packet Size	512 B
Random Noise in CBR	Disabled

After accumulating the training and test data from the generated simulations, we train our SVM with random inputs in each iteration until the best result is achieved. In each training, the input data is normalized before training. The result is an SVM, which identifies malicious behavior in the network and passes the results to the response module that has its own set of rules to generate a final decision.

4.3 Trust Value Table

To make a final decision on isolating a malicious node from the network, our response module establishes a shared Trust Value Table (TVT) among the neighborhood of every node. Each node has its own TVT that includes an initial trust value entry for every newly registered neighbor. The values are updated according to the output from the detection module. Normal TVT update broadcast is periodical while in the case of intruder detection it is on-demand. Trust values have an initial value between the maximum and minimum. It cannot go above the maximum value to avoid exploiting extremely high trust value. The trust value increment and decrement rate can be the same or different. The trust values are calculated by (4.1) and when the trust value reaches the minimum, the corresponding node is identified as malicious.

$$TVT_i^{new} = \begin{cases} TVT_i^{old} - d_t & \hat{y}_i > 0 \\ TVT_i^{old} + g_t & \hat{y}_i \leq 0 \end{cases} \tag{4.1}$$

where TVT_i and $\hat{y}_i$ are the i^{th} node trust value table entry and the IDS output respectively, d_t is the trust value decay, and g_t is the trust value gain.

Every time a node evaluates the performance of its next hop by using the detection module it either decreases or increases the current trust value associated with the said hop to represent the current likelihood of a node being an attacker or not. The maximum value in TVT for this IDS is set as 10 and the minimum as 0 with both gain and decay set to 1. When the trust value of a node reaches zero, the response module

triggers an alarm, which sends a route error message to start a new route discovery while ignoring any route reply messages from the malicious node or any other node with zero trust value. This value is also broadcasted to the immediate neighbors to prevent future intrusion from the same node in other locations.

Figure 4.1 depicts a block diagram of the TSIDS and below is the workflow of the algorithm in form of pseudo-code. Regarding the time complexity of the TSIDS algorithm above, where the variables are the number of network packets and active routes, the time complexity of each mechanism is as follows. Calculation of PDC, PTD, and PFI increase linearly by the number of packets going through the current vehicle. The detection module runs once every second within a constant time for each network route that is passing the current vehicle. This means the SVM function, which always runs with a fixed number of inputs, has to run once per second for every active route at the vehicle. The response module also behaves similarly and updates TVT when there is an active route available or when it is broadcasted from other neighbors. Therefore, the entire algorithm processing time increases linearly, which means the speed of the detection process will not be affected drastically in a crowded environment.

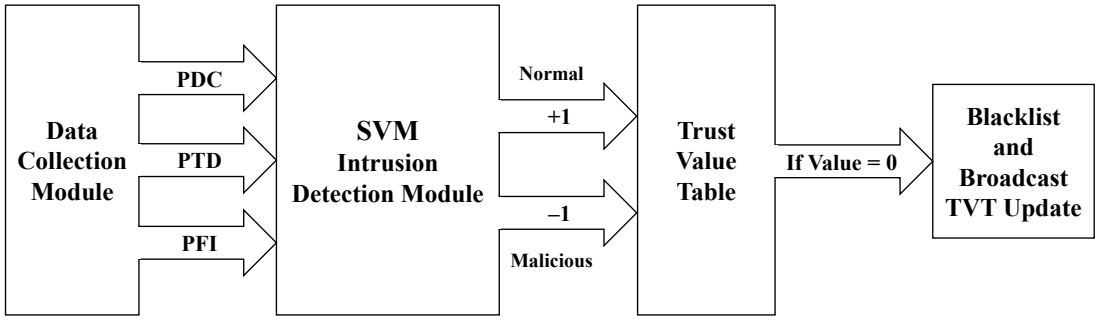

Figure 4.1: TSIDS block diagram.

TSIDS Algorithm

Input: *packet header*

Output: *attack, no attack*

// Data Gathering Module

1. ***if*** *(node i is my next hop and not a destination node)* ***then***

2. *calculate (PDC_i)*

3. *calculate (PTD_i)*

4. *calculate (PFI_i)*

5. ***end if***

// Detection Module

6. *svm_ids(PDC_i, PTD_i, PFI_i)*

// Response Module

7. *response_m(svmOutput)*

8. ***if*** *(svmOutput = 1)* ***then***

9. *decrease $trustValue_i$ by 1 down to 0*

10. ***else***

11. *increase $trustValue_i$ by 1 up to 10*

12. ***end if***

13. ***if*** *($trustValue_i$ = 0)* ***then***

14. *set node i as malicious*

15. *broadcast TVT*

16. *send route error message*

17. ***end if***

After completing the IDS design, to find an optimal initial value with the highest performance we evaluated network performance under different default values. The result of this evaluation is presented in Section 4.4 and the practical performance of the TSIDS is presented in Section 4.5.

4.4 Performance Analysis

To assess the result of our study, we prepared distinct setups of computer simulations by employing MOVE (Mobility model generator for Vehicular networks) [97] together with micro-traffic simulator SUMO for generating urban and highway scenarios for vehicle mobility as shown in Figure 4.2, and ns-2 [54] for the network simulation. Each setup focuses on the different variables that can affect the performance and behavior of the network in VANETs. There are three main scenarios with the different numbers of vehicles and attackers. The attackers or malicious nodes are vehicles that interrupt normal packet routing by dropping or delaying packets.

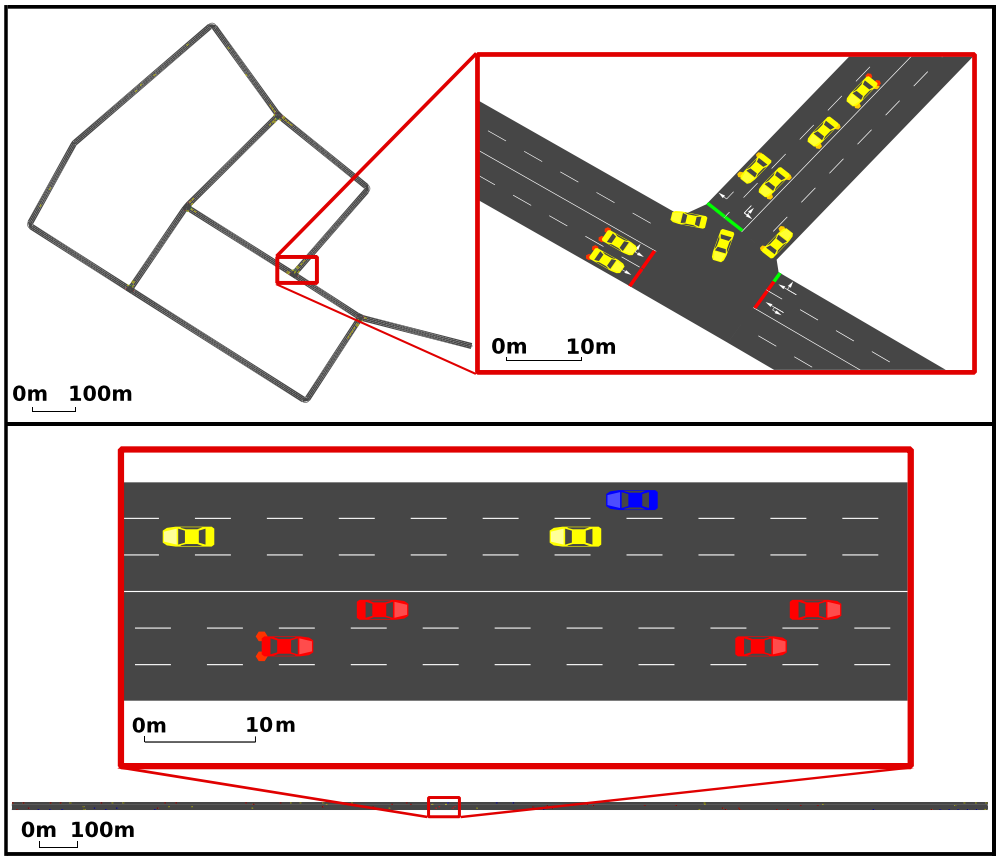

Figure 4.2: Urban (top) and highway (bottom) mobility models in SUMO.

To extend the reliability of the test results in comparison to real-life scenarios, we adjusted the behavior of the vehicles in SUMO in a way that actions such as alteration

of the vehicle speed and lane changes occur according to the class of the vehicle, the rules of the road, and random deviations. For example, in a highway setup, there are three types of vehicles with different maximum speeds and classes to prevent every vehicle from only choosing the high-speed lane. Additionally, the speed deviation can cause variation in speeding and lane changing on the road. The vehicles enter the road from a random edge and lane, choose their next destination randomly and exit the road through a random lane. This eliminates a controlled environment with a completely predictable outcome, leading to a more dynamic environment. The environment itself is divided into urban and highway maps. As it can be seen in Figure 4.2, and the parameters in Table 4.1, the urban scenario includes two lanes for each edge with a speed limit of 50 kilometers per hour, with traffic lights at three-way junctions. The highway scenario is 25000 meters long, consisting of two edges with three lanes for each edge, the lane speed limits are 80, 100, and 120 kilometers per hour accordingly.

After the preparation of the simulation environments and the TSIDS, simulations are conducted using the parameters presented in Table 4.1. We repeated each simulation scenario 50 times to obtain an average response for normalizing the effect of random variables such as driving speed and behavior of vehicles in road junction turns.

We divide the performance analysis into three sections, which include SVM classification performance, in the beginning, TVT evaluation in the next part, and finally, we discuss overall performance improvement in presence of TSIDS.

4.4.1 SVM Classification Performance

The results of the urban scenario in Table 4.2 show that the trained SVM classifier identifies malicious behavior in the network at a high success rate. We evaluate the classification performance of our SVM by real-time computer simulation using the

same tools as mentioned before, by analyzing the Precision and Recall of the trained SVM. The scenarios are selected in a way to be different from the training data. The data in Table 4.2 represents the classification analysis in detail through different scenarios. As it is shown, the trained SVM maintains a high level of Precision and Recall in various scenarios with the different total number of vehicles and malicious nodes.

Table 4.2: Performance of the trained SVM in terms of Precision, Recall, and F-Score in urban scenarios.

Number of Vehicles	Evaluation Metrics	Number of Attackers					Average
		1	2	3	4	5	
10	**Precision %**	100.00	100.00	99.99	100.00	100.00	100.00
	Recall %	97.15	97.02	96.61	96.63	96.55	96.79
	F-Score %	98.55	98.49	98.27	98.29	98.24	98.37
30	**Precision %**	99.99	99.99	99.99	99.99	99.99	99.99
	Recall %	96.69	96.76	96.60	96.54	96.58	96.63
	F-Score %	98.31	98.35	98.27	98.24	98.26	98.28
50	**Precision %**	99.97	99.99	99.98	99.99	100.00	99.99
	Recall %	96.99	96.97	96.70	96.75	96.75	96.83
	F-Score %	98.46	98.46	98.32	98.34	98.35	98.38

It can be observed that the classifier maintains remarkably high Precision performance in a way that it has on average a very low or almost no False Positive (FP) during the simulations. This lowers the chance of classifying genuine vehicles as attackers by mistake. The Recall is as high as 97% on average throughout all the simulations. Hence, considering the harmonic mean of the Precision and Recall which is known as the F-Score, we can see that the total classification performance is above 98%.

The follow-up in Table 4.3 is the result of the trained SVM in the highways scenario with 50 vehicles and 5 attackers in the area. It can be observed that the change in the physical topology of the area from urban to the highway and the increase in the speed of the vehicles affect the functionality. The performance differences are roughly 7%, which still keeps the F-Score above 91%.

Table 4.3: Performance of the trained SVM in terms of Precision, Recall, and F-Score in the highway scenario with 50 Vehicles and 5 Attackers.

Precision (%)	Recall (%)	F-Score (%)
93.89	88.62	91.18

4.4.2 Trust Value Analysis

As mentioned in Section 4.3 we measure the initial trust value to find a value with the best results and carry on the rest of the simulations based on this outcome. This setup contains 30 vehicles with 4 of them being malicious nodes. In our observation, setting the initial value to 3 will increase the best performance in terms of Packet Delivery Ratio (PDR) and End-to-End Delay (EED) compared to the other values. The results presented in Figure 4.3 and Figure 4.4 show that the best result is achieved when the initial trust value is set to 3 and the lowest performance is obtained with the initial value of 7.

After evaluating the core of the two major components which are SVM in the detection module and TVT in the response module, we initiate the performance analysis of the TSIDS using real-time simulation with all the modules working together in Section 4.5.

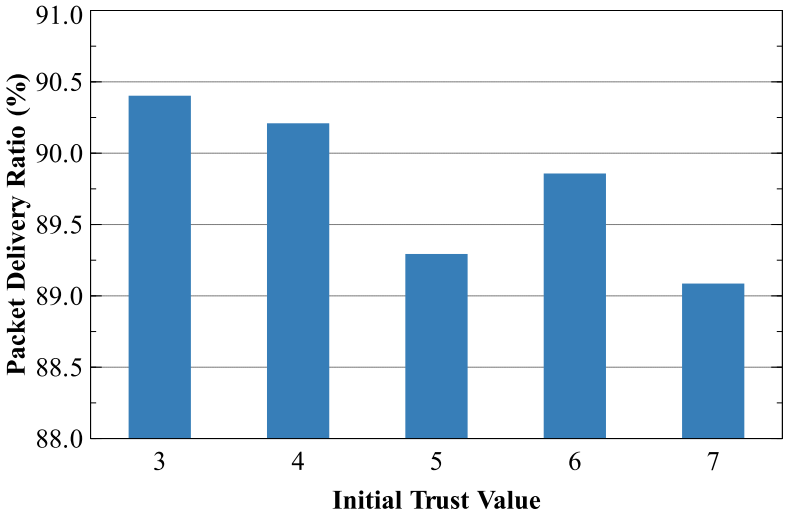

Figure 4.3: Initial trust value performance against PDR with 30 nodes and 4 attackers.

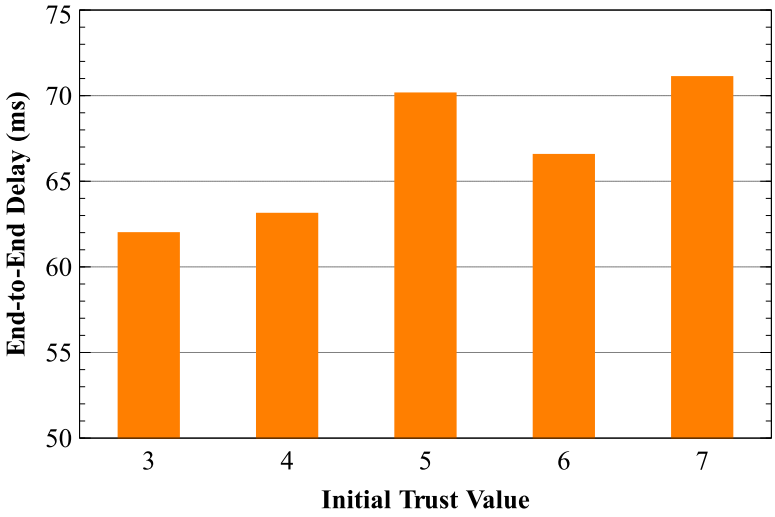

Figure 4.4: Initial trust value performance against EED with 30 nodes and 4 attackers.

4.4.3 Computational Complexity of SVM

Generally, for an IDS, the classification time complexity is more important than the training time complexity. However, it is worth noting that the training time complexity for SVM is $O(f \times q^2 + q^3)$ in the worst case, where f is the number of features, and q is the number of samples . For our IDS we only used three features which is a notable advantage in the training time of the SVM as well as the classification time complexity

which is $O(v \times f)$ where v is the number of support vectors. The classification time complexity is also very when we have only three features.

4.5 TSIDS Simulation Result

Two main criteria that we consider for the performance analysis are PDR and EED. The mentioned measures are directly influenced by the number of packets that are dropped during the data packet transfer and the transmission delay caused by packet queueing and processing time. The TSIDS performance analysis is done thoroughly in the urban area with a brief comparison between urban and highway scenarios in the end.

Two major points should be noted in the results. The first one is the chance of network routing participation. The attackers need to be in the physical range of data transmission and be able to participate in packet routing to have a chance to affect the network. This means that with more attackers in the area there is a higher chance of a network attack occurring, and potentially requires more time to eliminate the malicious vehicles. The second point is the ratio of attackers to the total number of vehicles. When the total number of vehicles increases in the simulation area and the number of attackers remain fixed, it reduces the chance of malicious vehicles being selected as intermediate hops and affecting the network. Hence, in the scenarios with a lower number of vehicles, the performance drop is more severe. The rest of the performance analysis is carried on by keeping these points in mind.

The effect of the TSIDS on the PDR of VANET in three different network sizes with the different numbers of malicious vehicles in the urban map is represented in Figure 4.5, Figure 4.6, and Figure 4.7. As it is visible from the graphs, introducing one

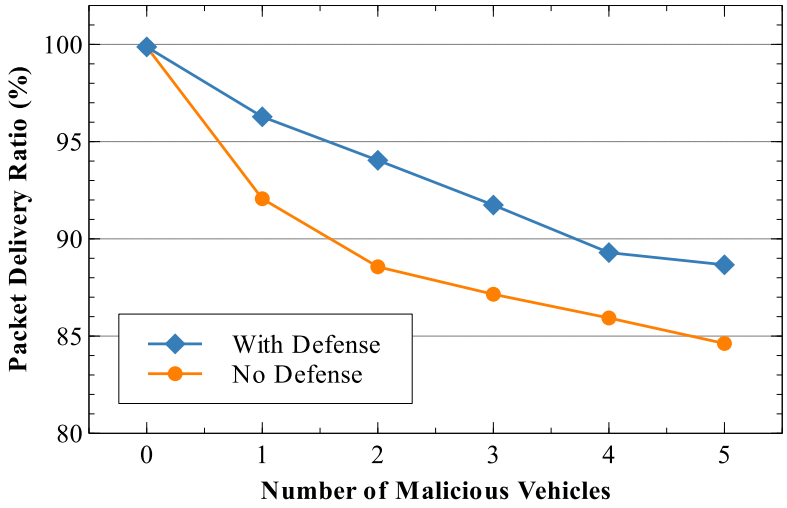

Figure 4.5: TSIDS performance analysis for PDR in 10 vehicles scenario.

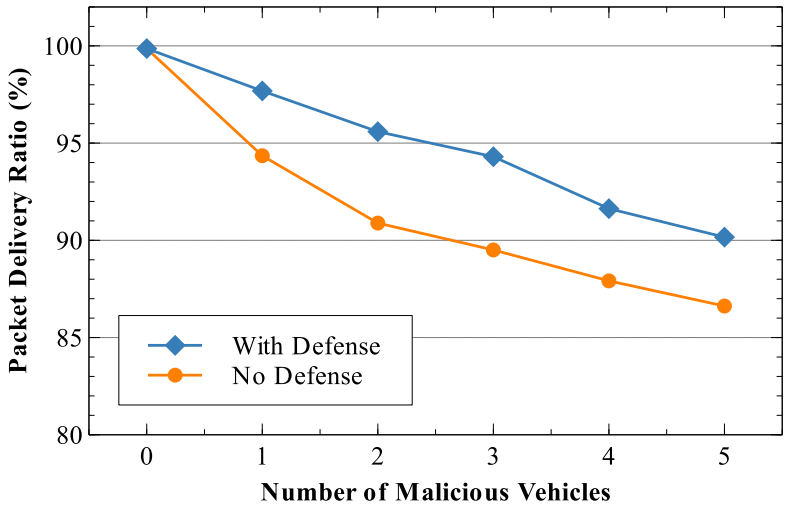

Figure 4.6: TSIDS performance analysis for PDR in 30 vehicles scenario.

or more attackers to the network decreases the PDR gradually, and equipping the rest of the vehicles with the TSIDS improves the PDR notably by eliminating the malicious vehicles.

Monitoring the average EED of the VANET in our urban simulations, we observe a similar improvement in the network in all different scenarios. The results in Figure 4.8, Figure 4.9, and Figure 4.10 illustrate an EED reduction in the presence of the TSIDS,

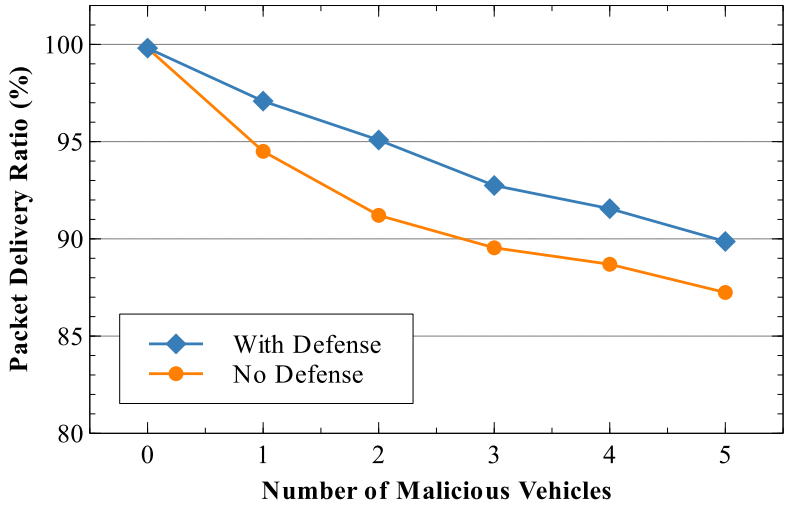

Figure 4.7: TSIDS performance analysis for PDR in 50 vehicles scenario.

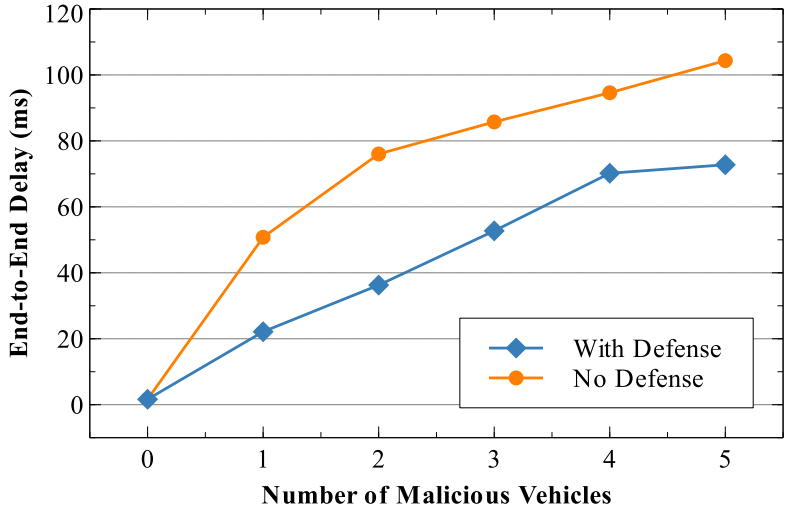

Figure 4.8: TSIDS performance analysis for EED in 10 vehicles scenario.

which, by block-listing the malicious nodes prevents any further drop in performance. The improvement is very close in different scenarios with the different number of vehicles and attackers.

Further performance observation for comparison between the highway and urban environments is available in Figure 4.11 and Figure 4.12. In this case, the highway environment consists of 50 vehicles, and the number of attackers is set to 5. Generally,

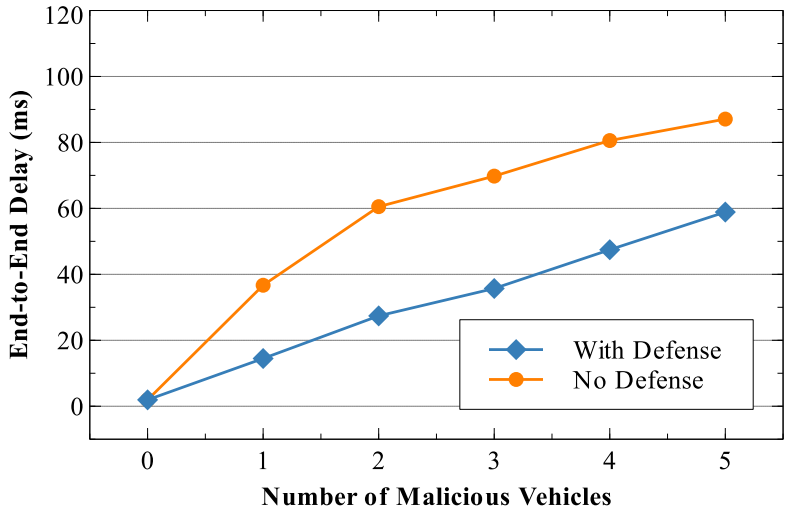

Figure 4.9: TSIDS performance analysis for EED in 30 vehicles scenario.

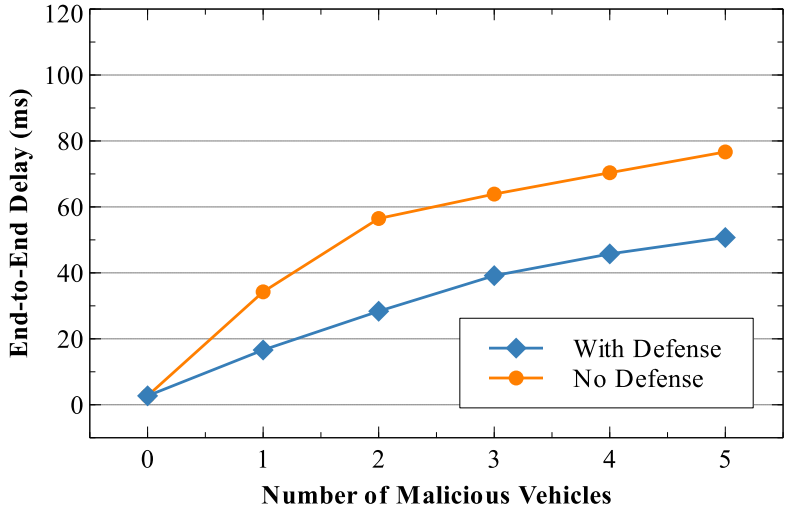

Figure 4.10: TSIDS performance analysis for EED in 50 vehicles scenario.

PDR and EED are negatively affected by the higher vehicle speed and rapid network topology changes on the highway; the adverse effects of malicious activities are also more severe in the highway environment. However, TSIDS is still able to effectively increase the performance of the VANET in the highway scenarios by increasing PDR from 77% to 87% and EED from 139 millisecond (ms) to 60 ms.

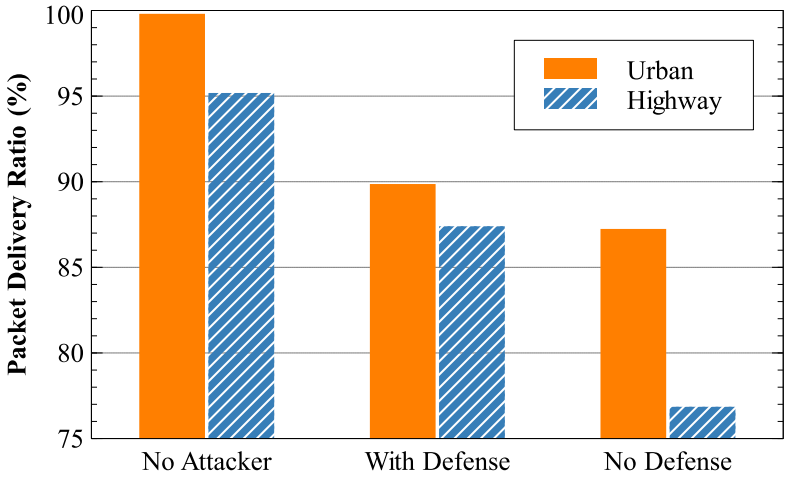

Figure 4.11: TSIDS performance analysis for PDR in highway vs. urban scenarios with 50 vehicles and 5 attackers.

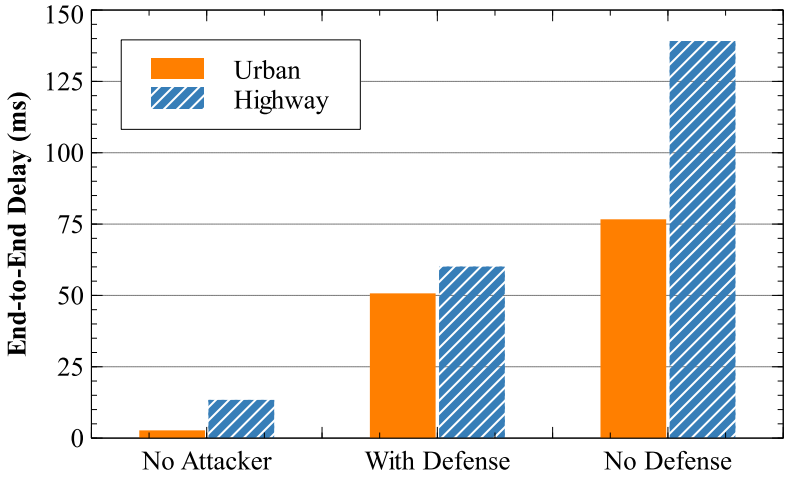

Figure 4.12: TSIDS performance analysis for EED in highway vs. urban scenarios with 50 vehicles and 5 attackers.

Finally, to conclude the performance analysis, we compared TSIDS with other proposed IDSs, namely GKAVIN [101], b-SPEC [102], and LAM [103]. As before, we used PDR and EED as performance parameters in a highway scenario with 76 vehicles and 8 attackers. The results are presented in Figure 4.13 and Figure 4.14. In terms of PDR, TSIDS performs 5.5% better than GKAVIN, about 44% better than LAM, and 229% higher than b-SPEC, which manages to present about 24% PDR in

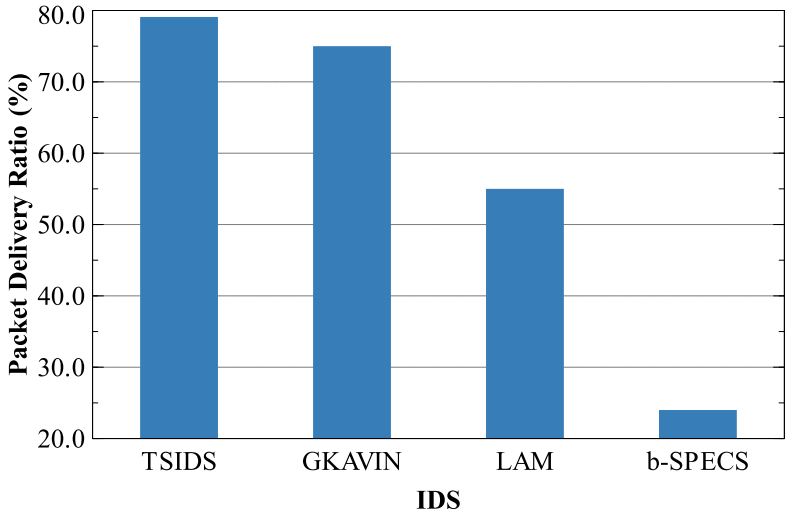

Figure 4.13: PDR performance comparison with 76 vehicles and 8 attackers.

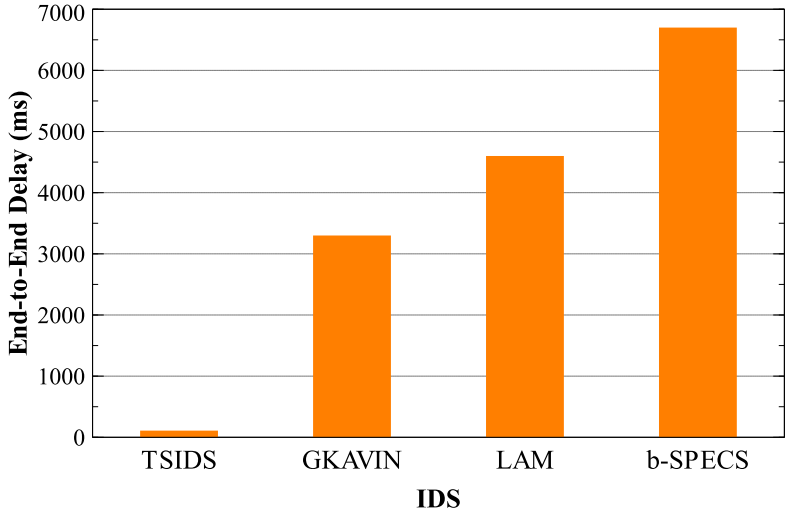

Figure 4.14: EED performance comparison with 76 vehicles and 8 attackers.

general. When comparing the EED performance between the above systems, TSIDS shows on average a 109 ms EED which is 97% to 98% lower than the other methods in comparison.

Chapter 5

CONTEXT-AWARE FEATURE EXTRACTION-BASED CNN MULTI-CLASS IDS

There are situations where solely detecting a presence of intrusion is not enough and we need to act according to the type of the attack. For instance, in a blackhole attack where the malicious user drops all incoming packets, there is no reason to keep this user in the network and it must be eliminated from the routing process. On the other hand, in a wormhole attack that is a passive kind of attack it can still be used in emergency situations where there are no other immediate neighbors to help in forwarding network packets.

In this chapter, we propose an IDS with multi-class classification that outperforms other existing methods for both NIDS and HIDS models. The initial sections of this chapter are the results of our study in [4] and the rest of the chapter is regarding our study regarding the new VANET dataset and the performance of the proposed Context-aware Feature Extraction-based CNN (CAFECNN) in the said environment.

5.1 CAFECNN System Model Design

Several variables can affect the performance of an IDS. Among them are the reference dataset and its size as well as the decision-making algorithm with its various parameters. In this study, we decided to focus on training a detection engine by using CNN as its core. We can define the design process of our proposed IDS in five consequent steps regardless of the chosen dataset: feature extraction, feature selection,

data augmentation, network training, and fine-tuning if applicable. Although, we used the same steps for every IDS dataset, the details within each step are based on the feature data types and whether it is NIDS or HIDS. The block diagram of this process is available in Figure 5.3 after all preprocessing steps are explained.

Feature extraction and feature selection go together and are explained together in the next subsection. However, before delving into the mentioned subjects we need to have a clear understanding of each dataset to be able to decide on the best feature sets for each of them.

5.1.1 Dataset Structure

As stated before, we used four openly available intrusion detection datasets for our research; NSL-KDD, CICIDS2017, ADFA-LD, and ADFA-WD. In this section, we confer more details about the said datasets to provide the context required to understand the proposed feature extraction methods for each dataset.

5.1.1.1 NSL-KDD Dataset

NSL-KDD dataset includes 41 heterogeneous features in each sample. In terms of data type, the features in this dataset can be grouped as categorical such as protocol type, or numerical such as duration. In terms of data gathering methods, the features can be classified into two different classes: packet-based or flow-based. The former is gathered from the packet header whereas the latter is obtained by looking into the payload attributes such as packet size. The categorical features that are represented as text are converted to discrete form for convenience to use in CNN training. The class labels of this dataset are available in both binary and multiclass, but we used the latter and converted class numbers into one-hot vectors. The full training dataset includes 125,973 samples with normal and attack data which we used for training the proposed IDS. For evaluation, there are two test datasets available, test+ which is the full test

dataset, and test-21 which omits samples with difficulty level 21. We used the full test dataset for evaluation. More details are available in Table 5.1 and Table 5.2.

Table 5.1: Original dataset size.

Dataset	Training data	Test data	Total
NSL-KDD	125,973	22,544	148,517
CICIDS2017	-	-	2,830,743
ADFA-LD	1,579	4,372	5,951
ADFA-WD	5,897	1,827	7,724

Table 5.2: Dataset feature information regarding data types.

Dataset	Data Types	Data Sample
NSL-KDD	Categorical / Numerical	0, tcp, ftp_data, SF, 491, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, … , 0.00, 0.05, 0.00
CICIDS2017	Categorical / Numerical	3268, 112740690, 32, 16, 6448, 1152, 403, 0, 201.5, 204.7242047, 72, 72, 72, …, 11.99801571, 380, 343, 16100000, 498804.8203, 16400000, 15400000
ADFA-LD	Categorical	6, 6, 63, 6, 42, 120, 6, 195, 120, 6, 6, 114, 114, 1, 1, 252, 252, 252, 1, 1, 1, 1 …
ADFA-WD	Categorical	… kernel32.dll+0x16d4f, user32.dll+0x8709, user32.dll+0x87eb, user32.dll+0x89a5, user32.dll+0x89e8, kernel32.dll+0x16d4f, user32.dll+0x8709, user32.dll+0x87eb, user32.dll+0x89a5, user32.dll+0x89e8, kernel32.dll+0x16d4f …

5.1.1.2 CICIDS2017 Dataset

The labeled flows in CICIDS2017 are generated using CICFlowmeter [104]. The original feature space includes more than 80 features, however, for ML applications, the authors reduced the feature space to 78 features. Similar to NSL-KDD, the features are packet-based or flow-based. The features in the ML dataset are ready to be used for training and do not need any data conversion. Only the class labels need to be converted to discrete numbers. The normal data is labeled as benign, and the attack

data are each labeled according to their known attack name. Originally, the class labels include 14 attack types, however, to reduce the impact of class imbalance on classification performance we updated the class labels according to the suggestion in [105] which is also adopted in [30]. The old and new attack labels are available in Table 5.4.

The MachineLearningCSV file, which is available in the University of New Brunswick dataset repository includes 2,830,743 samples across 8 different files. We randomly extracted 20% of the dataset for training and testing while ensuring every type of attack is included in both testing and training sections, considering their old and new labels. This data is then split into two subsets, 70% of the total dataset is used for training and the remaining 30% is kept for testing. We chose the number of class samples to be the same as the class distribution in [30] for comparison purposes. In this dataset there are some features with inf and NaN values, one solution is to eliminate these samples, and another solution that we used is to replace these values with the highest valid value in the generated training dataset.

5.1.1.3 ADFA Datasets

ADFA datasets are host-based and only contain categorical data type in each sample. The samples consist of a single trace with variable lengths across the dataset. Every trace is a collection of system calls that are gathered on the host system during normal or under attack situations. ADFA-LD traces include Linux system calls defined by the specific system call number, whilst ADFA-WD traces are recordings of the Windows Dynamic Link Library (DLL) calls defined by the DLL name plus the specific memory access address during the call. Table 5.2 includes a brief comparison among the four dataset structures with an example. In terms of data groups both ADFA datasets include training, validation, and attack data separately. Originally, ADFA datasets are

divided into training, attack, and validation data. We used training and attack data for training the IDS and the validation data is kept for testing. It should be noted that while ADFA-WD includes both malicious and normal data in its validation dataset, ADFA-LD has only normal data in the validation set. Hence, we selected 20% of attack data from each class of attack plus the entire validation dataset of the ADFA-LD to be included in the test data of ADFA-LD [43].

Since we are using 2D convolution layers in our CNN, we need to carefully adjust the features to create image-like 2D vectors for the input of the network. Therefore, it is necessary to decide on the final set of input features as well as the shape of the input as the first step before defining the network parameters. Feature extraction and feature selection methods of this study are explained in the coming sections.

5.1.2 Feature Extraction

The first preprocessing step for preparing the inputs of the network is feature extraction. Existing feature extraction methods such as PCA [106] or Linear Discriminant Analysis (LDA) [107] are generally used to reduce the dimensionality of the original data. However, we used feature selection to reduce the size of the input data. Moreover, feature extraction methods similar to PCA are better at dealing with linear data but fall short when it comes to nonlinear data [108]. We aimed to use a new feature extraction method by focusing on the original data structure of the dataset. NSL-KDD and CICIDS2017 already have 41 and 78 features respectively; however, since the features are heterogeneous it is better to use a form of feature extraction method to unify every input feature. ADFA-LD and ADFA-WD are both system-call traces with variable lengths and cannot be used directly in the first place, hence, both datasets require different feature extraction methods.

Since CNN is generally used for image recognition and the inputs are grayscale or colored images, we decided to define every feature as an individual pixel or set of pixels with values between 0 and 255 in a colored image. The feature extraction process is different for every dataset and is explained separately. The reason is the different contexts in the collected data in every dataset.

5.1.2.1 Feature Extraction for NSL-KDD and CICIDS2017

As mentioned in Section 5.1.1, NSL-KDD and CICIDS2017 have both categorical and numerical features. Based on this information, the feature will be represented as one or more pixels. Generally speaking, the feature values are either nominal or non-negative numerals.

Categorical and ratio features are considered as one pixel and calculated by using (5.1).

$$C_p = \frac{x_c \times 255}{m_c} \tag{5.1}$$

where x_c is the original value of the feature and m_c is the maximum value of the category. Ratios are always real values between 0 and 1. On the other hand, categorical features either have discrete or symbolic values. Symbolic features in the original dataset are converted to discrete values starting from 1. For example, values of the protocol feature are 1, 2, or, 3 corresponding to TCP, ICMP, or, UDP respectively. If the value of the protocol is set as 2 out of 3, then the value of the pixel will be

$$C_p = \frac{2 \times 255}{3} = 170.$$

The numerical values can be continuous with no certain bound. Since the max values vary significantly across different features, we chose to spread their values into several pixels instead of one as mentioned above. We first start by looking at the maximum values of these features individually in the training set. There are two different methods in total. For the first method, the number of pixels and their values for numerical

features are calculated by using (5.2). It should be noted that continuous features have positive values.

$$N_p = \begin{cases} \frac{x_c}{255} & x_c \leq 25{,}500 \\ \frac{x_c}{255^2} & x_c > 25{,}500 \end{cases} \tag{5.2}$$

$$frac(N_p) = (N_p - \lfloor N_p \rfloor) \times 255$$

where N_p is the number of pixels for the numerical values, $\lfloor N_p \rfloor$ is the integer part of N_p representing the number of pixels with the value 255, and $\mathrm{frac}(N_p)$ is the value of the $(\lfloor N_p \rfloor + 1)^{th}$ pixel. The remaining pixels, if any, have their value set to 0. The total number of pixels for each feature is defined by the maximum value of the feature calculated by rounding up N_p to the next integer value. To avoid any feature with a large value taking up more than 100 pixels and slowing down the computation, we decided to divide the features with max values of more than 25,500 by 255^2. This issue is especially important in CICIDS2017 with larger values.

As an example, if a feature has the maximum feature value of 7,479, we can calculate the total number of pixels and their values as follows,

$$N_p = \frac{7{,}479}{255} = 29.33,$$

$$\lfloor N_p \rfloor = 29,$$

$$frac(N_p) = 0.33 \times 255 = 84.$$

This feature will have a total number of 30 pixels where 29 of them take the value 255 and the 30th pixel takes the value 84. Since this value is the maximum value in the training set this feature always takes up to 30 pixels in the final feature space. For another sample, when calculating the pixel values of the same feature that has the value 568, the first two pixels take the value 255, the third pixel takes the value 58 and the remaining 27 pixels will be set to 0. If the same feature has a higher maximum value

in the test data, the number of pixels will not exceed the originally defined number during the training process.

The second method for the numerical values is also based on (5.2) however, it is only applied to the features which indicate data size, such as src_bytes and dst_bytes. Since the value range of these features is noticeably larger than other features and it will be inefficient to solely rely on the previous method, we decided to assign 16 pixels for this feature and divide it into 4 quadruple sections. The first four pixels represent samples of less than 1 Kilobyte in size, the second four pixels are for samples between 1 Kilobyte and 1 Megabyte, the third four pixels are for samples between 1 Megabyte and 1 Gigabyte and the last four pixels are for samples with more than 1 Gigabyte data size. The pixel values are still calculated by (5.2); however, the data is converted to its relevant data size measurement before calculating the pixel values. The benefit of spreading the value of the feature across several pixels compared to one is that by doing so, it will prevent the min-max normalization from reducing the impact of other features with lower values which can potentially have a greater contribution to the classification of a certain intrusion. Also, by using this technique, it is more sensible to use average pooling during the convolution process rather than max pooling.

5.1.2.2 Feature Extraction for ADFA-LD

This dataset is generated by recording the system calls in a fully patched Ubuntu Linux 11.04 [109]. This version of Linux is based on kernel version 2.6 and by default includes 326 system calls. However, for our purpose, we decided to use a feature vector of the size 350 for possible custom system calls. Every entry of this feature vector represents a distinct system call and its value is the ratio of a system call to the size of the whole trace (5.3).

$$SP_i = \frac{|S_i|}{|T|}, \quad i \in \mathbb{N} \text{ and } 1 \leq i \leq 350 \tag{5.3}$$

where SP_i is the i^{th} system call ratio in the final feature vector, S_i is the i^{th} system call, $|S_i|$ is the total number of times where the S_i appears in the current trace, T is the trace, and $|T|$ is the length of the trace based on the number of system calls. For example, a trace with the length 10 which includes 6 system calls with the value 1, and 4 system calls with the value 4, will have its first and fourth entry values set as 0.6 and 0.4 respectively and all others are set to 0. We eliminated any trace with less than 150 system calls for better results. It should be noted that the system call numbers in this dataset start from 1.

5.1.2.3 Feature Extraction for ADFA-WD

The Windows version of the ADFA dataset keeps track of 9 specific DLL calls together with its memory access address. Our custom feature vector for this dataset is as follows. Every unique DLL which is used for generating the traces of this dataset together with its memory access address makes 12 elements of the feature vector with a total of 108 features. Each 12-pair feature vector is defined as follows.

$$D_i = \{d_{i,1}, d_{i,2}, \dots, d_{i,12}\}, \quad i \in \mathbb{N} \text{ and } 1 \leq i \leq 9$$

where D_i is the 12 pair feature vector corresponding to the i^{th} DLL. Elements of D_i are defined in (5.4).

$$d_{i,j} = \begin{cases} |dc_i|, & j = 1 \\ da(dc_i, dc_{j-1}), & 2 \leq j \leq 10 \\ dm_i(ma), & j = 11 \\ df_i, & j = 12 \end{cases} \tag{5.4}$$

where, dc_i is the i^{th} DLL call out of the 9 specified DLL calls in the dataset. $|dc_i|$ is the first element of the feature vector defined by the number of times dc_i is called in the trace. The next 9 elements are defined by *da* which is the number of times dc_i is followed by dc_{j-1} with the emphasis on the order. For example, assume *ntdll.dll* is dc_1

and *kernel32.dll* is dc_2, the first element of this section is *da*(dc_1, dc_1) which is the number of *ntdll.dll* calls that are followed by another *ntdll.dll* call and the next element is *da*(dc_1, dc_2) which is the number of times *ntdll.dll* call is followed by *kernel32.dll* call and so on. The eleventh element is defined by dm_i which is the number of unique dc_i+*memory_address* calls in the trace where *memory_address* is denoted by *ma*. Finally, the last element is defined by df_i that shows the number of times dc_i appears in every 5 consecutive DLL calls. For performance improvement, we decided to eliminate traces with less than 150 DLL calls.

The next step of our preprocessing routine is to eliminate unnecessary or less important features from the samples. This will reduce the computational time while keeping the performance the same or even higher.

5.1.3 Feature Selection

Feature selection is done for reducing the feature space of a dataset and expecting improvement in both training time and performance of the final IDS. Other than the raw performance of the IDS, determining the feature set before feeding the input into the CNN saves the extra processing time needed for the CNN to learn the features that are not contributing as much as the other relevant features to the final classification. Therefore, feature selection reduces the feature space which in return improves the classification time and performance. We used the same feature selection method across all datasets with the only difference being the order; for NSL-KDD and CICIDS2017 feature selection is done before feature extraction for the best result whereas for the ADFA datasets it must be done after the feature extraction process.

In this research, we employed Extremely Randomized Trees (ERT) [Extremely randomized trees 43], also known as Extra Trees, and Select K-Best (SKB) with chi-

squared score function by using the scikit-learn [110] classification tools for feature selection. The mentioned feature selection methods assign a score to each feature which represents its importance in comparison to other features. ERT classifier resulted in better performance and is used mainly for this study. SKB is only used for the NSL-KDD and CICIDS2017 datasets after feature extraction to remove the unnecessary pixels (features) that potentially have a lesser impact on classification and therefore, improve the performance while also reducing the input size.

ERT creates an ensemble model based on the classical top-down decision tree with two major differences. Node splitting is done fully at random, and it uses the whole training data in contrast to a bootstrap replica in other decision-tree-based classifiers. The fully randomized node split makes the final score slightly different each time the classifier is run. However, in our experiment on the ERT feature selection on the NSL-KDD dataset we discovered that if we choose to eliminate a minimum of 11 features with the lowest feature scores, we almost always end up with the same features being dropped with less than 1% chance to get a different result. We used the same feature reduction ratio of about 27% which worked best on NSL-KDD, for the CICIDS2017 to reduce the original feature space from 78 features to 57. The ERT feature weights in this dataset are not as consistent in comparison to the NSL-KDD dataset, however, the selected features proved to be more relevant in the training time. The eliminated features with collectively lowest feature scores for the NSL-KDD and CICIDS2017 are available in Table 5.3 in no particular order.

The number of eliminated features is based on the feature importance score as mentioned above with consideration of the final shape of the image. The ideal image

Table 5.3: Eliminated features from the NSL-KDD and CICIDS2017 datasets after ERT features selection.

Dataset	Eliminated features
NSL-KDD	*land*, *urgent*, *num_failed_logins*, *root_shell*, *su_attemted*, *num_root*, *num_file_creation*, *num_shells*, *num_access_files*, *num_outbound_cmds*, and *is_host_login*
CICIDS2017	*Bwd PSH Flags, Bwd Avg Bulk Rate, Bwd URG Flags, ECE Flag Count, Fwd Avg Bytes/Bulk, Fwd Avg Packets/Bulk, RST Flag Count, Bwd Avg Packets/Bulk, Fwd Avg Bulk Rate, Bwd Avg Bytes/Bulk, Fwd URG Flags, CWE Flag Count, Active Std, Active Max, Active Min, Subflow Bwd Bytes, Total Length of Fwd Packets, Idle Std, Flow Bytes/sO1, Bwd IAT Min, Bwd Packets/s*

size is obtained through trial and error after each network training. For example, for ADFA-WD with 108 initial features and the final image size of 10 by 10 pixels, we need to eliminate 8 lowest-ranked features. As for NSL-KDD and CICIDS2017, we initially dropped 27% of the original features before starting the feature extraction process. Afterward, we reduced the resolution of the final samples by running the SKB feature selection on the entire pixels. This step helped to reduce the pixel count by 48% which in return improved the classification time and the accuracy of the IDS at the same time.

5.1.4 Data Augmentation

Data augmentation in ML applications is referred to as a method that is meant to increase the training data pool by diversifying the existing data in different ways. Rotation, mirroring, cropping, and padding are some common methods of data augmentation for image datasets. It will give the network a chance to learn the class of the same object from different angles. However, in this study, we work on network intrusion datasets that are modified to represent an image-like object. Hence, we do not expect it to react the same way as image recognition datasets. Therefore, instead of creating new data through augmentation, we extended the existing data to add two

more layers and create an RGB image from a single sample. The performance improvement of the RGB in comparison to greyscale encoding [111] is investigated in [112], which shows a notable improvement in the accuracy of the trained model. In our study, we use a different method for creating RGB channels, which is based on the feature extraction step. The major difference between our method with the one-hot or binary encoding is that we also took the context of the feature into the consideration. For instance, as mentioned below, we used different encoding for the sent and received bytes based on the packet payload size. The data augmentation process is done in 4 steps. After using this method, we observed a notable performance increase in comparison to a single-channel or greyscale image.

The first step is to change the shape of the original data to a 2D vector. Afterward, for the second channel, we mirrored the original 2D data horizontally. The third channel is generated differently for NSL-KDD/CICIDS2017 and ADFA datasets. For the former datasets, pixel values are the same as the first layer but the value of pixels that represent continuous and percentile features are subtracted from 255. For the ADFA datasets, it is simply generated by subtracting the value of each pixel from the maximum value of the current sample.

After all of the channels are generated, we added the standard deviation of each row and column to the end of its corresponding row and column starting from the rows. After each channel is generated, all three channels are combined and reshaped into an a by b by 3 vector which is suitable for convolution layers. Figure 5.1 is an arbitrary data augmentation process for an input with 9 features. Figure 5.2 is an example of the final result for every dataset comparing normal and attack data. This should be noted that the final results can be generated to have values between 0 and 1 during the

preprocessing to eliminate the need for min-max normalization or it needs to be normalized before the start of the network training. Figure 5.3 illustrates the block diagram of the preprocessing steps. Table 5.4 shows the final class distribution after the preprocessing steps are complete.

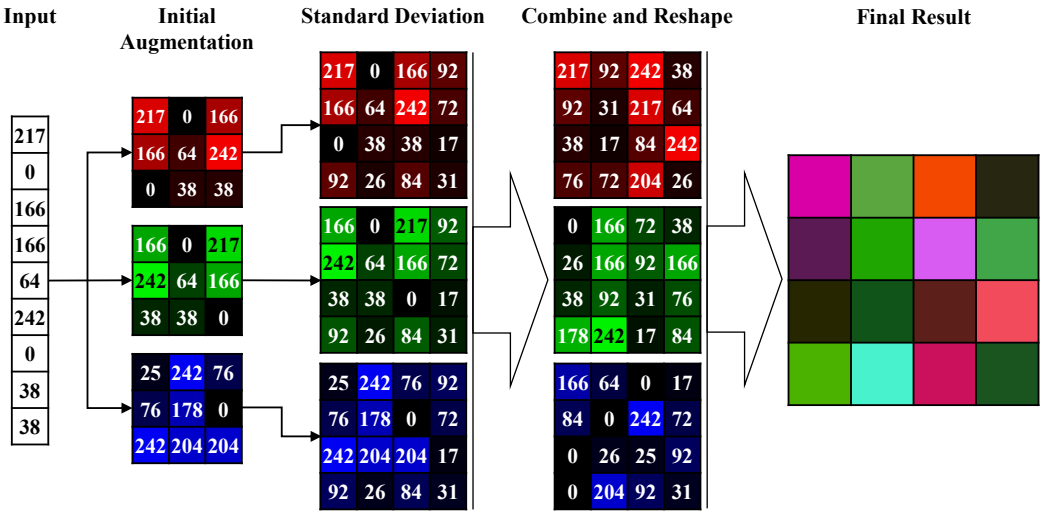

Figure 5.1: An example of data augmentation process on an arbitrary sample with 9 features.

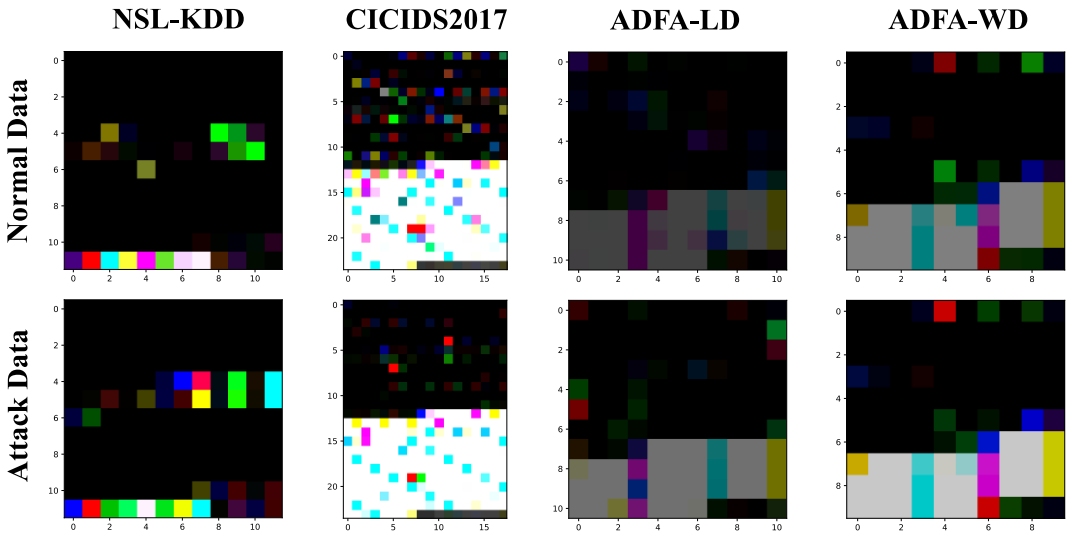

Figure 5.2: Visual example of the final preprocessing output for NSL-KDD, CICIDS2017, ADFA-LD, and ADFA-WD datasets.

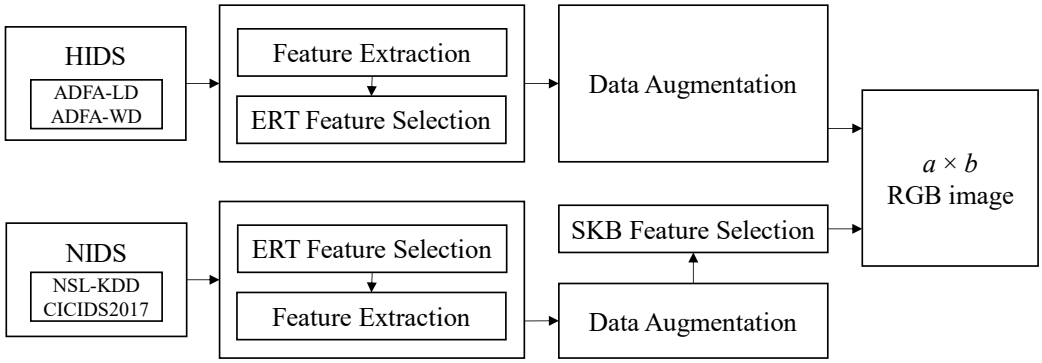

Figure 5.3: Preprocessing block diagram for HIDS and NIDS datasets.

Table 5.4: Final dataset class distribution after the complete preprocessing steps.

Dataset	Classes		Training data		Test data		Total
			Individual	Total	Individual	Total	
NSL-KDD	Normal		67,343	125,973	9711	22,544	148,517
	DoS		45,927		7,456		
	U2R		52		202		
	R2L		995		2,754		
	Probe		11,656		2,421		
CICIDS2017	Normal	Benign	318,087	396,304	136,219	169,845	566,149
	Botnet ARES	Bot	265		102		
	Brute Force	FTP-Patator	1,002		607		
		SSH-Patator	902		206		
	DoS/DDoS	DDoS	3,700		4,300		
		DoS GoldenEye	5,300		1,700		
		DoS Hulk	39,909		14,206		
		DoS Slowhttptest	2,161		1,159		
		DoS slowloris	2,350		1,650		
		Heartbleed	7		3		
	Infiltration	Infiltration	5		1		
	PortScan	PortScan	22,324		9,558		
	Web Attack	Web Attack - Brute Force	190		94		
		Web Attack - Sql Injection	15		5		
		Web Attack - XSS	87		35		
ADFA-LD	Normal		563	985	3,398	3,502	4,487
	Adduser		58		19		
	Hydra FTP		84		19		
	Hydra SSH		78		21		
	Java Meterpreter		85		20		
	Meterpreter		49		9		
	Web Shell		68		16		
ADFA-WD	Normal		4	4,212	12	1,592	5,804
	CesarFTP		323		10		
	WebDAV		333		101		
	Icecast		256		13		
	Tomcat		288		40		
	OS SMB		234		14		
	OS Print Spool		373		109		
	PMWiki		301		11		
	Wireless Karma		407		264		
	PDF		333		136		
	Backdoored Executable		400		119		
	Browser Attack		405		181		
	Infectious Media		555		582		

5.2 CAFECNN IDS Design and Training

At this point where the preprocessing of the datasets is complete, we need to find the most suitable CNN model for each dataset individually. A typical CNN consists of three main layers. Convolution layer which takes the input vector with the most important parameters of this layer being input shape, filter and padding size, and activation function. A pooling or downsampling layer is commonly either maximum or average pooling. Another important parameter of the pooling layer is the shape of the pooling layer. Finally, there is commonly a fully connected or dense layer at the end of the network. The dense layer consists of fully connected neurons to optimize the outputs of the previous layers. This layer usually ends with an activation function for predicting the class of the initial input. The mentioned parameters of each layer are usually referred to as hyperparameters which need to be fine-tuned differently for any individual dataset.

For this study, we used the Keras Deep Learning library [113] with TensorFlow r1.14 Application Programming Interface (API) [114] backend for training our CNN-based IDS. The final performance of any CNN is affected by its structure and hyperparameters of the network as well as the training dataset complexity. Hence, the preprocessing section is necessary to prepare the datasets in a way that can be generalized better by CNN, and the aim of the network design which is explained below is to achieve the best performance based on the available dataset.

Convolution layers in our proposed network have identical hyperparameters with Rectified Linear Unit (ReLU) as the activation function for every dataset; however, the pooling and dense layers required more tuning for better results. Average pooling

proved to be more suitable for NSL-KDD and CICIDS2017 whereas for ADFA datasets maximum pooling resulted in better performance.

After the last pooling layer, the network output is flattened into a single dimension vector for the dense layers. The number of dense layers and their neurons in the initial layers vary from dataset to dataset and the number of neurons in the final layer is according to the number of classes in the dataset. The output class of the network is determined using the SoftMax function.

One of the most severe drawbacks of deep learning applications is overfitting. This is an issue arising when the network is heavily biased towards the provided training dataset with high accuracy but is unable to properly generalize objects outside of the training data. Data augmentation is one way to reduce overfitting. The use of regularization methods such as Least Absolute Shrinkage Selection Operator (LASSO) [115], Ridge [116], and Dropout [117] can also solve overfitting issues [118]. In this study, we used dropout after the first dense layer. This helps the network to generalize the training dataset by ignoring the weights of random neurons from the previous layer. However, the amount of ignored neurons must be fine-tuned for each dataset since a low dropout rate is not highly effective whereas a high ratio will reduce the performance of the network. More detailed information about the network layers is available in Table 5.5.

With the above hyperparameters defined, the network is ready to be trained. The training parameters are defined as follows. The loss function is Categorical Cross-Entropy which is suitable for multiclass datasets. The optimizer is set to Adam [119] with the default hyperparameters in Keras. We also used balanced class weights to

address the issue of imbalanced class representation in each dataset. Note that the class imbalance in ADFA datasets occurs and became more pronounced after excluding traces with less than 150 in length.

During the training process, we monitored the loss, mean absolute error, and accuracy of the network. The dropout mechanism adds more randomness to the training process in a way that each time we have different sets of ignored neurons. Therefore, we tested the trained network after each epoch against the test dataset to keep the best result until the predefined maximum epoch number is reached. In this method, if the trained CNN manages to generalize the training set with high accuracy with low overfitting, we can capture the best model without waiting for the maximum epoch and the end of the training. No validation data is used during the training, and test data evaluation does not affect the weight updates. It should be noted that using evaluation data can be useful in many cases and improves the performance of the trained model. However, in our study when we used 10% and 20% of the NSL-KDD training dataset as the evaluation set, we did not see testing accuracy of above 80.1% and 80.4% respectively, whilst our best result with no evaluation set had 81.9% accuracy.

5.2.1 Model Fine-Tuning

Model fine-tuning is a technique that is used for improving the performance of a trained network or in transfer learning without the need to train the network from scratch. Fine-tuning requires a pre-trained network model as a basis, and it is a natural part of the transfer learning [120]. In transfer learning, it allows using the previously extracted features or adjusted weights for a set of input data to be used for a different set of outputs. In other words, it allows transferring the acquired knowledge from one set to another. However, in our study, we are using the same set for fine-tuning the pre-trained model. The aim is to increase the performance of the current model using

the same set by adjusting new weights at the end of the model. For instance, when we fine-tune our proposed IDS for the NSL-KDD dataset, the training accuracy improvement is only about 0.01% (from 99.90% to 99.91%), whereas, on the test dataset, the improvement in accuracy is about 1.54%. Therefore, the new model can generalize the unseen data better than before. More detail about the final result is available in Section 5.3.

In this process, normally, the last layer of the network that corresponds to the output class is removed and a new layer or layers will be added for training. The weights from the original network, especially in the top layers, are fully or partially locked to preserve their values. Fine-tuning is not guaranteed to be effective, hence, researchers in various studies such as [121] investigate the possibility of a use case for fine-tuning. In this study, we removed the last layer from the models and kept all previously trained neurons untrainable, meaning that the weights of those neurons are not going to be updated. After inserting new dense layers into the model, we trained the model with the new layers and observed that the performance only improved for three out of the four datasets. Details of the new layers are available in Table 5.5. As it can be observed, the ADFA-WD dataset has no fine-tuning layers because it could not improve the performance. Figure 5.4 illustrates the workflow of the proposed IDS. More discussion on the results is available in Section 5.3.

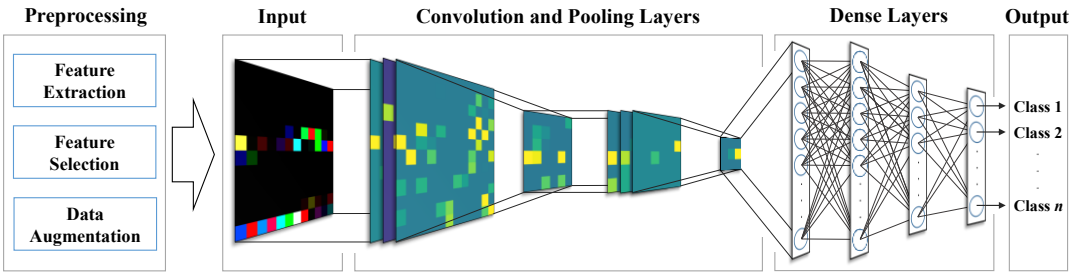

Figure 5.4: Workflow of the proposed CAFECNN IDS.

Table 5.5: CNN structure for each dataset with and without fine-tuning.

Dataset	Convolution (Conv)* and Pooling Layers	Dense Layers**	Fine-Tuning**
NSL-KDD	3× 2D Conv Layers (64, 3×3, same, ReLU) 1× Average Pooling (pooling size: 2×2) 3× 2D Conv Layers (64, 3×3, same, ReLU) 1× Average Pooling (pooling size: 2×2)	Dense (256, Linear) Dropout (30%) Dense (256, Linear) Dense (5, Linear) Dense (5, SoftMax)	Dense (20, Linear) Dense (5, SoftMax)
CICIDS2017	3× 2D Conv Layers (64, 3×3, same, ReLU) 1× Average Pooling (pooling size: 2×2) 3× 2D Conv Layers (64, 3×3, same, ReLU) 1× Average Pooling (pooling size: 2×2)	Dense (1,536, Linear) Dropout (30%) Dense (512, Linear) Dense (60, Linear) Dense (7, SoftMax)	Dense (60, Linear) Dropout (30%) Dense (20, Linear) Dense (7, SoftMax)
ADFA-LD	3× 2D Conv Layers (64, 3×3, same, ReLU) 1× Max Pooling (pooling size: 2×2) 3× 2D Conv Layers (64, 3×3, same, ReLU) 1× Max Pooling (pooling size: 2×2)	Dense (576, Linear) Dropout (30%) Dense (256, Linear) Dense (7, Linear) Dense (7, SoftMax)	Dense (1,024, Linear) Dropout (30%) Dense (256, Linear) Dense (7, SoftMax)
ADFA-WD	3× 2D Conv Layers (64, 3×3, same, ReLU) 1× Max Pooling (pooling size: 2×2) 3× 2D Conv Layers (64, 3×3, same, ReLU) 1× Max Pooling (pooling size: 2×2)	Dense (256, Linear) Dropout (20%) Dense (128, Linear) Dense (13, Linear) Dense (13, SoftMax)	-

* 2D Conv Layers (filter size, strides, padding, activation function)
** Dense (number of neurons, activation function), Dropout (ratio of ignored neurons)

5.3 Performance Analysis of CAFECNN

In this section, we present and discuss the result of the benchmark dataset evaluation by illustrating the performance of the initial model as well as the fine-tuned model using the test data. Furthermore, we compare our results with previously conducted research and their proposed methods. It should be noted that we did not include results from studies that only include the performance of the training data or combined training and test data classification.

For the performance evaluation of the CAFECNN we chose the evaluation metrics to be Accuracy of the IDS as shown in (5.5), Precision, Recall, and F-Score. Since we used multiclass datasets with imbalanced class samples in the test dataset, we used the weighted average of the above metrics as shown in (5.6)-(5.8). The calculations are

based on four classification parameters as follows. True Positive (TP) is the number of correctly classified samples; FP is the number of classes that do not belong to the specific class but are classified as such; True Negative (TN) which is equivalent to TP and is the number of correctly rejected samples that do not belong to the intended class; False Negative (FN) which indicates how many samples are wrongly rejected whereas it should be classified as a specific class and is directly related to FP.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{5.5}$$

$$Weighted\text{-}Precision\ (WPR) = \frac{1}{\sum_{i=1}^{n} q_i} \sum_{i=1}^{n} q_i \frac{TP_i}{TP_i+FP_i} \tag{5.6}$$

$$Weighted\text{-}Recall\ (WRC) = \frac{1}{\sum_{i=1}^{n} q_i} \sum_{i=1}^{n} q_i \frac{TP_i}{TP_i+FN_i} \tag{5.7}$$

$$Weighted\text{-}F\text{-}Score\ (WFS) = \frac{1}{\sum_{i=1}^{n} q_i} \sum_{i=1}^{n} q_i \frac{2\ Precision_i\ Recall_i}{Precision_i+Recall_i} \tag{5.8}$$

where n is the number of classes, q_i is the number of samples in class i.

Before we start analyzing the performance of the proposed IDS, we are going to briefly discuss the computational time of CAFECNN. Regarding the preprocessing steps for every new sample, since the feature selection step only consists of ignoring the eliminated features, we only need to compensate for the feature extraction and data augmentation calculations which require basic operations explained in Section 5.1. Regarding the detection module, the proposed system is based on an RGB encoding instead of greyscale, which is more computationally extensive, however, it provides higher accuracy. In our study, we used a notebook PC with an Intel Core i7 7700HK processor and NVidia GeForce GTX 1060 mobile.

We evaluated the performance of our proposed preprocessing steps in comparison to a model with No Preprocessing (NP), as well as other models with only one

preprocessing step such as only Feature Extraction (FE), Feature Selection (FS), Data Augmentation (DA) or a combination of them up to the complete set. The testing is done on the KDDTest+ dataset. We measured the classification time as well as the accuracy of the system to find the middle ground for the best performance and classification time combination. Since the input size of the NSL-KDD dataset is the largest (12 by 12 pixels), we conducted the test on the NSL-KDD dataset as the reference.

Figure 5.5 displays the effect of preprocessing steps on the Accuracy of the IDS. The performance of the IDS with only one preprocessing method is roughly the same for DA and FS methods which for both is around 77% Accuracy and is almost the same as the performance of the IDS with no preprocessing step. We can also notice that our unconventional DA method is not effective when it is used on its own. Whereas its effectiveness, when combined with FS, is noticeably positive which is around 79%. This indicates that terminating the unnecessary features from the samples improves the effect of the standard deviation in our DA method.

On the other hand, the proposed FE method shows a significant improvement on its own with Accuracy of around 80%. As it can be seen from the graph, our context-aware FE also improves the performance of the IDS to around 79% Accuracy when combined with only one of the FS or DA methods compared to the former methods on their own. However, the best performance is achieved when we used a combination of all preprocessing methods by reaching up to 81.90% Accuracy.

Concerning the classification or detection time of the IDS, Figure 5.6 shows the classification time of the IDS per sample in microseconds (μs) with different

preprocessing steps. The classification time is the average of running the test 10 times on the entire KDDTest+ set. One can observe from the graph that the FS method which we employed reduces the classification time by around 4% to the low of 32.61 µs, yet the Accuracy is not comparable to the best result. Our proposed FE method that we examined its better performance in terms of Accuracy shows a significant increase in the classification time as high as 57.05 µs when used on its own, which is 68% more than the baseline model. By paying attention to the combination of the proposed FE with DA or FS we can see that while the DA increases the classification time by 12%, the FS method effectively decreases the required time for classification to around 38 µs or 33% lower than the FE method on its own. Even though the combination of FE-FS has a relatively low classification time, it does not provide a competitive performance in comparison to the full model. On the other hand, the model that uses the full set of the preprocessing method has a reasonable classification time of 44 µs which is 34% higher than the baseline model in comparison to the FE model which shows a 68% increase in classification time. Additionally, the full preprocessing model shows the highest Accuracy of 81.90% compared to 77.16% Accuracy on the baseline model.

In conclusion, we achieved the best performance when we used a full set of our proposed preprocessing steps to prepare the datasets while keeping the computational requirement as close to the baseline model as possible.

Considering the detection performance of the CAFECNN, as mentioned before, we only present the results of the test data evaluation since the training accuracy can be highly affected by overfitting. For instance, the training result for NSL-KDD shows 99.91% Accuracy for the current model, which can be even higher without handling

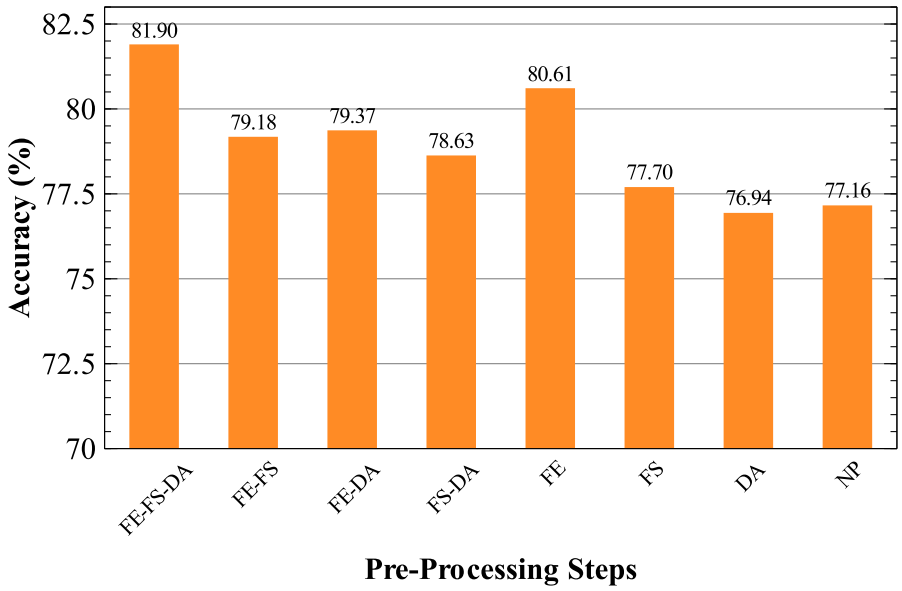

Figure 5.5: Accuracy of the CAFECNN on KDDTest+ with different preprocessing steps. Including NP, FE, FS, and DA preprocessing steps separately, or together.

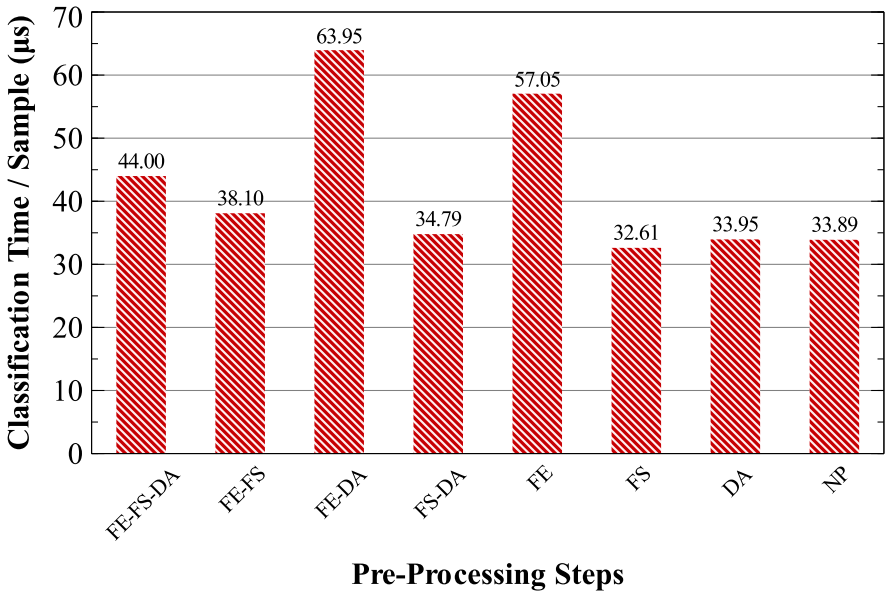

Figure 5.6: Classification time per sample for the CAFECNN on KDDTest+ with different preprocessing steps. Including NP, FE, FS, and DA preprocessing steps separately, or together.

the overfitting issue. For NSL-KDD we used the KDDTest+ set. The reason is that there are new methods of intrusions included in the KDDTest+ that are not available in the training set. Therefore, using the unseen data together with the unseen intrusion

methods provides a more robust performance test in comparison to combining the entire training and testing dataset and then slicing them into training, validation, and test data randomly. The random slicing process may include the new intrusion methods to the training data which can increase the metrics in the final result significantly without implying the performance of the network on potential zero-day attacks. For the ADFA-LD we used 20% of the original training data as the test data because the original test data only includes normal data. For ADFA-WD we used the original test data for evaluation.

Table 5.6 shows the accuracy of the proposed CAFECNN IDS after the initial training plus fine-tuned models for each dataset. Accuracy of the CICIDS is 99.22% which is higher than the other three, followed by ADFA-LD IDS with 95.12% Accuracy, next is NSL-KDD with 81.89% and the last result is for ADFA-WD with 77.01% accuracy. ADFA datasets are considered to have higher complexity in comparison to KDD-99 or NSL-KDD [9], particularly the similarity of normal and attack data in ADFA-WD [122] causes its accuracy to be considerably lower than the other two datasets. By looking at the images generated by the preprocessing methods we can also visually observe that the difference between normal and attack data in ADFA-WD is minimal in comparison to the other datasets. Also, fine-tuning could not improve the performance of the ADFA-WD dataset. Fine-tuning result for the ADFA-LD provides marginal improvement with an addition of 0.22% performance compared to the original Accuracy while it required two extra dense layers with dropout to achieve this performance. Similarly, fine-tuning improved the Accuracy of CICIDS2017 by a small margin. However, for NSL-KDD, one extra dense layer with 20 neurons improved the Accuracy of the IDS by an additional 1.54% Accuracy. This implies that fine-tuning can be effective but not guaranteed and it might not be cost-effective based on the

available computational resources. For instance, in our study, we used GPU as our main computational resource and by looking at the classification time in Table 5.6, we can see that the additional dense layers do not impact the classification time in a significant way.

Table 5.6: Accuracy and classification time of the proposed IDSs before and after fine-tuning.

Dataset	Base model		Fine-tuned model	
	Accuracy (%)	Classification Time per Sample	Accuracy (%)	Classification Time per Sample
NSL-KDD	81.89	44.00 µs	83.43	47.71 µs
CICIDS2017	99.22	87.58 µs	99.29	87.68 µs
ADFA-LD	95.12	47.69 µs	95.34	49.72 µs
ADFA-WD	77.01	43.63 µs	-	-

The final performance result of our proposed IDS in terms of Accuracy, WPR, WRC, and WFS is depicted in Figure 5.7. Overall, the performance of the CICIDS2017 with 99.29% is the highest, and ADFA-LD IDS with 95.12% Accuracy comes second. Even though the complexity and number of classes in CICIDS2017 and ADFA-LD are higher than the NSL-KDD, the final Accuracy of this dataset is 83.43% which comes in third place. Finally, ADFA-WD with 77.01% Accuracy is the lowest. Other evaluation metrics show the same trend in general; however, by looking more closely at WPR, it is apparent that ADFA-LD and NSL-KDD have a higher WPR rate compared to other metrics. This shows that the respective IDSs have a relatively lower FP rate compared to ADFA-WD. WRC record of every IDS is very close to Accuracy and lower than the WPR except for ADFA-WD. In terms of WFS, only ADFA-LD shows a better performance in comparison to Accuracy which means that the F-Score rates and therefore Precision and Recall rates across all of its classes are closer to each

other and more balanced compared to NSL-KDD and ADFA-WD. CICIDS2017 shows the most stable result with very close WPR, WRC, and WFS values. This shows that the classifier can generalize all classes relatively better with low FP and FN rates.

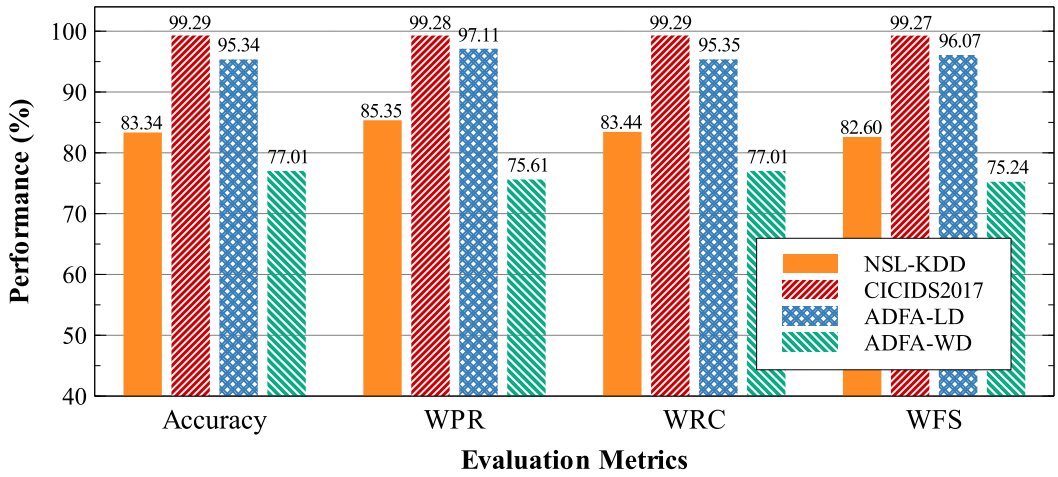

Figure 5.7: Performance of the proposed CAFECNN IDS for NSL-KDD, ADFA-LD, and ADFA-WD datasets in terms of Accuracy, WPR, WRC, and WFS.

The comparison of the proposed methods is done separately for every dataset. For NSL-KDD we compared our results in terms of Accuracy with other multiclass RNN-based [25], [26] and CNN-based [28], [29] models plus other common classification models such as NB, C4.5 Decision Tree (J48), NB Decision Tree (NB Tree), Multi-Layer Perceptron (MLP), SVM, Random Forest, and Random Tree [29]. The result of the comparison in Figure 5.8 illustrates the effectiveness of the CAFECNN IDS which is followed by SCAD-RNN and multi-CNN in second and third place. Comparing our results with two other CNN-based IDSs shows that our preprocessing method plus the CNN structure that we used improves the Accuracy of the IDS in multiclass classification.

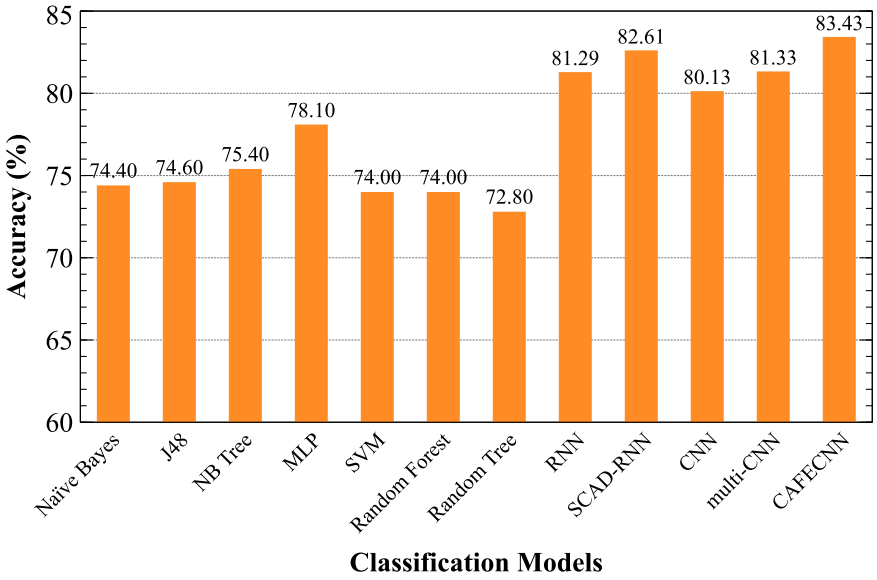

Figure 5.8: Performance comparison of the proposed CAFECNN IDS with the other models for the NSL-KDD Test+ dataset regarding the multiclass classification Accuracy. Including results from RNN [25], SCAD-RNN [26], CNN [28], multi-CNN [29], and other methods that are mentioned in [29].

The Accuracy of the CAFECNN model on the CICIDS2017 dataset is above 99% which is remarkably high compared to other datasets. Our comparison results in Figure 5.9 depict the highest performing multiclass IDSs in [30], [32]–[35] which shows that our proposed method is either higher or on par with the other proposed IDSs. Overall, the IG models proposed by [30] performed very similarly with the same number of features and the same number of training and test samples. Kurniabudi *et al.* evaluated their IG method on 5 different ML models, which we defined in Figure 5.9 by the name of the method (IG), the name of the classifier, and the number of selected features. We chose 57 feature evaluation results which is the same feature number as our experiment. We eliminated the lowest-performing IG model which is NB with less than 50% Accuracy with 57 features. The remaining models are Random Forest (IG-RF-57), Bayes Network (IG-BN-57), Random Tree (IG-RT-57), and J48 (IG-J48-57). Among the mentioned models IG-J48-57 is the highest performing with 99.87% Accuracy and IG-BN-57 has the lowest performance with 94.16% Accuracy.

The rest of the presented models do not follow the same dataset size and/or labels. For hybrid models, CNN+LSTM11 [34] performs better than the other methods with 99.81% Accuracy followed by Cu(LSTM-CNN) [32] with 98.60% Accuracy and DL-IDS [33] with 98.67% Accuracy. The slight differences in the performance of the mentioned models can be due to random data selection from the originally large number of samples. In other words, there is a randomness in the difficulty of the final dataset because of the randomized sampling.

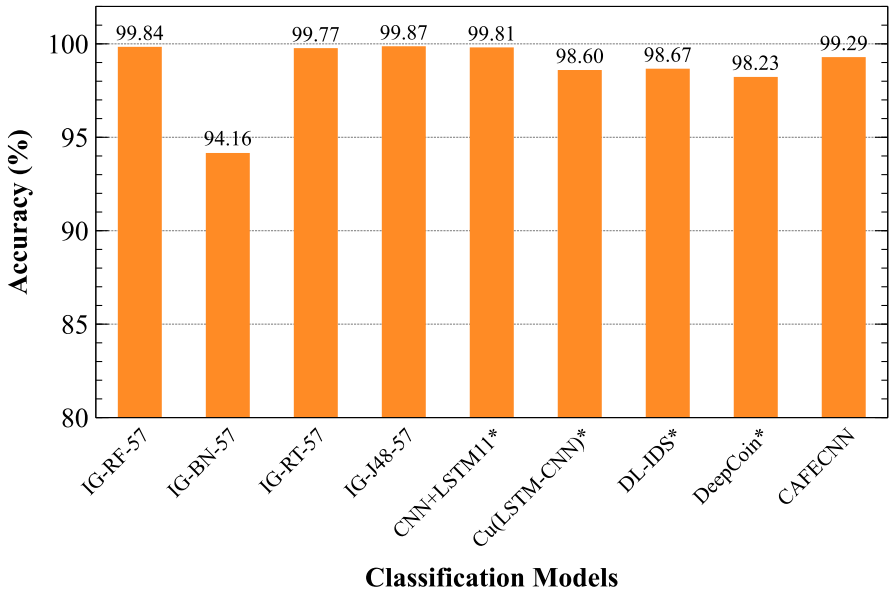

Figure 5.9: Performance comparison of the proposed CAFECNN IDS with the other models for the CICIDS2017 dataset regarding the multiclass classification Accuracy. Including results from IG models [30], Cu(LSTM-CNN)* [32], DL-IDS* [33], CNN+LSTM11* [34], and DeepCoin* [35].
(*these models have different numbers of classes and/or dataset sizes for training and testing)

Next, we compared our proposed CNN model for ADFA-LD and ADFA-WD with the results of MLP-based IDS in [43]. In their method, they included a portion of all attack data in the test dataset for ADFA-LD similar to what we did in this study. Another reason that we chose their work for comparison is that the authors considered both

ADFA-LD and ADFA-WD for the evaluation of their IDS. It should be noted that we did not take the processing time of the two methods into account.

The graph in Figure 5.10 shows the performance comparison between the CAFECNN and MLP method with the top 80 features for 1-gram and 2-gram with 6 neurons in the hidden layer. Overall improvement can be seen in all evaluation metrics. The improvement in Accuracy and WRC are marginal. However, in WPR which is 97.10%, it is more visible and shows our proposed model has a considerably lower FP rate. WFS, which is the harmonic mean of Precision and Recall shows an improvement of around 1%.

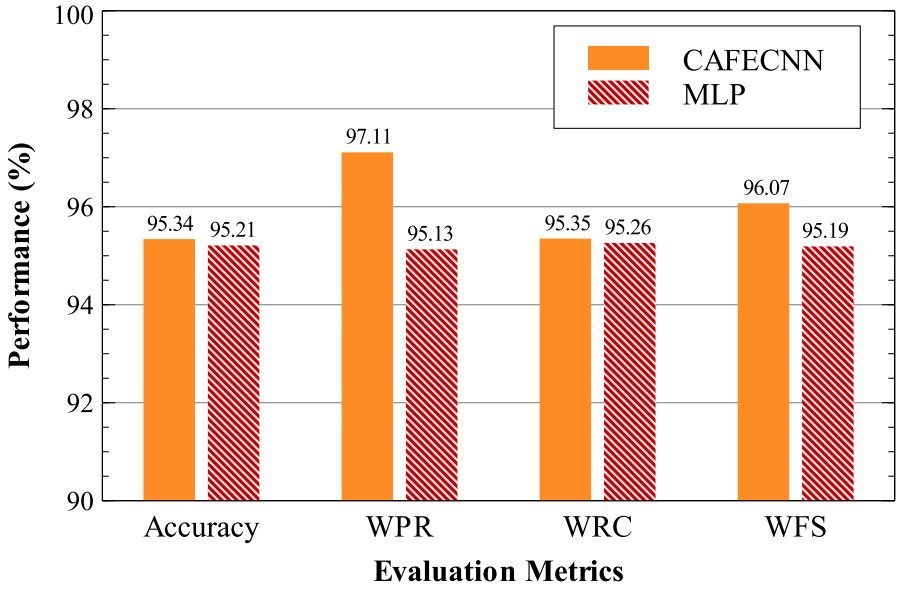

Figure 5.10: Performance comparison of the proposed CAFECNN IDS with the MLP-based model [43] regarding the ADFA-LD dataset in terms of Accuracy, WPR, WRC, and WFS.

The performance benchmark of the ADFA-WD dataset in Figure 5.11 shows that when compared to the MLP model with the top 80 features for 1-gram and 2-gram with 6 neurons in the hidden layer. The CAFECNN depicts noticeably better performance.

Overall, the performance of both models is below 80% which is due to the complexity of the dataset and considerably more types of attacks in this dataset compared to other datasets. In this benchmark, both models have higher WRC compared to WPR which indicates higher detection rates compared to the false alarm.

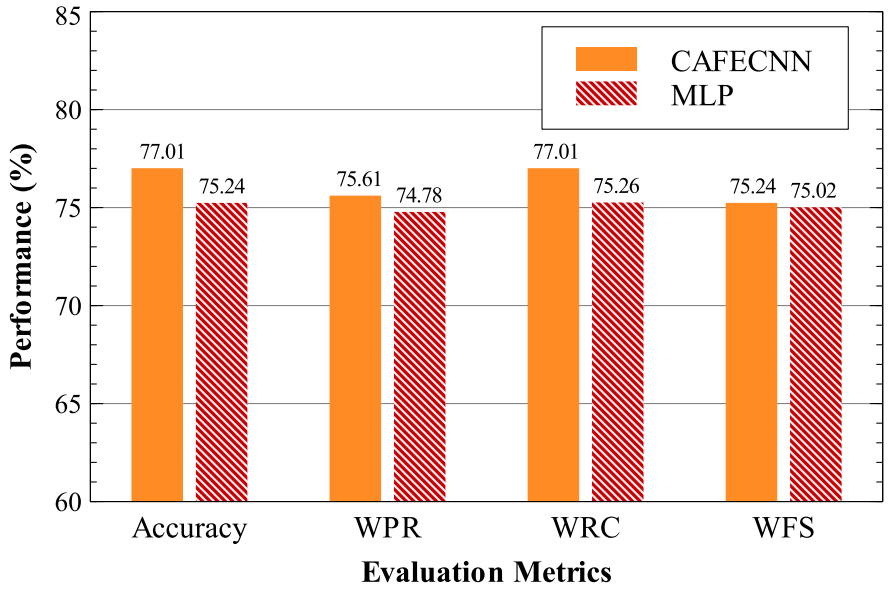

Figure 5.11: Performance comparison of the proposed CAFECNN IDS with the MLP-based model [43] regarding the ADFA-WD dataset in terms of Accuracy, WPR, WRC, and WFS.

In general, our proposed method can provide a solid classification method for the aforementioned datasets for detecting various types of attacks in both host-based and network-based scenarios.

5.4 New VANET IDS Dataset

Up to this point, we have evaluated our proposed feature extraction coupled with a CNN classifier using popular benchmarks. However, since the NIDS benchmarks are generated using infrastructure communication networks and this research aims to address security attacks in VANET, we need to use a dataset that is representative of the said environment. As mentioned before, Rahal *et al.* have generated a realistic

VANET dataset using two vehicles. The dataset contains four types of DoS attacks as well as the normal data. We evaluated this dataset using the CAFECNN method and got 99.992% Accuracy in the highway scenario which is almost the same as the authors' result from RFC in the best set by 99.998% Accuracy. We did not investigate urban and rural scenarios since the performance of the RFC for multi-class classification in the original paper is not varying significantly, therefore we concluded that the performance of the CAFECNN method will also be around the same. This dataset, while being a great contribution is limited in terms of the number of vehicles and therefore, scalability. Additionally, we want to include passive attack data for evaluating our proposed multi-class IDS. The challenge in detecting passive attacks is that there are no obvious effects on the network performance. Hence we generated a new VANET IDS dataset using a computer simulation that includes both active and passive attacks as well as the normal data. We also generated additional data with different network traffic rates to be included in the test data to further evaluate the generalizability of the classifiers.

5.4.1 Simulation Setup

We conducted the simulations by using SUMO and ns-3 to generate both vehicular mobility and network communication as accurately as possible. The SUMO map is a 2 by 3 grid Manhattan scenario with 50 vehicles in 5 different types. The vehicles start from a random junction and continue driving on the designated map following the traffic rules until the simulation time is over. The details of the SUMO simulation are available in Table 5.7.

After the vehicular mobility trace is generated, we convert it to the ns-2 mobility trace that is compatible with ns-3. We initiated all the vehicles and RSUs using VANET-specific parameters that are mentioned in Table 5.7. The RSUs are strategically placed

Table 5.7: Simulation Parameters (ns-3, SUMO).

Parameters	**Values**	
Simulation Area	600 × 400 m^2	
Simulation Time	2000 s	
Number of Vehicles	50	
Number of RSUs	2	
Vehicle placement / Movement	Random Starting Points with Random Junction Turns	
Vehicle Speed	0-50 km/h	
Vehicle Type / Count	Passenger	35
	Bus	5
	Delivery	5
	Truck	3
	Trailer	2
Mac Protocol	IEEE 802.11p	
Communication Range	Vehicles: 145 m RSUs: 250 m	
Routing Protocol	AODV	
Network Traffic Model	CBR	
Network Traffic Rate	2.048 Kbps	
Packet Size	64 B	

on the map to cover 300 by 300-meter cells as shown in Figure 5.12. With the 300 m^2 cells there are small areas on the map that are not covered by the RSUs, so we set the RSU range to 250 m rather than the 212 m that is for the mentioned cell size. Especially, in the beginning, before the 100-second mark when vehicles are still entering the simulation area. Therefore, we ignored the first 100 seconds of simulation plus another 200 seconds to allow the network to be in a stable condition. For simulating the active and passive intrusions in ns-3 we modified the existing AODV protocol to enable blackhole and wormhole attacks. In the simulations for training and test data, 20 vehicles communicate using CBR network traffic model where 10 are

senders and 10 are receivers. Also, in intrusion scenarios, 10 other vehicles act as malicious nodes. After the simulation setup is complete, we collect the network flow data for training our IDS.

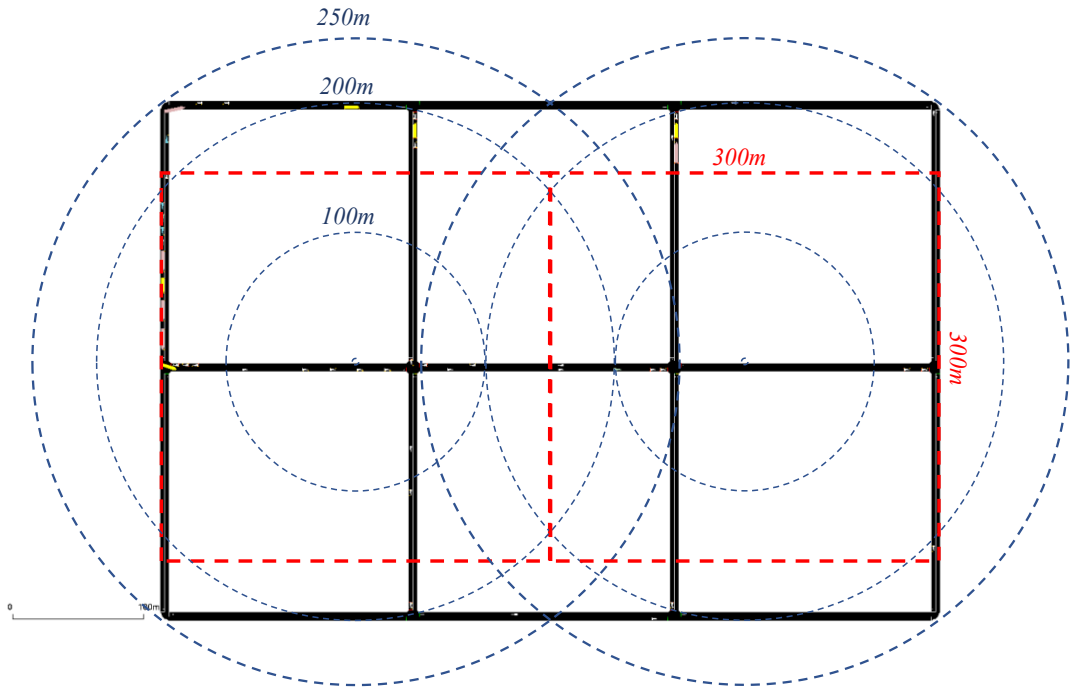

Figure 5.12: Simulation map and RSU placement.

5.4.2 Network Flow Data Collection

Each of the collected samples from the simulation is a network flow that is generated by the flow monitor module of the ns-3. Flows are identified by the sender-destination pair Internet Protocol (IP) addresses. In this module which is also our data gathering module for this IDS, the packet information is collected at the IP level without the need to add any overhead. This also means that any packet retransmits, or similar types of level 4 protocol packets are considered a new packet. The collected data for each packet is related to the time of transmission, size of the packets, or one of the four packet statuses which are *sent*, *forwarded*, *received*, or *dropped*. We either used the collected data directly or calculated new features based on the collected data. In the

end, we created 27 features which are described in Table 5.8. All the features of this dataset are continuous type.

Table 5.8: CAFECNN features for VANET IDS

Feature	Description
current_tx_packets	The number of sent packets in the current segment.
total_tx_packets	The total number of sent packets of the flow.
current_tx_bytes	Size of the sent data in the current segment.
total_tx_bytes	The total size of the sent data in the flow.
first_tx_packet	Time of the first sent packet.
last_tx_packet	Time of the last packet sent.
tx_duration	First - last sent packet time.
tx_pkt_interval	The interval between sent packets in the current segment.
tx_idle_time	Idle time between the current time and the last packet sent.
current_rx_packets	The number of received packets in the current segment.
total_rx_packets	The total number of received packets of the flow.
current_rx_bytes	Size of the received data in the current segment.
total_rx_bytes	The total size of the received data in the flow.
first_rx_packet	Time of the first received packet.
last_rx_packet	Time of the last packet received.
rx_duration	First - last received packet time.
rx_pkt_interval	The interval between received packets in the current segment.
rx_idle_time	Idle time between the current and the last packet received.
delay_sum	Sum of the end-to-end delay.
delay_avg	Average end-to-end delay value.
jitter_sum	Sum of the jitter.
jitter_avg	Average jitter value.
current_lost_packets	The number of lost packets in the current segment.
total_lost_packets	The total number of lost packets.
times_forwarded	The number of times the packets are forwarded.
duration	The total duration of the flow.
throughput	Throughput of the current segment.

The number of flows that can be recorded by sharing data between all RSUs is going to be an excessive amount, therefore, we decided to record the flows every 10 seconds.

This will reduce the number of samples that are needed for training as well as reduce the computational load of the devices.

After collecting the flow data from the simulation, it is time for cleaning and labeling the data. During the cleaning process, we removed all duplicates as well as any sample that has NaN or Inf value. For the labeling part, we labeled the data based on the IP address of the attackers and non-attackers. After the labeling and cleaning is completed, we removed the time and flow-id columns from the dataset and created the train and test data. The training data is all collected from the same simulation while the test data is collected from two different simulations with different traffic rates and packet sizes.

We repeated this process for 3 simulations to collect enough normal and malicious data for training and testing. The final dataset includes 282,319 and 137,972 samples for training and testing, respectively. The class distribution details are given in Table 5.9. As can be observed the normal data has more samples in comparison to the attack samples combined. We mitigate this problem by using class weights during training and also using balanced or weighted metrics for the performance evaluation. From here on we refer to the new dataset as the CVI (CAFECNN VANET IDS).

Table 5.9: New dataset class distribution

Dataset	**Classes**		**Training data**		**Test data**		**Total**
			Individual	**Total**	**Individual**	**Total**	
CVI	0	Normal (Benign)	245,654	282,319	118,774	137,972	420,291
	1	Blackhole	18,613		10,059		
	2	Wormhole	18,052		9,139		

5.5 Proposed IDS for the CVI Dataset

5.5.1 Preprocessing

The next step in creating the CAFECNN IDS for VANET is to do the steps explained in sections 5.1.2 to 5.2.1 with a slight difference in the data augmentation step. The difference is that for the third channel, which is green, we did not change any data and it is identical to the red channel. This redundancy can be useful for some of the features, however, most of it will be filtered out during the SKB feature selection.

During the first step of the preprocessing, 6 features with the lowest ERT scores are eliminated, and finally, at the end of the preprocessing steps, we have a 12 by 12 pixels RGB image for each sample. The eliminated features are *current_tx_packets*, *tx_pkt_interval*, *current_rx_packets*, *current_rx_bytes*, *rx_pkt_interval*, and *current_lost_packets*. A sample image of each class is shown in Figure 5.13.

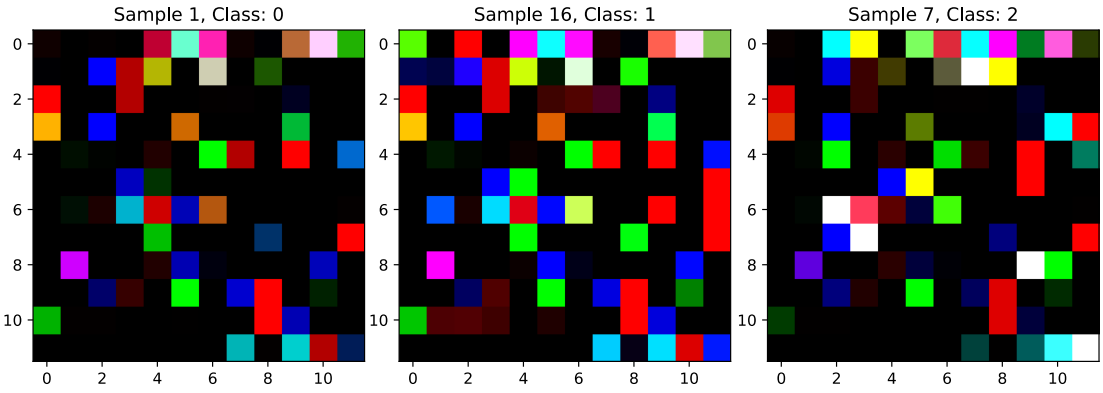

Figure 5.13: Generated samples as the result of CAFECNN preprocessing for the CVI dataset.

5.5.2 Training and Fine-Tuning

Following the preprocessing steps, we train the detection module using a similar CNN structure that we used for the previous benchmark datasets. The next step after the training is to evaluate the model using the test dataset and also use the previously

mentioned transfer learning technique to further fine-tune the model. In this case, we manage to improve the performance of the IDS using transfer learning by removing all the dense layers except the very first and adding new trainable layers. It should be noted that we can improve the Accuracy by some slight amount, however, it would reduce the Balanced Accuracy (BAC). The details of the CNN structure and the fine-tuning layers are available in Table 5.10 and the performance analysis is presented in the next section.

Table 5.10: CNN structure for CVI dataset with and without fine-tuning.

Dataset	**Convolution (Conv)* and Pooling Layers**	**Dense Layers****	**Fine-Tuning****
CVI	3× 2D Conv Layers (64, 3×3, same, ReLU) 1× Average Pooling (pooling size: 2×2) 3× 2D Conv Layers (64, 3×3, same, ReLU) 1× Average Pooling (pooling size: 2×2)	Dense (512, Linear) Dropout (30%) Dense (512, Linear) Dense (60, Linear) Dense (3, SoftMax)	Dense (512, Linear) Dropout (30%) Dense (256, Linear) Dense (60, Linear) Dense (3, SoftMax)

* 2D Conv Layers (filter size, strides, padding, activation function)
** Dense (number of neurons, activation function), Dropout (ratio of ignored neurons)

5.5.3 CAFECNN Intrusion Detection Module

Following the previous steps, we trained new detection modules using five different machine learning methods other than the proposed CAFECNN method. The other methods are RFC and DTC both using gini criteria, MLP, and SVM. After training, we evaluated the result using accuracy. However, simple Accuracy (5.5) which is widely used in literature is best for balanced data and binary classification, and it can be misleading in multi-class classification. For example, if we look at the results using Accuracy as shown in Table 5.11, we have above 90% accuracy for CAFECNN and RFC. However, as suggested in [123] for multi-class classification when the class distribution is imbalanced, it is preferable to use BAC. The BAC represents the arithmetic mean of class recall scores, calculated using (5.9).

$$BAC = \frac{1}{n}\sum_{i=1}^{n}\frac{TP_i}{TP_i + FN_i} \tag{5.9}$$

where TP_i and FN_i are the true positive and the false negative values respectively for the class i, and n is the number of classes.

Table 5.11: Accuracy and BAC of train and test data for the CVI dataset

Model	**Accuracy (%)**		**BAC (%)**	
	Training	**Test**	**Training**	**Test**
CAFECNN	99.87	92.54	99.85	80.03
RFC	100.00	93.19	100.00	78.78
DTC	100.00	89.99	100.00	79.51
SVM	90.96	88.06	54.30	50.17
MLP	90.41	87.11	53.24	52.95

It should be noted that the RFC and DTC methods can naturally reach 100% Accuracy given enough depth, however, the generalization must be evaluated using a proper test dataset. We can also observe from Table 5.11 that the difference in Accuracy and BAC for MLP and SVM are more visible, and the BAC is drastically lower. If we look at the normalized confusion matrix of each model for the test data in Figure 5.14 we can see the reason for this result. As we mentioned before, a passive attack such as wormhole is not easy to detect, and while CAFECNN, RFC, and DTC can detect it to a good degree of around 60%, MLP and SVM fail to find the difference between the normal and wormhole data with around 0% TP rate. Also, excluding the poorly performing SVM and MLP, we can see that CAFECNN has the best TP for blackhole, RFC the highest TP for normal (benign), and DTC has the upper hand TP for the wormhole. However, these differences are marginal and explain the close BAC values of these models.

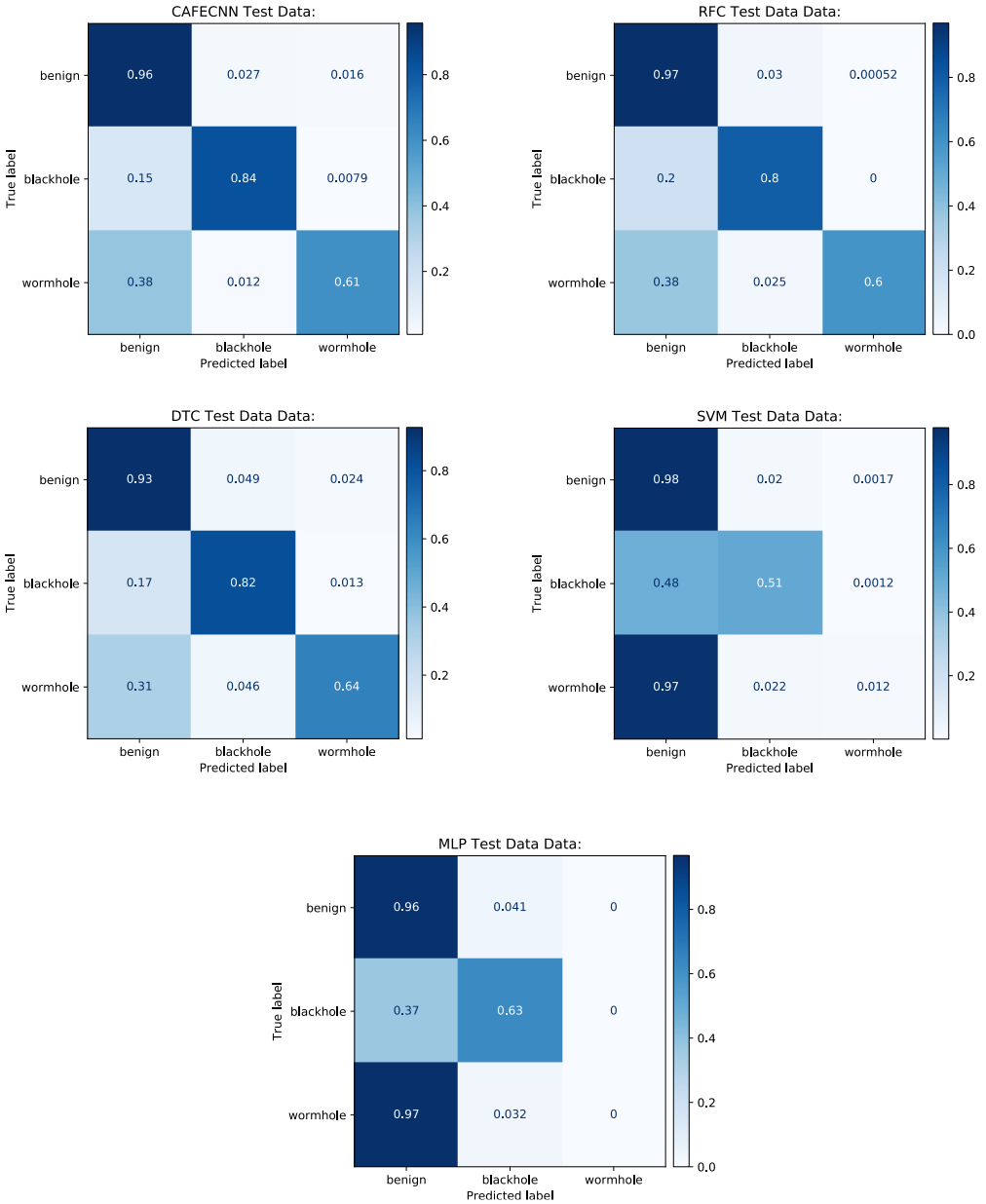

Figure 5.14: Confusion matrix of CVI test data for CAFECNN, RFC, DTC, MLP, and SVM models.

We also evaluated the mentioned models based on the weighted average as well as the macro average of Precision, Recall, and F-Score of the test data. In the previous sections where we used benchmark datasets and for the sake of comparison, we followed the trend of using weighted scores for evaluation purposes, however, as we showed, for imbalanced datasets, it is better to use balanced metrics by macro

averaging. Therefore, here we are presenting both methods for evaluating the final models.

By looking at the weighted metrics in Figure 5.15 we see the same result as the simple Accuracy in Table 5.11 where the RFC model with around 93% Accuracy, WPR, WRC, and WFS shows a slight edge over the others followed by CAFECNN with around 92.5% across the board. DTC, SVM, and MLP have the lowest performance in the mentioned order.

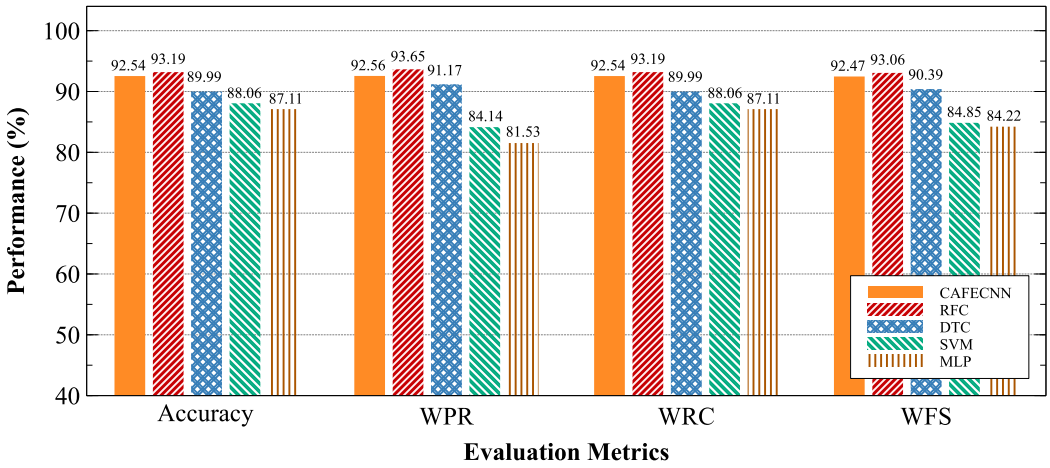

Figure 5.15: Performance comparison of the proposed CAFECNN IDS with the RFC, DTC, SVM, and MLP-based models using the CVI dataset in terms of Accuracy, WPR, WRC, and WFS.

Despite the weighted metrics, when we use balanced or macro averages for performance analysis, we see a different result. As it is shown in Figure 5.16 and as we stated before, CAFECNN shows the highest BAC followed by the DTC and RFC which the latter had the highest simple Accuracy in the previous test. In terms of Macro Precision (MPR) RFC model manages to excel in this score by 87.32%. The reason is, as we can see in Figure 5.14, RFC has a very low FP rate for both blackhole and wormhole attacks. However, both CAFECNN and DTC show a higher Macro Recall

(MRC) rate with 80.03% and 79.51% respectively. The Macro F-Score (MFS) RFC with an 81.27% score is higher than the CAFECNN and DTC with 79.9% and 75.48% respectively. The SVM and MLP both display inferior performance in this test due to failing to classify wormhole attacks successfully.

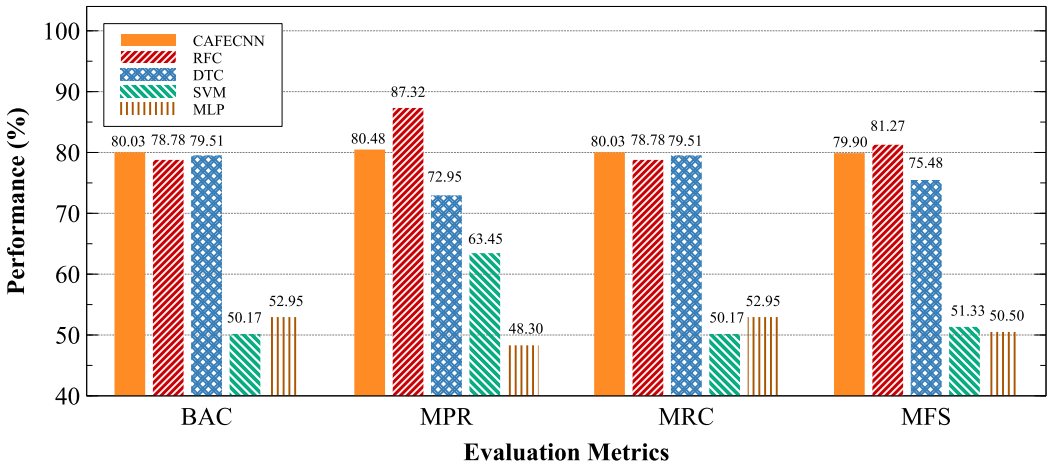

Figure 5.16: Performance comparison of the proposed CAFECNN IDS with the RFC, DTC, SVM, and MLP-based models using the CVI dataset in terms of BAC, MPR, MRC, and MFS.

By comparing both weighted and macro average metrics, we can also observe that the CAFECNN model is exceptionally stable or not always higher than the other methods. In other words, the model manages to keep the same TP and TN across all classes. This is an advantage that makes us decide to use this model for the final parts of the evaluation which is the TVT and real-time simulation result. The latter is presented in Section 5.6.

5.5.4 Trust Value Analysis

Now that the detection module is ready, we need to prepare the response module. Like before, we are going to use TVT in (4.1) for decision-making in the response module. However, for this experiment, we changed the g_t value to one-fourth of the d_t value. We also let the trust value start from 0 and reach the designated upper and lower limit

where the lower limit is the arithmetic inverse of the upper limit. For example, when g_t and d_t are set to 1 and 4 respectively, and the trust value limit is set to 12, vehicles start with the trust score of 0 and gain trust scores by 1 up to 12 or lose the trust score by 4 down to -12. The vehicles with the trust score at the lower limit will be eliminated from participating in the routing process.

For analyzing different trust value limits, we used a scenario with 50 vehicles and 5 blackhole attackers. We evaluated three trust value limits of 4, 12, and 20 using the recorded PDR and EED to see how they are affected in the said simulation scenario. The reason for choosing 3 values to find the best limit is that in our previous observation in Section 4.4.2 we saw that increasing the trust value threshold will only decrease the performance after some point. In our observations, which are shown in Figure 5.17 and Figure 5.18, when the trust value limit is set to 12 we have the best PDR but intermediate EED, while the limit value of 20 has the lowest PDR and EED, and the limit value 4 results in an intermediate PDR with the worst EED. Therefore, we decided to use the limit value 12 for the TVT limit.

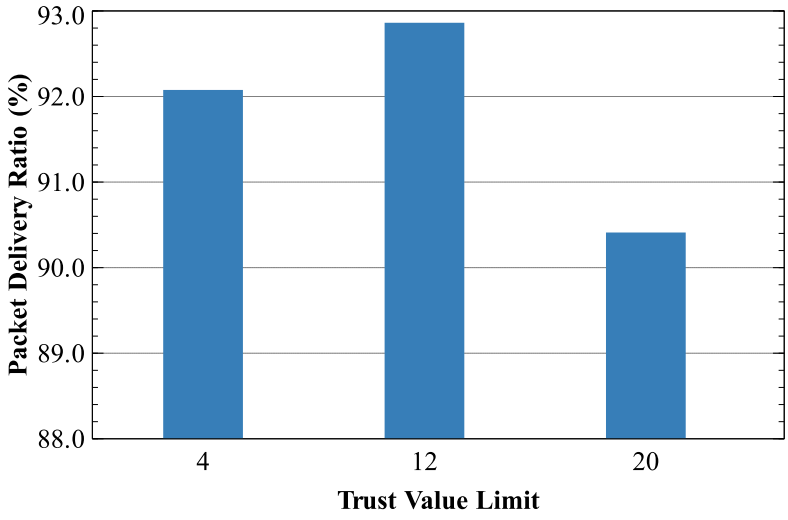

Figure 5.17: Trust value limit performance against PDR with 50 nodes and 5 attackers.

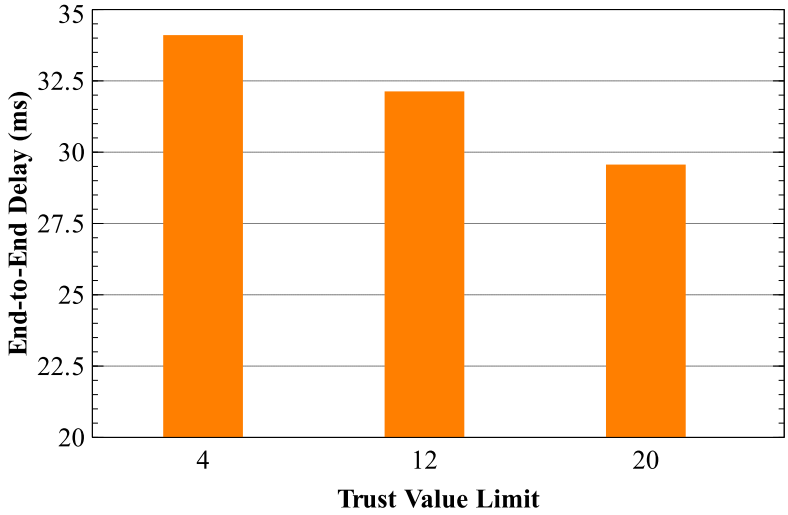

Figure 5.18: Trust value limit performance against EED with 50 nodes and 5 attackers.

After deciding the appropriate trust value limit, we initiated the final simulation setup for evaluating the performance of the complete IDS in real-time scenarios. The results of the simulations are available in Section 5.6.

5.5.5 Computational Complexity of CAFECNN

To assess the computational complexity of the CAFECNN model, we need to consider both preprocessing steps and the CNN classifier. For the preprocessing steps we have:

- FE: Determine the number of pixels in $O(f)$ where f is the number of features in the initial set.
- DA: Flip the original matrix and set it as the green channel $O(p)$, and calculate the standard deviation for each row and column $O(6\sqrt{p})$ where p is the number of pixels generated by the FE step.
- FS: Remove the unnecessary elements $O(6\sqrt{p} + p)$.

Considering the steps above, the time complexity of the preprocessing steps is $O(f + 2p + 12\sqrt{p})$. Considering that our generated images are within 10 to 12 pixels

in width and height, this does not have a significant impact on the processing power of the devices.

As for the CNN algorithm itself, the classification time complexity depends on the size of strides, the number of filters, and the number of convolutional, pooling, and dense layers in the final model. We presented the architecture of each IDS based on the training dataset and we saw that the architecture may vary slightly between all models. However, our proposed models are not as complex and deep as image recognition models with more than 10, 100, or more classes. We also discussed the classification time in μs in Section 5.3 where it is arguably low on the mentioned portable hardware. In the end, CNN is not as lightweight as SVM and it is better to be used on devices with more computational power, or cloud computing services.

5.6 CAFECNN Simulation Result

Similar to the previous section, we prepared different simulations with and without the proposed CAFECNN IDS. The aim is to see if the new IDS can improve the performance of the VANET in terms of PDR and whether it has any effect on EED. The blackhole attack, unlike the packet delaying attack, does not severely affect EED, therefore, included EED in the results to see if there are any adverse effects in presence of the IDS. It should be noted that since the wormhole attack that we executed is not designed to have any active effect on the network, *i.e.,* no follow-up attacks, we only used the blackhole attack to measure the performance improvement. Another difference compared to the previous section is that we used a different number of connections instead of changing the number of vehicles which should not be technically much different. Not all vehicles might need to transfer application packets other than safety messages at all times. The connections are selected in a way that for

k number of connections there are k senders and k receiving nodes. Also, since our previous observations show that the increase in the number of malicious users in smaller numbers follows the same pattern, we skipped small increments in the number of attackers, and instead, we prepared three scenarios with 0, 5, and 10 attackers. As we decided before, the trust value limit for all simulations is set to 12 with d_t and g_t equal to 4 and 1, respectively. The simulation time is set to 2000 seconds for all scenarios.

The first scenario includes 50 vehicles with 10 connections. For the PDR test we can observe in Figure 5.19 that while the attackers can substantially reduce the PDR by 10 to 15 percent, the proposed CAFECNN IDS reduces this gap. The same can be said about EED, where in Figure 5.20 we can see that the delay is comparably lower when the IDS manages to eliminate the attackers. It should be noted that the difference in delay is not as severe as in the previous section with the packet delaying attack.

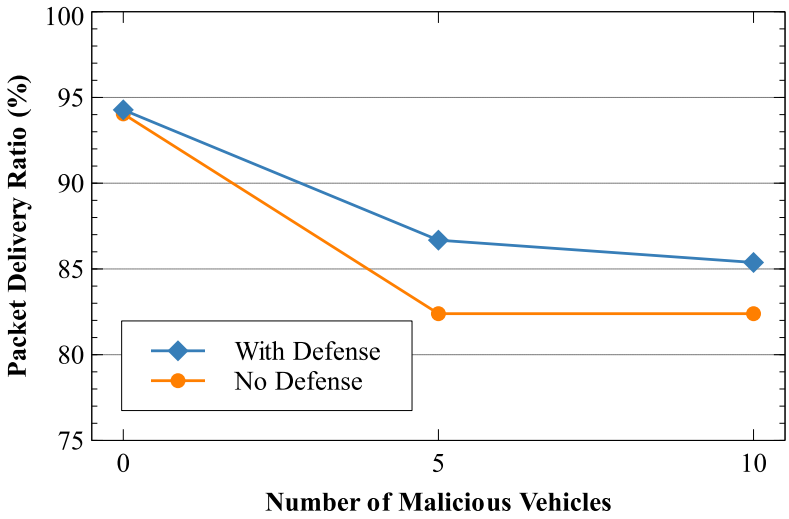

Figure 5.19: CAFECNN performance analysis for PDR in 50 vehicles and 10 connections scenario.

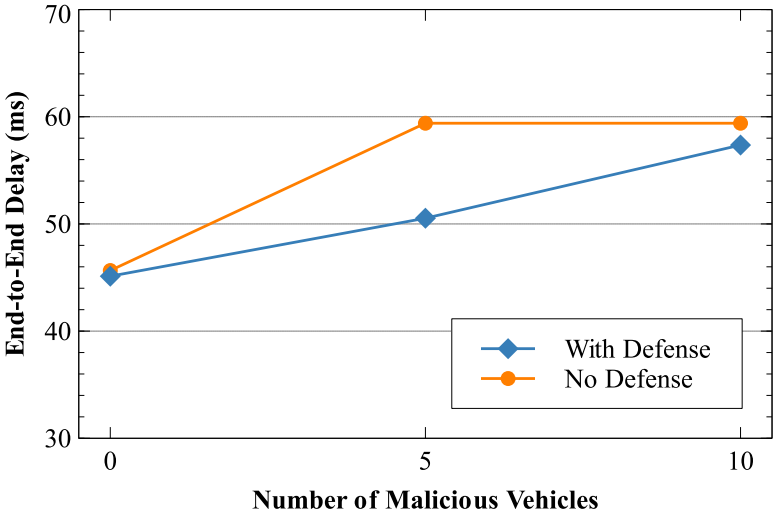

Figure 5.20: CAFECNN performance analysis for EED in 50 vehicles and 10 connections scenario.

The next scenario includes the same number of vehicles but with 20 connections. The graph in Figure 5.21 shows that the PDR performance hit in presence of attackers is more severe than in the previous scenario. Nevertheless, the proposed IDS is still able to identify and eliminate attackers which leads to overall performance improvement. Similarly, when we examine Figure 5.22, the EED performance improves when there are 5 attackers in the network. However, in the 10 attackers simulation, the EED is more or less the same. The reason for this behavior can be explained by looking at the network behavior and nature of the blackhole attack. When a malicious user succeeds in executing the blackhole attack it means that there is a chance that either it was the only available route, or it was actually the shortest route to the destination. Even though the latter can be falsified by the attacker. Therefore, when the IDS eliminates the attacker, it forces a new route discovery process that might go through a longer path or through a node that is actively forwarding packets from other routes. This can cause the overall EED to go up, however, as we can see in the graph, the difference is negligible.

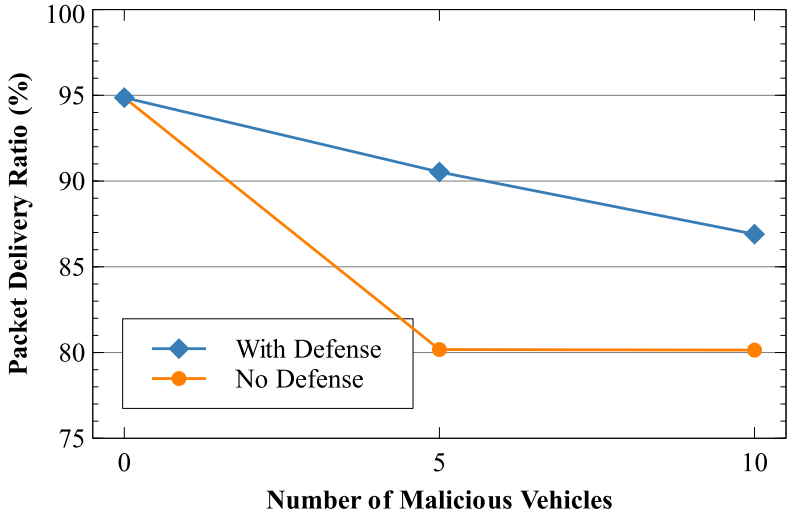

Figure 5.21: CAFECNN performance analysis for PDR in 50 vehicles and 20 connections scenario.

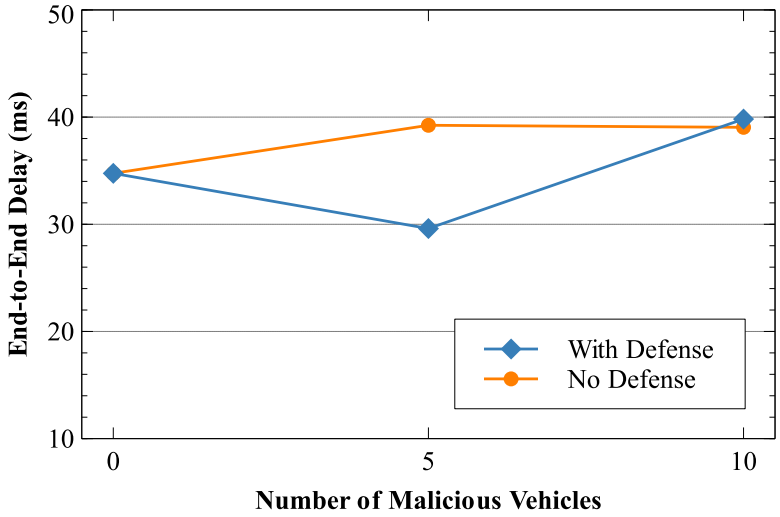

Figure 5.22: CAFECNN performance analysis for EED in 50 vehicles and 20 connections scenario.

In conclusion, the CAFECNN IDS is able to negate the adverse effect of the blackhole attack without having an impact on the other aspects of the network.

For the wormhole attack, we can see that according to Figure 5.23 and Figure 5.24 there are no immediate effects on the performance of the network. Therefore, simply eliminating the attackers and forcing re-authentication for the affected users can solve

the possible harmful effect of the follow-up attacks. The displayed results are for 20 connections, 5 wormhole tunnels, and 10 blackhole attackers.

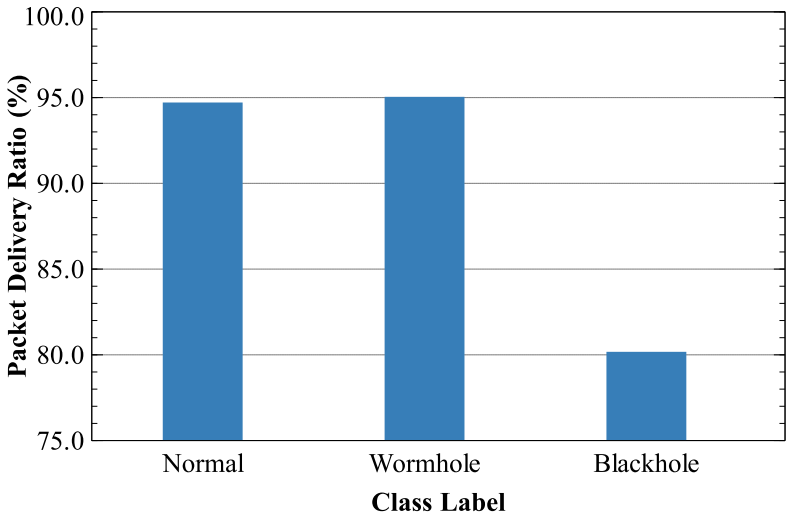

Figure 5.23: Performance impact of attacks on the VANET in terms of PDR.

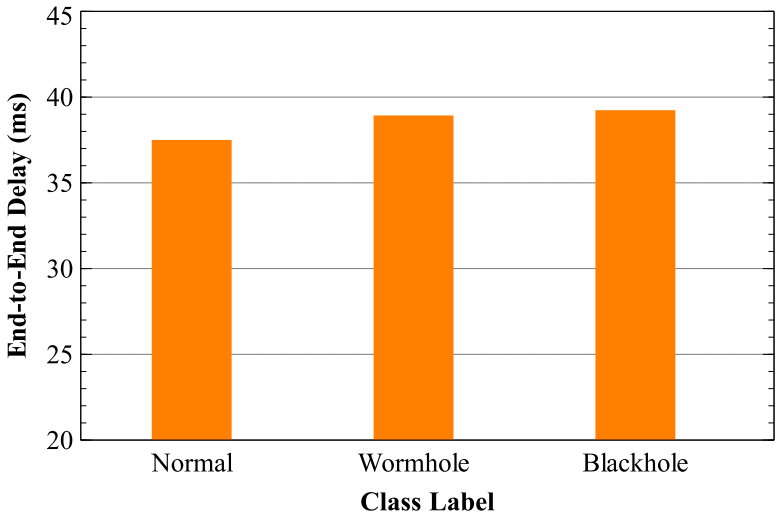

Figure 5.24: Performance impact of attacks on the VANET in terms of EED.

Chapter 6

CONCLUSION AND FUTURE STUDY

6.1 Conclusion

The safety and convenience of an interconnected autonomous transportation system are not dependable without proper defensive mechanisms against malicious users. The performance of the VANET reduces substantially when attackers are targeting the availability and integrity of the network. IPS and IDS solutions are two effective ways to counter the security issues that come with VANET. We investigated ML tools as an effective way to create an IDS with a high detection rate in a short amount of time.

As a result of this study, we designed two IDS models to address the impact of the intruders in VANET. Our first IDS which we refer to it as TSIDS, uses SVM with three simple yet effective inputs to detect packet dropping and packet forward delaying attacks. This IDS is lightweight and can be installed on any vehicle. In this model, every vehicle that is forwarding packets through the network uses a modified promiscuous mode to monitor the next hop in the routing process to see any signs of malicious activity. TSIDS can effectively detect and eliminate the threats in the network and improve the overall performance of the VANET.

The second IDS that we proposed uses the network flow data that is gathered by RSUs to detect the passive and active types of attacks. In this model, we proposed a new feature extraction method to convert the flow data into RGB images that are going to

be used by the CNN-based classifier. We refer to this IDS as CAFECNN and even though it is more complex it is more effective than SVM in detecting wormhole attacks. Nevertheless, similar to TSIDS, this model can also effectively improve the performance of the network in terms of PDR by roughly 7%.

The final module of both proposed IDS is a trust-based response module that is designed to prevent false alarms resulting in the elimination of legitimate users. We exhaustively investigated the effect of the TVT using different initial values and limits to find the best score. In our experiments, the best initial value/limit combination is when the attacking users are eliminated by three consecutive malicious flags.

6.2 Future Study

The cybersecurity solutions for VANET are still in the preliminary stages and they are mostly proactive measures for a technology that is going to be widespread in the near future. We are planning to investigate IDS models that would provide network security as well as software and privacy protection. Software solutions for VANETs are still evolving, however, we can speculate that the types of intrusions are similar to IoT devices where the attackers target the sensor information for either eavesdropping to falsifying attacks. In our future study, we aim to provide complementary solutions to the IDS in the form of authentication methods that are tailored for the VANET environment.

Another issue in the autonomous transportation ecosystem is traffic management through ITS. This is also an ongoing popular research area that we would like to address in our future studies.

REFERENCES

[1] A.-S. K. Pathan, *The State of the Art in Intrusion Prevention and Detection*. CRC press, 2014. doi: 10.1201/b16390.

[2] Y. Otoum and A. Nayak, "AS-IDS: Anomaly and signature based IDS for the internet of things," *J. Netw. Syst. Manag.*, vol. 29, no. 3, Mar. 2021, doi: 10.1007/s10922-021-09589-6.

[3] G. Kaur, A. Bansal, and A. Agarwal, "Wavelets based anomaly-based detection system or J48 and naïve bayes based signature-based detection system: A comparison," in *Advances in Intelligent Systems and Computing*, 2018, vol. 696, pp. 213–224. doi: 10.1007/978-981-10-7386-1_19.

[4] E. A. Shams, A. Rizaner, and A. H. Ulusoy, "A novel context-aware feature extraction method for convolutional neural network-based intrusion detection systems," *Neural Comput. Appl.*, vol. 33, no. 20, pp. 13647–13665, Oct. 2021, doi: 10.1007/s00521-021-05994-9.

[5] KDD, "Kdd Cup 1999," *University of California, Irvine (UCI)*, 1999. http://www.kdd.org/kdd-cup/view/kdd-cup-1999 (accessed Oct. 25, 2019).

[6] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *IEEE Symposium on Computational Intelligence for Security and Defense Applications, CISDA 2009*, 2009, no. Cisda, pp. 1–6. doi: 10.1109/CISDA.2009.5356528.

[7] "NSL-KDD," *University of New Brunswick (UNB)*. https://www.unb.ca/cic/datasets/nsl.html (accessed Oct. 25, 2019).

[8] R. P. Lippmann *et al.*, "Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation," *Proc. - DARPA Inf. Surviv. Conf. Expo. DISCEX 2000*, vol. 2, pp. 12–26, 2000, doi: 10.1109/DISCEX.2000.821506.

[9] G. Creech and J. Hu, "Generation of a new IDS test dataset: Time to retire the KDD collection," *IEEE Wirel. Commun. Netw. Conf. WCNC*, pp. 4487–4492, 2013, doi: 10.1109/WCNC.2013.6555301.

[10] I. Sharafaldin, A. Habibi Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proceedings of the 4th International Conference on Information Systems Security and Privacy*, 2018, pp. 108–116. doi: 10.5220/0006639801080116.

[11] G. Creech and J. Hu, "A semantic approach to host-based intrusion detection systems using contiguousand discontiguous system call patterns," *IEEE Trans. Comput.*, vol. 63, no. 4, pp. 807–819, 2014, doi: 10.1109/TC.2013.13.

[12] G. Creech, "Developing a high-accuracy cross platform host-based intrusion detection system capable of reliably detecting zero-day attacks," University of New South Wales, 2014.

[13] Kyoto University, “Kyoto 2006+,” 2015. http://www.takakura.com/Kyoto_data/ (accessed Feb. 25, 2020).

[14] University of New South Wales, “UNSW-NB15,” 2017. https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets/ (accessed Feb. 25, 2020).

[15] University of the Aegean, “AWID Dataset,” 2018. http://icsdweb.aegean.gr/awid/features.html (accessed Feb. 25, 2020).

[16] D. Mishkin, N. Sergievskiy, and J. Matas, “Systematic evaluation of convolution neural network advances on the Imagenet,” *Comput. Vis. IMAGE Underst.*, vol. 161, pp. 11–19, Aug. 2017, doi: 10.1016/j.cviu.2017.05.007.

[17] E. A. Shams, A. Rizaner, and A. H. Ulusoy, “Trust aware support vector machine intrusion detection and prevention system in vehicular ad hoc networks,” *Comput. Secur.*, vol. 78, pp. 245–254, Sep. 2018, doi: 10.1016/j.cose.2018.06.008.

[18] L. N. Tidjon, M. Frappier, and A. Mammar, “Intrusion detection systems: A cross-domain overview,” *IEEE Commun. Surv. Tutorials*, vol. 21, no. 4, pp. 3639–3681, 2019, doi: 10.1109/COMST.2019.2922584.

[19] H. Choi, M. Kim, G. Lee, and W. Kim, “Unsupervised learning approach for network intrusion detection system using autoencoders,” *J. Supercomput.*, vol. 75, no. 9, pp. 5597–5621, 2019, doi: 10.1007/s11227-019-02805-w.

[20] A. Kaur, S. K. Pal, and A. P. Singh, “Hybridization of K-Means and Firefly algorithm for intrusion detection system,” *Int. J. Syst. Assur. Eng. Manag.*, vol. 9, no. 4, pp. 901–910, 2018, doi: 10.1007/s13198-017-0683-8.

[21] M. Latah and L. Toker, “Towards an efficient anomaly-based intrusion detection for software-defined networks,” *IET Networks*, vol. 7, no. 6, pp. 453–459, 2018, doi: 10.1049/iet-net.2018.5080.

[22] E. A. Shams, A. Rizaner, and A. H. Ulusoy, “Trust aware support vector machine intrusion detection and prevention system in vehicular ad hoc networks,” *Comput. Secur.*, vol. 78, pp. 245–254, 2018, doi: 10.1016/j.cose.2018.06.008.

[23] A. Alabdallah and M. Awad, “Using weighted support vector machine to address the imbalanced classes problem of intrusion detection system,” *KSII Trans. Internet Inf. Syst.*, vol. 12, no. 10, pp. 5143–5158, 2018, doi: 10.3837/tiis.2018.10.027.

[24] M. A. Ambusaidi, X. He, P. Nanda, and Z. Tan, “Building an intrusion detection system using a filter-based feature selection algorithm,” *IEEE Trans. Comput.*, vol. 65, no. 10, pp. 2986–2998, 2016, doi: 10.1109/TC.2016.2519914.

[25] C. Yin, Y. Zhu, J. Fei, and X. He, “A deep learning approach for intrusion detection using recurrent neural networks,” *IEEE Access*, vol. 5, pp. 21954–21961, 2017, doi: 10.1109/ACCESS.2017.2762418.

[26] P. Singh, S. Krishnamoorthy, A. Nayyar, A. K. Luhach, and A. Kaur, “Soft-computing-based false alarm reduction for hierarchical data of intrusion detection system,” *Int. J. Distrib. Sens. Networks*, vol. 15, no. 10, 2019, doi: 10.1177/1550147719883132.

[27] R. Blanco, P. Malagon, J. J. Cilla, and J. M. Moya, “Multiclass network attack classifier using cnn tuned with genetic algorithms,” in *2018 28th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, Jul. 2018, pp. 177–182. doi: 10.1109/PATMOS.2018.8463997.

[28] Y. Ding and Y. Zhai, “Intrusion detection system for NSL-KDD dataset using convolutional neural networks,” *ACM Int. Conf. Proceeding Ser.*, pp. 81–85, 2018, doi: 10.1145/3297156.3297230.

[29] Y. Li *et al.*, “Robust detection for network intrusion of industrial IoT based on multi-CNN fusion,” *Meas. J. Int. Meas. Confed.*, vol. 154, p. 107450, 2020, doi: 10.1016/j.measurement.2019.107450.

[30] Kurniabudi, D. Stiawan, Darmawijoyo, M. Y. Bin Idris, A. M. Bamhdi, and R. Budiarto, “CICIDS-2017 dataset feature analysis with information gain for anomaly detection,” *IEEE Access*, vol. 8, pp. 132911–132921, 2020, doi: 10.1109/ACCESS.2020.3009843.

[31] Z. Chiba, N. Abghour, K. Moussaid, A. El Omri, and M. Rida, “Intelligent approach to build a deep neural network based IDS for cloud environment using combination of machine learning algorithms,” *Comput. Secur.*, vol. 86, pp. 291–317, Sep. 2019, doi: 10.1016/j.cose.2019.06.013.

[32] J. Malik, A. Akhunzada, I. Bibi, M. Imran, A. Musaddiq, and S. W. Kim, “Hybrid deep learning: an efficient reconnaissance and surveillance detection mechanism in SDN,” *IEEE Access*, vol. 8, pp. 134695–134706, 2020, doi: 10.1109/ACCESS.2020.3009849.

[33] P. Sun *et al.*, “DL-IDS: Extracting features using CNN-LSTM hybrid network for intrusion detection system,” *Secur. Commun. Networks*, vol. 2020, pp. 1–11, Aug. 2020, doi: 10.1155/2020/8890306.

[34] Y. Zhang, X. Chen, L. Jin, X. Wang, and D. Guo, “Network intrusion detection: Based on deep hierarchical network and original flow data,” *IEEE Access*, vol. 7, pp. 37004–37016, 2019, doi: 10.1109/ACCESS.2019.2905041.

[35] M. A. Ferrag and L. Maglaras, “DeepCoin: A novel deep learning and blockchain-based energy exchange framework for smart grids,” *IEEE Trans. Eng. Manag.*, vol. 67, no. 4, pp. 1285–1297, Nov. 2020, doi: 10.1109/TEM.2019.2922936.

[36] W. Elmasry, A. Akbulut, and A. H. Zaim, “Empirical study on multiclass classification-based network intrusion detection,” *Comput. Intell.*, no. March, pp. 919–954, 2019, doi: 10.1111/coin.12220.

[37] S. Lv, J. Wang, Y. Yang, and J. Liu, “Intrusion prediction with system-call sequence-to-sequence model,” *IEEE Access*, vol. 6, pp. 71413–71421, 2018, doi: 10.1109/ACCESS.2018.2881561.

[38] G. Serpen and E. Aghaei, “Host-based misuse intrusion detection using PCA feature extraction and kNN classification algorithms,” *Intell. Data Anal.*, vol. 22, no. 5, pp. 1101–1114, 2018, doi: 10.3233/IDA-173493.

[39] R. Vijayanand, D. Devaraj, and B. Kannapiran, “A novel intrusion detection system for wireless mesh network with hybrid feature selection technique based on GA and MI,” *J. Intell. Fuzzy Syst.*, vol. 34, no. 3, pp. 1243–1250, 2018, doi: 10.3233/JIFS-169421.

[40] N. N. Tran, R. Sarker, and J. Hu, “An approach for host-based intrusion detection system design using convolutional neural network,” in *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST*, vol. 235, 2018, pp. 116–126. doi: 10.1007/978-3-319-90775-8_10.

[41] Y. Shin and K. Kim, “Comparison of anomaly detection accuracy of host-based intrusion detection systems based on different machine learning algorithms,” *Int. J. Adv. Comput. Sci. Appl.*, vol. 11, no. 2, pp. 252–259, 2020, doi: 10.14569/ijacsa.2020.0110233.

[42] A. Tomovic, P. Janicic, and V. Keselj, “n-Gram-based classification and unsupervised hierarchical clustering of genome sequences,” *Comput. Methods Programs Biomed.*, vol. 81, no. 2, pp. 137–153, Feb. 2006, doi: 10.1016/j.cmpb.2005.11.007.

[43] B. S. Khater, A. W. B. A. Wahab, M. Y. I. Bin Idris, M. A. Hussain, and A. A. Ibrahim, “A lightweight perceptron-based intrusion detection system for fog computing,” *Appl. Sci.*, vol. 9, no. 1, 2019, doi: 10.3390/app9010178.

[44] S. Mahdavifar and A. A. Ghorbani, “Application of deep learning to cybersecurity: A survey,” *Neurocomputing*, vol. 347, pp. 149–176, 2019, doi: 10.1016/j.neucom.2019.02.056.

[45] S. Sharma and A. Kaul, “A survey on intrusion detection systems and honeypot based proactive security mechanisms in VANETs and VANET cloud,” *Veh. Commun.*, vol. 12, pp. 138–164, Apr. 2018, doi: 10.1016/j.vehcom.2018.04.005.

[46] J. Hoopes, Ed., “Honeypotting,” in *Virtualization for Security*, Boston: Elsevier, 2009, pp. 117–143. doi: 10.1016/b978-1-59749-305-5.00005-0.

[47] R. W. Van Der Heijden, S. Dietzel, T. Leinmüller, and F. Kargl, “Survey on misbehavior detection in cooperative intelligent transportation systems,” *IEEE Commun. Surv. Tutorials*, vol. 21, no. 1, pp. 779–811, 2019, doi: 10.1109/COMST.2018.2873088.

[48] M. Hasan, S. Mohan, T. Shimizu, and H. Lu, “Securing Vehicle-to-Everything (V2X) communication platforms,” *IEEE Trans. Intell. Veh.*, vol. 5, no. 4, pp. 693–713, Dec. 2020, doi: 10.1109/TIV.2020.2987430.

[49] A. Hbaieb, S. Ayed, and L. Chaari, “A survey of trust management in the internet of vehicles,” *Comput. Networks*, vol. 203, Feb. 2022, doi: 10.1016/j.comnet.2021.108558.

[50] T. Zhang and Q. Zhu, “Distributed privacy-preserving collaborative intrusion detection systems for VANETs,” *IEEE Trans. Signal Inf. Process. over Networks*, vol. 4, no. 1, pp. 148–161, 2018, doi: 10.1109/TSIPN.2018.2801622.

[51] B. Subba, S. Biswas, and S. Karmakar, “A game theory based multi layered intrusion detection framework for wireless sensor networks,” *Int. J. Wirel. Inf. Networks*, vol. 25, no. 4, pp. 399–421, 2018, doi: 10.1007/s10776-018-0403-6.

[52] “ns-3 | a discrete-event network simulator for internet systems,” 2022. https://www.nsnam.org/ (accessed Jun. 28, 2021).

[53] “SUMO Simulation of Urban MObility.” http://sumo.dlr.de/ (accessed Oct. 22, 2017).

[54] “The Network Simulator 2 - ns-2.” http://www.isi.edu/nsnam/ns/ (accessed Oct. 22, 2017).

[55] S. Sharma and A. Kaul, “Hybrid fuzzy multi-criteria decision making based multi cluster head dolphin swarm optimized IDS for VANET,” *Veh. Commun.*, vol. 12, pp. 23–38, Apr. 2018, doi: 10.1016/j.vehcom.2017.12.003.

[56] Y. Gao, H. Wu, B. Song, Y. Jin, X. Luo, and X. Zeng, “A distributed network intrusion detection system for distributed denial of service attacks in vehicular ad hoc network,” *IEEE Access*, vol. 7, pp. 154560–154571, 2019, doi: 10.1109/ACCESS.2019.2948382.

[57] A. Meddeb Makhlouf and M. Guizani, “SE-AOMDV: Secure and efficient AOMDV routing protocol for vehicular communications,” *Int. J. Inf. Secur.*, vol. 18, no. 5, pp. 665–676, Oct. 2019, doi: 10.1007/s10207-019-00436-z.

[58] M. Poongodi, M. Hamdi, A. Sharma, M. Ma, and P. K. Singh, “DDoS detection mechanism using trust-based evaluation system in VANET,” *IEEE Access*, vol. 7, pp. 183532–183544, 2019, doi: 10.1109/ACCESS.2019.2960367.

[59] L. Nie, Y. Wu, H. Wang, and Y. Li, “Anomaly detection based on spatio-temporal and sparse features of network traffic in VANETs,” *IEEE Access*, vol. 7, pp. 177954–177964, 2019, doi: 10.1109/ACCESS.2019.2958068.

[60] K. N. Tripathi and S. C. Sharma, “A trust based model (TBM) to detect rogue nodes in vehicular ad-hoc networks (VANETS),” *Int. J. Syst. Assur. Eng. Manag.*, vol. 11, no. 2, pp. 426–440, Apr. 2020, doi: 10.1007/s13198-019-00871-0.

[61] F. Z. Mostefa, Z. M. Maaza, and C. Duvallet, “Secure communications by tit-for-tat strategy in vehicular networks,” *Int. J. Networked Distrib. Comput.*, vol. 8, no. 4, pp. 214–221, Oct. 2020, doi: 10.2991/IJNDC.K.200925.001.

[62] M. Zhou, L. Han, H. Lu, and C. Fu, “Distributed collaborative intrusion detection system for vehicular Ad Hoc networks based on invariant,” *Comput. Networks*, vol. 172, May 2020, doi: 10.1016/j.comnet.2020.107174.

[63] A. Wegener, M. Piórkowski, M. Raya, H. Hellbrück, S. Fischer, and J. P. Hubaux, “TraCI: An interface for coupling road traffic and network simulators,” in *Proceedings of the 11th Communications and Networking Simulation Symposium, CNS’08*, 2008, pp. 155–163. doi: 10.1145/1400713.1400740.

[64] “OMENT++ discrete event simulator,” 2022. https://omnetpp.org/ (accessed Jun. 26, 2022).

[65] R. Kolandaisamy, R. M. Noor, M. R. Z’aba, I. Ahmedy, and I. Kolandaisamy, “Adapted stream region for packet marking based on DDoS attack detection in vehicular ad hoc networks,” *J. Supercomput.*, vol. 76, no. 8, pp. 5948–5970, Aug. 2020, doi: 10.1007/s11227-019-03088-x.

[66] R. Rahal, A. Amara Korba, and N. Ghoualmi-Zine, “Towards the development of realistic dos dataset for intelligent transportation systems,” *Wirel. Pers. Commun.*, vol. 115, no. 2, pp. 1415–1444, Nov. 2020, doi: 10.1007/s11277-020-07635-1.

[67] B. Amar Bensaber, C. G. Pereira Diaz, and Y. Lahrouni, “Design and modeling an Adaptive Neuro-Fuzzy Inference System (ANFIS) for the prediction of a security index in VANET,” *J. Comput. Sci.*, vol. 47, Nov. 2020, doi: 10.1016/j.jocs.2020.101234.

[68] C. Sommer, R. German, and F. Dressler, “Bidirectionally coupled network and road simulation for improved IVC analysis,” *IEEE Trans. Mob. Comput.*, vol. 10, no. 1, pp. 3–15, Jan. 2011, doi: 10.1109/TMC.2010.133.

[69] D. A. Schmidt, M. S. Khan, and B. T. Bennett, “Spline-based intrusion detection for VANET utilizing knot flow classification,” *Internet Technol. Lett.*, vol. 3, no. 3, May 2020, doi: 10.1002/itl2.155.

[70] F. A. Ghaleb *et al.*, “Misbehavior-aware on-demand collaborative intrusion detection system using distributed ensemble learning for VANET,” *Electron.*, vol. 9, no. 9, pp. 1–17, Sep. 2020, doi: 10.3390/electronics9091411.

[71] A. Alsarhan, M. Alauthman, E. Alshdaifat, A. R. Al-Ghuwairi, and A. Al-Dubai, “Machine Learning-driven optimization for SVM-based intrusion detection system in vehicular ad hoc networks,” *J. Ambient Intell. Humaniz. Comput.*, 2021, doi: 10.1007/s12652-021-02963-x.

[72] K. Stępień and A. Poniszewska-Marańda, “Security measures with enhanced behavior processing and footprint algorithm against sybil and bogus attacks in vehicular ad hoc network,” *Sensors*, vol. 21, no. 10, May 2021, doi: 10.3390/s21103538.

[73] F. A. Ghaleb, A. Zainal, M. A. Maroof, M. A. Rassam, and F. Saeed, “Detecting bogus information attack in vehicular ad hoc network: A context-aware approach,” *Procedia Comput. Sci.*, vol. 163, pp. 180–189, 2019, doi: 10.1016/j.procs.2019.12.099.

[74] B. Yu, C.-Z. Xu, and B. Xiao, “Detecting Sybil attacks in VANETs,” *J. Parallel Distrib. Comput.*, vol. 73, no. 6, pp. 746–756, Jun. 2013, doi: 10.1016/j.jpdc.2013.02.001.

[75] K. Stepień and A. Poniszewska-Marańda, “Security measures in vehicular ad-hoc networks on the example of bogus and sybil attacks,” in *Advances in Intelligent Systems and Computing*, 2020, vol. 1151 AISC, pp. 419–430. doi: 10.1007/978-3-030-44041-1_38.

[76] N. C. Velayudhan, A. Anitha, and M. Madanan, “Sybil attack detection and secure data transmission in VANET using CMEHA-DNN and MD5-ECC,” *J. Ambient Intell. Humaniz. Comput.*, Jul. 2021, doi: 10.1007/s12652-021-03379-3.

[77] G. Raja, S. Anbalagan, G. Vijayaraghavan, P. Dhanasekaran, Y. D. Al-Otaibi, and A. K. Bashir, “Energy-efficient end-to-end security for software-defined vehicular networks,” *IEEE Trans. Ind. Informatics*, vol. 17, no. 8, pp. 5730–5737, Aug. 2021, doi: 10.1109/TII.2020.3012166.

[78] H. Bangui, M. Ge, and B. Buhnova, “A hybrid machine learning model for intrusion detection in VANET,” *Computing*, 2021, doi: 10.1007/s00607-021-01001-0.

[79] F. Gonçalves, J. Macedo, and A. Santos, “An intelligent hierachical security framework for vanets,” *Inf.*, vol. 12, no. 11, Nov. 2021, doi: 10.3390/info12110455.

[80] F. Goncalves *et al.*, “Synthesizing datasets with security threats for vehicular ad-hoc networks,” in *2020 IEEE Global Communications Conference, GLOBECOM 2020 - Proceedings*, 2020, vol. 2020-Janua, pp. 1–6. doi: 10.1109/GLOBECOM42002.2020.9348149.

[81] H. Sedjelmaci, I. H. Brahmi, N. Ansari, and M. H. Rehmani, “Cyber security framework for vehicular network based on a hierarchical game,” *IEEE Trans. Emerg. Top. Comput.*, vol. 9, no. 1, pp. 429–440, Jan. 2021, doi: 10.1109/TETC.2018.2890476.

[82] J. Shu, L. Zhou, W. Zhang, X. Du, and M. Guizani, “Collaborative intrusion detection for VANETs: A deep learning-based distributed SDN approach,” *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 7, pp. 4519–4530, Jul. 2021, doi: 10.1109/TITS.2020.3027390.

[83] A. Sharma and A. Jaekel, “Machine learning based misbehaviour detection in VANET using consecutive BSM approach,” *IEEE Open J. Veh. Technol.*, vol. 3, pp. 1–14, 2022, doi: 10.1109/OJVT.2021.3138354.

[84] R. W. van der Heijden, T. Lukaseder, and F. Kargl, “VeReMi: A dataset for comparable evaluation of misbehavior detection in VANETs,” in *International conference on security and privacy in communication systems*, 2018, pp. 318–337. doi: 10.1007/978-3-030-01701-9_18.

[85] M. Najafi, L. Khoukhi, and M. Lemercier, “Decentralized prediction and reputation approach in vehicular networks,” *Trans. Emerg. Telecommun. Technol.*, 2022, doi: 10.1002/ett.4456.

[86] R. Lavanya and S. Kannan, “Intrusion detection system for energy efficient cluster based vehicular adhoc networks,” *Intell. Autom. Soft Comput.*, vol. 32, no. 1, pp. 323–337, 2022, doi: 10.32604/iasc.2022.021467.

[87] Bitdefender, “2021 consumer threat landscape report,” 2022. [Online]. Available: https://www.bitdefender.com/files/News/CaseStudies/study/418/bitdefender-landscape-report-2021.pdf

[88] Q. G. Fan, L. Wang, Y. N. Cai, Y. Q. Li, and J. Chen, “VANET routing replay attack detection research based on SVM,” in *2016 International conference on mechatronics, manufacturing and materials engineering (MMME 2016)*, 2016, vol. 63. doi: 10.1051/matecconf/20166305020.

[89] R. Regan and J. M. L. Manickam, “A survey on impersonation attack in wireless networks,” *Int. J. Secur. ITS Appl.*, vol. 11, no. 5, pp. 39–48, May 2017, doi: 10.14257/ijsia.2017.11.5.04.

[90] M. Ring, S. Wunderlich, D. Scheuring, D. Landes, and A. Hotho, “A survey of network-based intrusion detection data sets,” *Comput. Secur.*, vol. 86, pp. 147–167, 2019, doi: 10.1016/j.cose.2019.06.005.

[91] G. Mullen and L. Meany, “Assessment of buffer overflow based attacks on an IoT operating system,” in *2019 Global IoT Summit (GIoTS)*, Jun. 2019, pp. 1–6. doi: 10.1109/GIOTS.2019.8766434.

[92] R. Amici, M. Bonola, L. Bracciale, A. Rabuffi, P. Loreti, and G. Bianchi, “Performance assessment of an epidemic protocol in VANET using real traces,” *Procedia Comput. Sci.*, vol. 40, no. C, pp. 92–99, 2014, doi: 10.1016/j.procs.2014.10.035.

[93] C. Celes, R. B. Braga, C. T. De Oliveira, R. M. C. Andrade, and A. A. F. Loureiro, “GeoSPIN: An approach for Geocast routing based on SPatial INformation in VANETs,” in *2013 IEEE 78th Vehicular Technology Conference (VTC Fall)*, Sep. 2013, pp. 1–6. doi: 10.1109/VTCFall.2013.6692215.

[94] J. Yuan *et al.*, “T-drive: Driving directions based on taxi trajectories,” in *GIS ’10: Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2010, pp. 99–108. doi: 10.1145/1869790.1869807.

[95] S. Uppoor, O. Trullols-Cruces, M. Fiore, and J. M. Barcelo-Ordinas, "Generation and analysis of a large-scale urban vehicular mobility dataset," *IEEE Trans. Mob. Comput.*, vol. 13, no. 5, pp. 1061–1075, May 2014, doi: 10.1109/TMC.2013.27.

[96] C. Celes, F. A. Silva, A. Boukerche, R. M. D. C. Andrade, and A. A. F. Loureiro, "Improving VANET simulation with calibrated vehicular mobility traces," *IEEE Trans. Mob. Comput.*, vol. 16, no. 12, pp. 3376–3389, 2017, doi: 10.1109/TMC.2017.2690636.

[97] F. K. Karnadi, Z. H. Mo, and K. C. Lan, "Rapid generation of realistic mobility models for VANET," in *IEEE Wireless Communications and Networking Conference, WCNC*, 2007, pp. 2508–2513. doi: 10.1109/WCNC.2007.467.

[98] E. A. Shams and A. Rizaner, "A novel support vector machine based intrusion detection system for mobile ad hoc networks," *Wirel. Networks*, vol. 24, no. 5, pp. 1821–1829, 2018, doi: 10.1007/s11276-016-1439-0.

[99] M. Martellini, S. Abaimov, S. Gaycken, and C. Wilson, *Information Security of Highly Critical Wireless Networks*. Cham: Springer International Publishing, 2017. doi: 10.1007/978-3-319-52905-9.

[100] J. Luo, C. Wei, H. Dai, and J. Yuan, "Robust LS-SVM-based adaptive constrained control for a class of uncertain nonlinear systems with time-varying predefined performance," *Commun. Nonlinear Sci. Numer. Simul.*, vol. 56, pp. 561–587, 2018, doi: 10.1016/j.cnsns.2017.09.004.

[101] G. Kumaresan and T. Adiline Macriga, “Group Key Authentication scheme for Vanet INtrusion detection (GKAVIN),” *Wirel. Networks*, vol. 23, no. 3, pp. 935–945, 2017, doi: 10.1007/s11276-016-1197-z.

[102] S. J. Horng *et al.*, “b-SPECS+: Batch verification for secure pseudonymous authentication in VANET,” *IEEE Trans. Inf. Forensics Secur.*, vol. 8, no. 11, pp. 1860–1875, Nov. 2013, doi: 10.1109/TIFS.2013.2277471.

[103] F. A. Ghaleb, M. A. Razzaque, and I. F. Isnin, “Security and privacy enhancement in VANETs using mobility pattern,” in *2013 Fifth International Conference on Ubiquitous and Future Networks (ICUFN)*, Jul. 2013, pp. 184–189. doi: 10.1109/ICUFN.2013.6614808.

[104] “CICFlowMeter (formerly ISCXFlowMeter),” *Canadian Institute for Cybersecurity*, 2017. https://www.unb.ca/cic/research/applications.html (accessed Aug. 16, 2021).

[105] R. Panigrahi and S. Borah, “A detailed analysis of CICIDS2017 dataset for designing Intrusion Detection Systems,” *Int. J. Eng. Technol.*, vol. 7, no. 3.24 Special Issue 24, pp. 479–482, 2018, [Online]. Available: https://www.sciencepubco.com/index.php/ijet/article/view/22797

[106] I. T. Jolliffe, *Principal Component Analysis*, 2nd ed. New York: Springer, 2002. doi: 10.1007/b98835.

[107] A. M. Martinez and A. C. Kak, “PCA versus LDA,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 23, no. 2, pp. 228–233, 2001, doi: 10.1109/34.908974.

[108] L. Lv, W. Wang, Z. Zhang, and X. Liu, “A novel intrusion detection system based on an optimal hybrid kernel extreme learning machine,” *Knowledge-Based Syst.*, vol. 195, 2020, doi: 10.1016/j.knosys.2020.105648.

[109] “Ubuntu 11.04.” http://old-releases.ubuntu.com/releases/11.04/ (accessed Oct. 25, 2019).

[110] F. Pedregosa *et al.*, “Scikit-learn: Machine Learning in Python,” *J. Mach. Learn. Res.*, vol. 12, no. 85, pp. 2825–2830, Jan. 2012, [Online]. Available: http://jmlr.org/papers/v12/pedregosa11a.html

[111] Z. Li, Z. Qin, K. Huang, X. Yang, and S. Ye, “Intrusion detection using convolutional neural networks for representation learning,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10638 LNCS, D. Liu, S. Xie, Y. Li, D. Zhao, and E.-S. M. El-Alfy, Eds. Cham: Springer International Publishing, 2017, pp. 858–866. doi: 10.1007/978-3-319-70139-4_87.

[112] T. Kim, S. C. Suh, H. Kim, J. Kim, and J. Kim, “An encoding technique for CNN-based network anomaly detection,” in *2018 IEEE International Conference on Big Data (Big Data)*, Dec. 2018, pp. 2960–2965. doi: 10.1109/BigData.2018.8622568.

[113] F. Chollet and others, “keras.” 2015. [Online]. Available: https://keras.io

[114] M. Abadi *et al.*, “TensorFlow: Large-scale machine learning on heterogeneous systems.” 2015. doi: 10.5281/zenodo.4724125.

[115] R. Tibshirani, “Regression shrinkage and selection via the Lasso,” *J. R. Stat. Soc. Ser. B*, vol. 58, no. 1, pp. 267–288, 1996, doi: 10.1111/j.2517-6161.1996.tb02080.x.

[116] A. E. Hoerl and R. W. Kennard, “Ridge regression: Biased estimation for nonorthogonal problems,” *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970, doi: 10.1080/00401706.1970.10488634.

[117] G. E. Hinton, A. Krizhevsky, and I. Sutskever, “System and method for addressing overfitting in a neural network,” US 9,406,017 B2, 2016 [Online]. Available: https://patents.google.com/patent/US9406017B2/en

[118] S. Akila Agnes and J. Anitha, “Analyzing the effect of optimization strategies in deep convolutional neural network,” in *Nature Inspired Optimization Techniques for Image Processing Applications*, J. Hemanth and V. E. Balas, Eds. Cham: Springer International Publishing, 2019, pp. 235–253. doi: 10.1007/978-3-319-96002-9_10.

[119] D. P. Kingma and J. L. Ba, “Adam: A method for stochastic optimization,” *3rd Int. Conf. Learn. Represent. ICLR 2015 - Conf. Track Proc.*, 2015, [Online]. Available: http://arxiv.org/abs/1412.6980

[120] M. Mehdipour Ghazi, B. Yanikoglu, and E. Aptoula, “Plant identification using deep neural networks via optimization of transfer learning parameters,” *Neurocomputing*, vol. 235, pp. 228–235, Apr. 2017, doi: 10.1016/j.neucom.2017.01.018.

[121] N. Tajbakhsh *et al.*, “Convolutional neural networks for medical image analysis: Full training or fine tuning?,” *IEEE Trans. Med. Imaging*, vol. 35, no. 5, pp. 1299–1312, 2016, doi: 10.1109/TMI.2016.2535302.

[122] W. Haider, G. Creech, Y. Xie, and J. Hu, “Windows based data sets for evaluation of robustness of host based Intrusion Detection Systems (IDS) to zero-day and stealth attacks,” *Futur. Internet*, vol. 8, no. 3, 2016, doi: 10.3390/fi8030029.

[123] K. H. Brodersen, C. S. Ong, K. E. Stephan, and J. M. Buhmann, “The balanced accuracy and its posterior distribution,” in *Proceedings - International Conference on Pattern Recognition*, 2010, pp. 3121–3124. doi: 10.1109/ICPR.2010.764.